

Topic 03: Information Extraction

[Conceptual Information Extraction](#)

[Goal](#)

[Our Example](#)

[System Architecture](#)

[Named Entity Recognition](#)

[Goal](#)

[Requirements](#)

[Method](#)

[Evaluation of NER systems](#)

[Information Extraction - Relationships](#)

[Information Extraction - Templates](#)

Conceptual Information Extraction

Goal

The goal of information extraction (IE) systems is the extraction of relevant relationships from unstructured data (i.e. text). It gets us almost all the way to semantic processing — programs which have a full understanding of text — but don't get too excited. We really want to identify all the people, places and organisations and a few important relationships among them. For example, a company which cares about mergers and acquisitions in general may scour the news for events such as one company acquiring another, so that company would build templates for these relationships. Using Information Retrieval techniques, we find documents relevant to the information being extracted. As such, IE overlaps with another NLP topic: question answering.

Here are some examples of IE systems:

- If someone sends email asking whether I want to have “lunch next Tuesday,” GMail offers to create a meeting for me... perhaps around the time when I would have lunch.
- When I perform a Google search (Information Retrieval task) for “[Zoe Kravitz](#),” I get a mini-bio as shown in Figure 1.

You may know about these examples because they've been tested and work most of the time. Not surprisingly, IE systems tend not to function as well as humans do. For example, the apartment listing in Figure 2 could fail on an IE system. (How many bedrooms are there?)

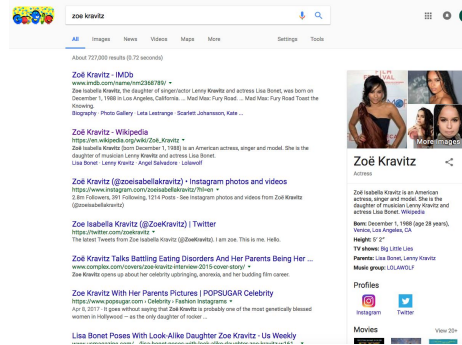


Figure 1: Google mini-bio for Zoe Kravitz

Some people may be tempted to create Information Retrieval systems for for searches, such as for apartments the one shown in Figure 2. This is likely an error for a few reasons:

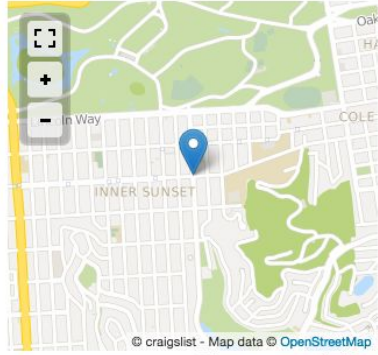

- The user may specify a range (eg. between \$3000-\$4000 per month for a 2+ bedroom), which is difficult to fit into the concept of the search term
- A property may have multiple locations / neighbourhoods; alternately, the user may specify a location and be willing to accept properties near this location

reply ☐ prohibited ^[2] Posted 2 days ago [print](#)

[prev](#) [next](#)

★ **\$1800 / 2br - 1 BR in 2 BR Apartment Inner Sunset SF** (inner sunset / UCSF)

image 1 of 2



© craigslist - Map data © OpenStreetMap

7th avenue at Judah
[\(google map\)](#)

2BR / 1Ba available jun 1

apartment
w/d in unit
carport

1 Bedroom available in a 2 Bedroom apartment near UCSF.

Close to restaurants, bars, markets, and Golden Gate Park. 2 minute walk to the N Judah. Hardwood floors and newly remodeled kitchen cabinets. Washer and dryer in-unit and parking is available (but not included with rent). Move in dates are flexible in either June or early July. Rent is \$1800 per month plus utilities and security deposit is \$1800.

You'd be sharing the apartment with another tenant in early 20's. The ideal roommate would keep the common areas neat and quiet on week nights, as I am up for work at 6am everyday.
If you're interested please message me a little about yourself and days you would be available to see the apartment.

