Relation extraction

Relation extraction plays an important role in extracting **structured information** from unstructured sources such as raw text. One may want to find interactions between drugs to build a medical database, understand the scenes in images, or extract relationships among people to build an easily searchable knowledge base.

For example, let's assume we are interested in marriage relationships. We want to automatically figure out that "Michelle Obama" is the wife of "Barack Obama" from a corpus of raw text snippets such as "Barack Obama married Michelle Obama in...". A naive approach would be to search news articles for indicative phrases, like "married" or "XXX's spouse". This would yield some results, but human language is inherently ambiguous, and one cannot possibly come up with all phrases that indicate a marriage relationship. A natural next step would be to use machine learning techniques to extract the relations. If we have some labeled training data, such as examples of pairs of people that are in a marriage relationship, we could train a machine learning classifier to automatically learn the patterns for us. This sounds like a great idea, but there are several challenges:

- How do we disambiguate between words that refer to the same entity? For example, a sentence may refer to "Barack Obama" as "Barack" or "The president".
- How do we get training data for our machine learning model?
- How do we deal with conflicting or uncertain data?

Entity linking

Before starting to extract relations, it is a good idea to determine which words refer to the same "object" in the real world. These objects are called **entities**. For example, "Barack", "Obama" or "the president" may refer to the entity "Barack Obama". Let's say we extract relations about one of the words above. It would be helpful to combine them as being information about the same person. Figuring out which words, or **mentions**, refer to the same **entity** is a process called **entity linking**. There are various techniques to perform entity linking, ranging from simple string matching to more sophisticated machine learning approaches. In some domains we have a database of all known entities to link against, such as a dictionary of all countries. In other domains, we need to be open to discovering new entities.

Dealing with uncertainty

Given enough training data, we can use machine learning algorithms to extract entities and relations we care about. There is one problem left: human language is inherently noisy. Words and phrases can be ambiguous, sentences are often ungrammatical, and spelling mistakes are frequent. Our training data may have errors in it as well, and we may have made mistakes in the entity linking step. This is where many machine learning approaches break down: they treat training or input data as "correct" and make predictions using this assumption.

DeepDive makes good use of uncertainty to improve predictions during the probabilistic inference step. For example, DeepDive may figure out that a certain mention of "Barack" is only 60% likely to actually refer to "Barack Obama", and use this fact to discount the impact of that mention on the final result for the entity "Barack Obama". DeepDive can also make use of domain knowledge and allow users to encode rules such as "If Barack is married to Michelle, then Michelle is married to Barack" to improve the predictions.

Probabilistic inference and factor graphs

This documents presents a high-level overview of **probabilistic inference** and an introduction to **factor graphs**, a model used by DeepDive to perform probabilistic inference.

Probabilistic inference is the task of deriving the probability of one or more random variables taking a specific value or set of values. For example, a Bernoulli (Boolean) random variable may describe the event that John has cancer. Such a variable could take a value of 1 (John has cancer) or 0 (John does not have cancer). DeepDive uses probabilistic inference to estimate the probability that the random variable takes value 1: a probability of 0.78 would mean that John is 78% likely to have cancer.

Factor graphs

A **factor graph** is a type of probabilistic graphical model. A factor graph has two types of nodes:

- **Variables**, which can be either *evidence variables* when their value is known, or *query variables* when their value should be predicted.
- **Factors**, which define the relationships between variables in the graph. Each factor can be connected to many variables and comes with a **factor function** to define the relationship between these variables. For example, if a factor node is connected to two variables nodes `A` and `B`, a possible factor function could be `imply(A,B)`, meaning that if the random variable `A` takes value `1`, then so must the random variable `B`. Each *factor function* has a ***weight*** associated with it, which describes how much influence the factor has on its variables in relative terms. In other words, the weight encodes the confidence we have in the relationship expressed by the factor function. If the weight is high and positive, we are very confident in the function that the factor encodes; if the weight is high and negative, we are confident that the function is incorrect. The weight can be learned from training data, or assigned manually.

A **possible world** is an assignment to every variable in a factor graph. The possible worlds are not usually equiprobable; rather, each possible world has a different probability. The probability of a possible world is proportional to a weighted combination of all factor functions in the graph, evaluated at the assignments specified by the possible world. The weights can be assigned statically or learned automatically. In the latter case, some *training data* is needed. Training data define a set of possible worlds and, intuitively, the learning process chooses the weights by maximizing the probabilities of these possible worlds.

Marginal inference is the task of inferring the probability of one variable taking a particular value. Using the law of total probability, it is straightforward to express this probability as the sum of the probabilities of possible worlds that contain the requested value for that variable.