Figure 2: Craigslist listing for a 1BR in a 2BR apartment. (Craigslist lists it under 2BR.)

Our Example

An article at CNN discussed the relationship between Mary Kay Letourneau and Vili Fualaau. In the 1990s, the two of them made news because they became a couple despite the age difference of 20 years, despite the fact that Mary Kay Letourneau was married and had four children at the time, and most sensationally, despite the fact that Vili Fualaau was 13 years old. Subsequently, the couple married and had children. They made the news again recently. The article is available at <http://www.cnn.com/2017/05/30/us/husband-files-for-separation-from-former-teacher-mary-kay-fualaau/>

Let's use the text version of this story as our example in building an IE system. The text from this story is available [here](#).

Information Extraction systems in general are used to extract “who did what to whom” information from text. In this case, assuming “VF” stands for Vili Fualaau and “MKL” stands for Mary Kay Fualaau (also known as Mary Kay Letourneau), we would like to see at minimum:

- VF filed for legal separation from MKL.
- MKL had four children (in 1996).
- MKL began an affair with VF in 1996.
- MKL gave birth to a(nother) child.
- MKL was released from prison in 2005.
- MKL married VF (in 2005?) (in Woodinville, WA).
- VF had two girls with his wife.

System Architecture

Since Information Extraction systems have been built, there have been many different types of relationships to be extracted. However, they tend to be built from similar sub-components. We assume that the input to an IE system is one (or more) documents on which the system will act. The output is a structured set of relationships, sufficient for populating a relationship graph, a database, etc.

The sub-components of the IE system may become procedures or steps in any system we build. These sub-components are:

- **Input Processor** — This sub-component handles text, making it is easy for “downstream” subcomponents to extract features and relationships.
- **Named Entity Recogniser** — This sub-component extracts the names of people, locations and organisations.

- **Relationship Extractor** — This sub-component establishes the relationships among the entities extracted above.
- **Template Filler** — Where needed, this sub-component changes relationships and entities into structured data.

The Input Processor handles the (unstructured) input in the way you are already familiar, Specifically, it performs tokenisation while being careful with punctuation. Since the input we have is in text format, we will only need the tools we've already used for this sub-component. But where the input is not in a format expected by the rest of the subsystems, the Input Processor performs conversion to an expected format. For example, if our input is speech, the Input Processor might run Automatic Speech Recognition and Sentence Segmentation.

Let's discuss the other sub-components in detail.

Named Entity Recognition

Goal

For Named Entity Recognition (NER), we expect our system to identify the names of people, organisations and locations. Despite the difficulty of this task, human performance on this task is quite high (~97-98%) for L1 language users.

Let's look into our document for an example.

The same month he would have celebrated his 12-year wedding anniversary, **Vili Fualaau** -- one half of the couple whose love story captivated national audiences -- filed for legal separation from his wife and former sixth-grade teacher, according to court documents obtained by **CNN**.

Mary Kay Fualaau, formerly **Letourneau**, was a married 34-year-old teacher and mother of four in **Seattle** in 1996 when she began an affair with **Fualaau**, her 13-year-old student.

In this text, we see that the four (4) named entities are highlighted. They are: *Vili Fualaau*, *CNN*, *Mary Kay Fualaau* (formerly Mary Kay Letourneau) and *Seattle*. Already, we see that this task may be difficult for at least two reasons. In this example specifically, there are two people with the same last name, so resolving the subject of the last named entity may require some care.

This becomes worse for very famous names such as JFK / John F. Kennedy (the name of a person, an airport, a bridge, several streets, a school, a space center, several aircraft carriers, and currency). Another, perhaps subtle, difficulty is that names have varying length. In the example above, one of our actors has a three-token name, another has a one-token name.

Requirements

Using the example above, an NER system must perform the following tasks:

- 1) The system must assign labels to each token in the input. The label must identify the type of entity — person (PER), location (LOC) or organisation (ORG). Some systems may have or require labels for geopolitical entity (GPE), facility (FAC) or vehicle (VEH). These labels must be attached regardless of the length of the tokens in the name of the entity.
- 2) The system must merge all aliases for an entity. In our document, for example, one entity has multiple aliases: *Vili Fualaau*, *Fualaau* and *Vili*. Each mention of his name must be resolved to this web of aliases.

Generally, NER systems are implemented as classifiers, taking a labelled document or set of documents. There are several labelling schemes. We will deal with two of them, called BIO and IO, which are popular.

Using the most restrictive set (PER, LOC and ORG for person, location and organisation respectively), BIO labels, each token is suffixed with a “/LABEL”, where LABEL is either “O” or one of the labels in our restrictive set, pre-pended with a “B-” at the start of a name or “I-” for a non-start of a name.

Here is an example from a subset of our running example above:

```
Mary/B-PER Kay/I-PER Fualaau/I-PER, /O formerly/O  
Letourneau/B-PER, /O was/O
```

We can see that “Mary” takes the the label “B-PER” because it is the start of the name sequence “Mary Kay Fualaau.” Other parts of the name take the label “I-PER” because it is “inside” the name. We also see that the name “Letourneau” takes the label “B-PER” in the same way and for the same reason “Mary” takes the tag in question. The name “Letourneau,” however, does not precede other tokens in a longer name, so it is both the beginning and end of the name sequence. In this case, we only use the “B-PER” label.

Notice that all tokens not somewhere in the name of a person, location or organisation are assigned the label “O.” This includes punctuation. Find a file which contains our example with BIO labels [here](#).

In the IO label scheme, our NER system accepts labels which only contain “O” labels or one of the labels in our restrictive set, sometimes (but often not) pre-pended with an “I-”. Here is the same example as above with IO labels:

```
Mary/PER Kay/PER Fualaa/PER, /O formerly/O  
Letourneau/PER, /O was/O
```

Find a file which contains our example with IO labels [here](#).

IO labels take fewer characters and have fewer categories to predict. For these reasons, we will use IO labels. However, there is a weakness of this scheme: if two entities in the same category find themselves together in a sentence, it is difficult to determine how they are related. An example could be:

```
... the Washington White House was ...
```

... where “Washington” and “White House” are two entities, but the three tokens would take the label “ORG.” Using BIO, because the starts of entities have unique labels, it would be clear how many entities exist in any sequence. However, this case is relatively rare.

Method

An NER system predicts a class label based on features of the token in question and the token sequence. The class labels are defined above. (We will use IO labels.)

NER systems have generally used features in Table 1 in predicting the class of a token. This list is not complete. Systems written for extraction of entities in specific domains may use more than these features. And some of the features in the table may have negligible or negative effects on system performance, in which case they should be excluded.

Feature	Description
Word “shape”	Does the token have mixed cases, punctuation, digits, etc.?
Nearby “shapes”	Shapes of the nearby tokens
POS tag	The part-of-speech of the token
Nearby POS tags	The part-of-speech tag(s) of preceding / subsequent tokens
Uppercase start	Boolean: does the token begin with an uppercase character?
All uppercase	Boolean: does the token contain only uppercase characters?
Hyphen	Boolean: does the token contain a hyphen?
Presence in a gazetteer	Boolean: is the token known to be an entity (PER, LOC, ORG)?
Prefix	Boolean: a particular character sequence to begin the token

Suffix	Boolean: a particular character sequence to end the token
--------	---

Table 1: Features in typical NER systems

Gazetteers

Traditionally, a gazetteer is an index for a map. In the case of NER prediction, we use gazetteers as known lists of entities. The CIA World Factbook contains an extensive section on [geographic names](#), perhaps the most extensive current gazetteer on current place names, areas and locations.

For names, we have already seen one such list: on Unix systems or on OS X machines, the file `/usr/share/dict/propernames` contains over 1300 common proper names. More extensive name lists may be available from the [Social Security Administration - baby names](#) (which has the advantages of a temporal component and much more complete data).

Gazetteers are not necessarily limited to lists of known names. We know, for example, that company names can end in certain keywords — “.com”, “.ly” and “.io”, for example. Some gazetteers have these bits of names for use in finding suffixes in tokens.

In the same vein, while some company names end with “Technologies,” others end in “Technology.” Levenshtein distance can handle many of these issues. Some company names like “Flickr” are deliberate misspellings, and person names are alternate spellings of others which may be present in a gazetteer. (For example Aiden has the same pronunciation but different spellings from its alternative forms: Aaden, Adan, Aden, Aidan, Aidyn, Aydan, Ayden, Aydin.) We have already seen that Soundex can be used to find sound-alike matches, and this definitely applies in this case.

Word Shape

Word shape features are perhaps the most unique in the area of NLP. An algorithm for generating a word shape is as follows:

- Uppercase characters are replaced with uppercase X
- Lowercase characters are replaced with x
- Digits are replaced with d
- Repeated sequences are collapsed into a single character

The shape of the word *O'Farrell* is X'Xx. Following the steps above, we arrive at the shape in the following way:

- Uppercase characters are converted to X → X'Xarrell
- Lowercase characters are converted to x → X'Xxxxxxx
- Digits would be replaced with d, but there are none
- Repeated characters are collapsed → X'Xx

Let's create a function to accept a token as a parameter and output the shape of the argument. Let's test the function with the following:

- *O'Farrell* → X'Xx
- U.S.A. → X.X.X.
- Guinea-Bissau → Xx-Xx

Part-of-speech tags

Many NLP toolkits have part-of-speech tagging modules. These modules assign a part-of-speech label to an input word, but the tags used may exceed the classic 8 parts of speech we may have been taught. By way of explanation: we know that words in the class "noun" describe "things." However, we make distinctions between "singular" and "plural" nouns in English. (For example, we attach different verb inflections.) Because we care about the distinction, we create different classes.

There are different tagsets in use. One of the most common tagsets was developed on a popular corpus called the Penn Treebank. [This page](#) lists the Penn Treebank POS set. We may notice that there already exist two proper noun tags — NNP (singular proper noun) and NNPS (plural proper noun). Obviously, this could be a strong signal of whether a token represents all or part of an entity, but it does not indicate what type of entity is predicted.

Aside: part-of-speech taggers are very accurate. (One person claimed that they can be 90% accurate simply by applying unambiguous labels where appropriate.) That said, they are not perfect. Some may assign an unknown label to our NER entities for lack of good training examples.

NLTK contains a part-of-speech tagger. It may be used on a tokenized sequence of text, so we normally create a pipeline for this. Here is a script in "main" to read a (text) file, tokenize it and tag the individual tokens:

```
import nltk
import sys

text = []
tagged_text = []
with open(input_file) as f:
    content = f.readlines()
    for line in content:
        text.extend(nltk.word_tokenize(line))
tagged_text = nltk.pos_tag(text)
print tagged_text
```

The full script (in main) is available [here, at GitHub](#). The output of running this script against our example is a list of tuples. Here is a sample of the output:


```
[ ..., ('The', 'DT'), ('same', 'JJ'), ('month', 'NN'), ('he', 'PRP'),  
('would', 'MD'), ('have', 'VB'), ('celebrated', 'VBN'), ('his',  
'PRP$'), ('12-year', 'JJ'), ('wedding', 'NN'), ('anniversary', 'NN'),  
(',', ','), ('Vili', 'NNP'), ('Fualaa', 'NNP'), ('--', ':'), ('one',  
'CD'), ('half', 'NN'), ('of', 'IN'), ...]
```

Each tuple contains the original token from the text file paired with its tag. The tag may be used as a feature of our NER system.

Stanford CoreNLP is another useful NLP toolkit which contains a POS tagger, along with other utilities. Unlike NLTK, it is implemented in Java (1.8), and it performs tokenization and tagging in one step with one of several different possible pre-trained models. One of these models, which considers only features from the current token and the three preceding tokens to the left, produces output as follows on the same text:

```
[ ... The_DT same_JJ month_NN he_PRP would_MD have_VB celebrated_VBN  
his_PRP$ 12-year_JJ wedding_NN anniversary_NN ,_, Vili_NNP  
Fualaa_NNP --_: one_CD half_NN of_IN ... ]
```

We can see that tags are attached to tokens with a “_” character, but the tagset is the same (Penn Treebank). In fact, the tag sequence for this portion of our sample text is identical to the NLTK-generated version.

In addition to a command-line interface and a Java-based API, Stanford CoreNLP also has a GUI which performs an identical function. Figure 3 shows an example of from our example in the GUI.

Tags from either POS module may be used as inputs to our classifier.

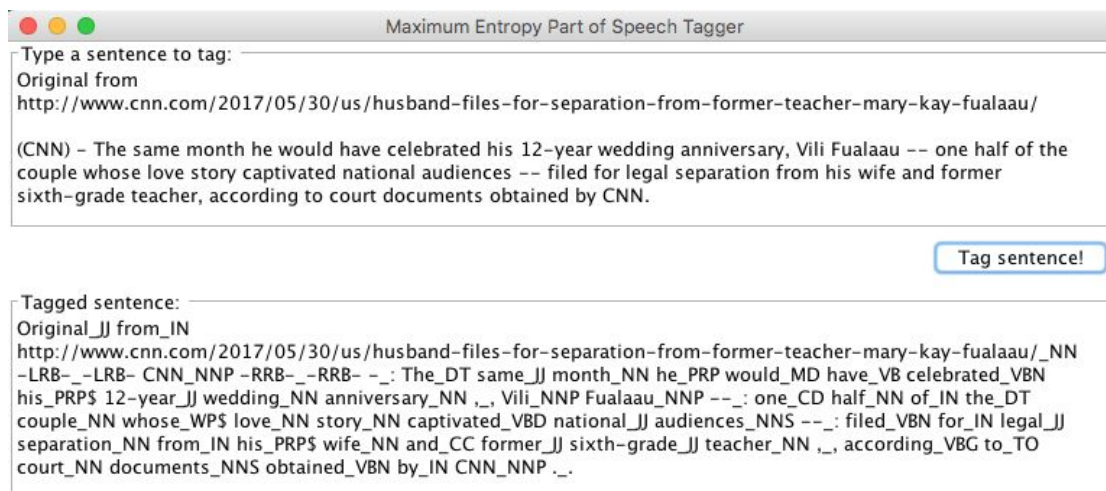


Figure 3: Stanford CoreNLP POS Tagger

Additional Features

In addition to the features above, we may also use the syntactic category of the token and surrounding tokens. Let's avoid these features at this time.

Classifier

We could on the language our system is asked to process being predictable. This predictability falls mostly in the realm of sequence modelling. As a result, one of two classifiers is generally recommended for tagging in NER sub-components: Conditional Random Fields (CRFs) and Maximum Entropy Markov Models (MEMMs). You should already know about MEMMs from Machine Learning; for this, we use Logistic Regression.

A Conditional Random Field is a classifier for which the label for a point considers the labels and features from surrounding data. We specifically care about a variant known as a linear-chain CRF since our assumption is that the prediction of a named entity depends on the previous (and possibly subsequent) labels and features. For a linear-chain CRF, we make the additional assumption that probabilities for generating labels may vary across the positions in the sequence of (hidden) states depending on the features.

That said, there is no implementation of a CRF in the most popular packages (sklearn and weka), so we make do with prediction via our usual set of classification algorithms (SVM, RandomForest, etc.) run over CRF-inspired features. For the sake of purity, we can use CRF packages, including: [CRFSuite](#) or [Wapiti](#).

CRFs have typically generated the highest-performing NER subsystems, but they are definitely not the only classifier used. Many academic and industry classifiers used the full set of classification tools available to us, including: HMMs, Decision Trees and SVMs.

Evaluation of NER systems

NER systems are known to exhibit a performance decrease when moving across domains (eg. from sports to business) or when moving across genres (eg. from blogs to literature). Because of this, some Information Extraction systems in general (and Named Entity Recognition systems in particular) are designed for specific document formats or web sites (using the template of the document or site) or are genre specific (eg. only for resumes). These systems often generate the highest performance and are difficult to compare against IE or NER systems which are less specific.

Scoring NER performance can be done with recall, precision and F_1 measure, much like for Information Retrieval systems. In fact, some NER systems are evaluated this way. Let's discuss one problem this type of evaluation. Assume that our target looks like the following:

Unlike Robert[PER], John Briggs Jr[PER] contacted Wonderful Stockbrokers Inc[ORG] in New York[LOC] and instructed them to sell all his shares in Acme[ORG].

... and let's assume that a system produces the following hypotheses:

Unlike[LOC] Robert, John Briggs Jr[ORG] contacted Wonderful Stockbrokers[ORG] Inc in New York[PER] and instructed them to sell all his shares in Acme[ORG].

We can see that the system produced 5 errors. In order, these errors are:

- “Unlike” is hypothesized where there should be no entity.
- “Robert” was missed.
- “John Briggs Jr” is has the incorrect label (should be [PER]).
- “... Stockbrokers...” has the wrong boundaries.
- “New York” has the wrong boundaries and the wrong label.

Depending on how we decide to define correctness, it is possible for systems using the precision/recall/ F_1 metrics to receive scores lower than 0, and this potential for unbounded (and poor) performance has driven others to explore a different set of metrics. It would be good to give “partial credit” for getting some hypotheses or some parts of hypotheses correct.

The Message Understanding conference (MUC) generates a score on two axes: ability to find the right text (TEXT) and ability to find the right type (TYPE). A system gets credit for TEXT if it finds the correct start and end boundaries for an entity, no matter whether the type it finds is a good match. Likewise, a system gets credit for TYPE if it hypothesises the correct type, regardless of boundaries as long as there are no overlaps. For each of TEXT and TYPE, three measures are counted:

- 1) The number of correct answers (COR)
- 2) The number of actual system hypotheses (ACT)
- 3) The number of possible entities in the reference (POS)

Given all of this, the MUC score is the micro-averaged F-measure (MAF), which is the F_1 measure over all entity slots on both axes. Specifically:

- Precision = COR / ACT
- Recall = COR / POS

For the example above, COR = 4 (2 TEXT + 2 TYPE) ... ACT = 10 (5 TEXT + 5 TYPE) ... POS = 10 (5 TEXT + 5 TYPE) ... so precision and recall are each 0.4, making MAF = 0.4 as well.

The MUC series was sponsored by NIST. It was abandoned almost 2 decades ago and was replaced by the Automatic Content Extraction (ACE) series. There is a separate ACE evaluation in which weights are assigned to different entity types. (Person entities could count more than location entities, for example.) The ACE metric also incorporates co-references. We will not use the ACE metric because of its complexity and also because it's less popular than MUC.

Aside: the ACE series was also abandoned about a decade ago for the [Text Analysis Conference](#), with two interesting foci: KBP (knowledge base population) and ADR (adverse drug reactions). If you're interested, this is an excellent starting point for a project.

Information Extraction - Relationships

The recognition of relationships between named entities is the next important subtask of Information Extraction, and it constitutes the next sub-system. Given text such as below, which is the headline from [an article in the World Socialist Web Site](#), we would like to infer that it is not the city of Washington which accused the country of Russia. Instead, we want to understand that the (US) government in Washington accused the Russian government (or military) of a bombing which occurred in Syria.

Washington accuses Russia of bombing US-backed forces in Syria

This kind of relationship, known as metonymy, substitutes the name of a part for the name of a whole. The Google definition contains a great example:

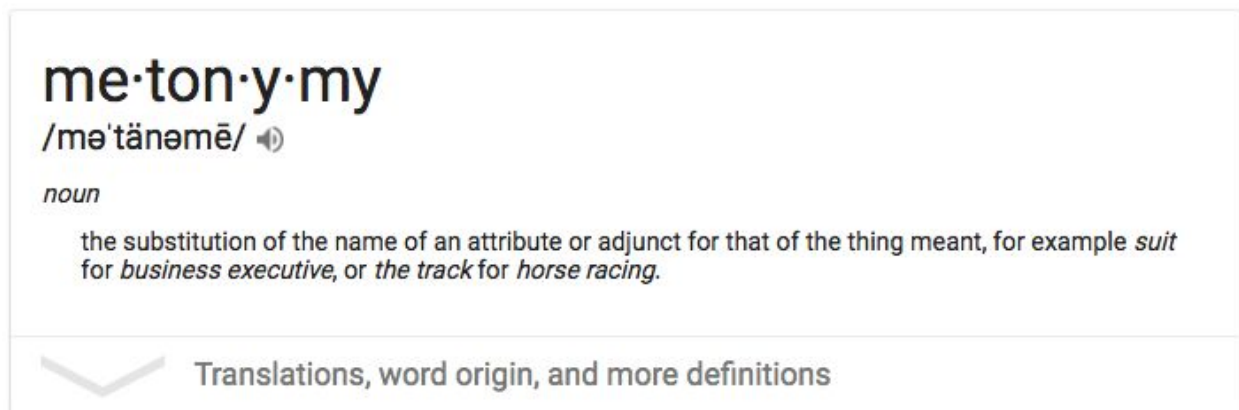


Figure 4: Google definition of metonymy, with "suit" example

This kind of relationship is difficult to extract because it assumes a lot of cultural knowledge which even expert speakers of the language may not possess. (This is very similar to a metaphor. For example, French people sometimes refer to their country as a hexagon because of its overall shape. Some people who speak French well but learned outside France may be unaware of this.)

While metonymy and metaphor constitute the more difficult relationships, some relationships are easier to infer from the text. One of the easiest of these relationships is called apposition. Here is an example:

[Tim Cook](#), CEO of [Apple](#), ...

In this example, we see that Tim Cook is listed as the CEO of Apple, Inc. And we know this despite the lack of the verb separating the two named entities. Generally, we know that where we see this kind of grammar structure, the two entities are in an “is_a” relationship with each other — Tim Cook is a CEO of Apple... or Tim Cook is the CEO of Apple. In English, the typical appositional template for Information Extraction is:

PER, Noun Phrase (NP), ⇒ PER is_a NP

Where the Noun Phrase (NP) contains another named entity, we can establish the relationship between entities with some confidence.

What other kinds of relationships can we discover knowing nothing about the entities themselves? There are many. For example, if we see the sentence fragment:

“... any kind of a clustering algorithm, such as DBSCAN or Mean Shift.”

... we can infer that DBSCAN and Mean Shift are examples clustering algorithms even if we did not know what a clustering algorithm is. This type of relationship (one example from a class of things... or member ⇒ group) is called a hyponym. The opposite relationship, from class to specific example, is a hypernym.

These are all examples of unsupervised relationship extraction. It is unsupervised because we start with no examples or knowledge of relationships, just with knowledge of lexical or phrase structure. The inference of relationships based solely on lexical structure was an approach which started more than 25 years ago with Marti Hearst in Berkeley. This is one of the bases of ontologies such as WordNet which contains a large (but not exhaustive) set of these relationships.

Hearst’s work continued into a semi-supervised approach. This is the most typical type of learning for relationship extraction in a practical or commercial setting. In it, our system starts with some seed knowledge of entity relationships. For example, our system may be given an existing relationship like < “writer”, “Isaac Asimov”, “The Robots of Dawn” >. With this knowledge, the relationship extractor may seek the various ways this relationship may be expressed, such as “Isaac Asimov wrote The Robots of Dawn” or “The writer of The Robots of Dawn, Isaac Asimov...”. Web pages, such as [the Wikipedia page for the book](#) gives us a some alternate ways of expressing the same relationship.

Building on that, one popular method of performing this kind of relationship extraction is using the web as a corpus (WAC). For example, knowing the relationship < “writer”, “Isaac Asimov”, “The Robots of Dawn” >, we may arrive at a book seller’s page, [like this one at Amazon](#). From here, we can acquire a few dozen more <author, title> pairs and use those to re-start an iteration.

The problem with iterating from a known set of examples is that for each iteration, we lose confidence in our extracted information. This is known as **semantic drift**: an error in a pattern may lead to an erroneous relationship, which may lead to additional errors in patterns.

Given a current set of tuples, T , a document collection, D , and a proposed pattern, p , we may measure our confidence based on a pattern’s performance based on two factors:

- Hits = the set of all tuples in T that match p while looking in D
- Finds = the total set of tuples that p find in D

... which we use to determine the confidence

$$\text{Conf}_{\text{RlogF}}(p) = (\text{hits}_p / \text{finds}_p) \log(\text{finds}_p)$$

This metric may be normalised to produce a confidence in the proposed new pattern. Also, the metric may be thresholded to determine whether to adopt the new pattern.

We have additional resources which we employ. We’ve discussed one of them many times already: WordNet. For the macabre word “kill,” WordNet defines zero, one or a number of similar concepts or similar words, called synsets, such as murder, assassinate, strangle, suffocate, etc. Much like with the semi-supervised approach, we may equate any of the words or phrases in a synset with some confidence.

Information Extraction - Templates

One very popular purpose of information extraction systems is the population of some structure, a SQL or NoSQL database being the most popular but definitely not the only structure. These are particularly popular with organisations (companies, etc.) interested in performing analysis on unstructured data. A typical solution for this kind of Information Extraction requirement is the creation of a template. (We can think of a template as a database schema or an object-relational mapping, ORM, which may not end up in a database.)

Using this solution, one or more templates are created for describing “prototype” events. Recall our motivating example:

```
The same month he would have celebrated his 12-year
wedding anniversary, Vili Fualaau -- one half of the
couple whose love story captivated national audiences --
filed for legal separation from his wife and former
```

sixth-grade teacher, according to court documents obtained by CNN.

Mary Kay Fualaau, formerly Letourneau, was a married 34-year-old teacher and mother of four in Seattle in 1996 when she began an affair with Fualaau, her 13-year-old student.

Our template classes may be relationships between people, which could be composed from events such as birth, death, graduation, befriending, starting a relationship, marriage, etc. A filled template based on the first paragraph of the above may look like:

LEGAL SEPARATION:

REQUESTOR: Vili Fualaau
SUBJECT: Mary Kay Fualaau
DATE: UNKNOWN

The filled templates of the other life events would clearly look different. For example, a “Birth” event may include parents, health care professionals, organisation (eg. hospital), etc. Some templates may have default values for certain slots. In the above, we see that the DATE value is “UNKNOWN.”

The current state-of-the-art technique is to break this sub-task into two phases. In the first phase, a template is selected for population. In the second phase, names and events are used to populate the template.

We assume that we have multiple templates corresponding to the information which is interesting to the organisation. Of these, the template which is appropriate must be selected. This is called **event recognition**, **slot filling** or sometimes **template recognition**. Generally, the correct template is selected based on the features of the text being considered, in other words, this is a text classification task. For example, words like “married,” “marriage,” “wedding,” or “nuptials” are strong indications of a marriage event. Clearly, one (or more) of the templates to be filled should be a NULL template — i.e. text with no corresponding template.

Generally, the template selected has one or more roles for entities, events and other interesting artifacts. For example, in our LEGAL SEPARATION template, the person requesting the separation (the REQUESTOR) is Vili Fualaau. Filling the empty slots in a template with named entities, events and other interesting artifacts constitutes the second phase. This is known as **role-filler extraction**. This is also a classification task in which the most likely hypothesis of some known entities (and possibly a NULL entity) is used to fill a role.