Exact inference is an intractable problem on factor graphs, but a commonly used method in this domain is **Gibbs sampling**. The process starts from a random possible world and iterates over each variable `v`, updating its value by taking into account the factor functions of the factors that `v` is connected to and the values of the variables connected to those factors (this is known as the *Markov blanket* of `v`). After enough iterations over the random variables, we can compute the number of iterations during which each variable had a specific value and use the ratio between this quantity and the total number of iterations as an estimate of the probability of the variable taking that value.

Inference in DeepDive

DeepDive allows the user to write inference rules to specify how to create the factor graph. A rule expresses concepts like "If John smokes then he is likely to have cancer" and, in other words, describes the factor function of a factor and which variables are connected to this factor. Each rule has a *weight* (either computed by DeepDive or assigned by the user), which represents the confidence in the correctness of the rule. If a rule has a high positive *weight*, then the variables appearing in the rule are likely to take on values that would make the rule evaluate to true. In the above example, if the rule "If John smokes then he is likely to have cancer" has a high weight and we are **sure** that John smokes, then we are also reasonably confident that John has cancer. However, if we are not sure whether or not John smokes, then we can not be sure about him having cancer either. In the latter case, both "John does not have cancer," and "John has cancer" would make the rule evaluate to true.

This is a subtle but very important point. Contrary to many traditional machine learning algorithms, which often assume that prior knowledge is exact and make predictions in isolation, DeepDive performs ***joint inference***: it determines the values of all events at the same time. This allows events to influence each other if they are (directly or indirectly) connected through inference rules. Thus, the uncertainty of one event (John smoking) may influence the uncertainty of another event (John having cancer). As the relationships among events become more complex this model becomes very powerful. For example, one could imagine the event "John smokes" being influenced by whether or not John has friends who smoke. This is particularly useful when dealing with inherently noisy signals, such as human language.

Distant supervision

Most machine learning techniques require a set of **training data**. A traditional approach for collecting training data is to have **humans label** a set of documents. For example, for the marriage relation, human annotators may label the pair "Bill Clinton" and "Hillary Clinton" as a positive training example. This approach is expensive in terms of both time and money, and if our corpus is large, will not yield enough data for our algorithms to work with. And because humans make errors, the resulting training data will most likely be noisy.

An alternative approach to generating training data is **distant supervision**. In distant supervision, we make use of an already existing database, such as Freebase or a domain-specific database, to collect examples for the relation we want to extract. We then use these examples to automatically generate our training data. For example, Freebase contains the fact that Barack Obama and Michelle Obama are married. We take this fact, and then

label each pair of "Barack Obama" and "Michelle Obama" that appear in the same sentence as a positive example for our marriage relation. This way we can easily generate a large amount of (possibly noisy) training data. Applying distant supervision to get positive examples for a particular relation is easy, but generating negative examples is more of an art than a science.

Generating negative evidence

In common relation extraction applications, we often need to generate negative examples in order to train a good system. Without negative examples the system will tend to classify all the variables as positive. However for relation extraction, one cannot easily find enough golden-standard negative examples (ground truth). In this case, we often need to use distant supervision to generate negative examples.

Negative evidence generation can be somewhat an art. The following are several common ways to generate negative evidence, while there might be some other ways undiscussed:

- incompatible relations
- domain-specific knowledge
- random sampling

Incompatible relations

Incompatible relations are other relations that are always, or usually, conflicting with the relation we want to extract. Say that we have entities x and y, the relation we want to extract is A and one incompatible relation to A is B, we have:

$$B(x, y) \Rightarrow \text{not } A(x, y)$$

As an example, if we want to generate negative examples for "spouse" relation, we can use relations incompatible to spouse, such as parent, children or siblings: If x is the parent of y, then x and y cannot be spouses.

Domain specific rules

Sometimes we can make use of other domain-specific knowledge to generate negative examples. The design of such rules are largely dependent on the application.

For example, for spouse relation, one possible domain-specific rule that uses temporal information is that "people that do not live in the same time cannot be spouse". Specifically, if a person `x` has `birth_date` later than `y`'s `death_date`, then `x` and `y` cannot be spouses.

Random samples

Another way to generate negative evidence is to randomly sample a small proportion among all the variables (people mention pairs in our spouse example), and mark them as negative evidence.

This is likely to generate some false negative examples, but if statistically variables are much more likely to be false, the random sampling will work.

For example, most people mention pairs in sentences are not spouses, so we can randomly sample a small proportion of mention pairs and mark them as false examples of spouse relation.

To see an example of how we generate negative evidence in DeepDive, refer to the example application tutorial.

DeepDive system overview and terminologies

This document presents an overview of DeepDive as a system. It assumes that you are familiar with some general concepts like inference and factor graphs, relation extraction, and distant supervision. It describes each step performed during the execution of a DeepDive application:

Extraction

The extraction step is a **data transformation** during which DeepDive processes the data to extract entities, perform entity linking, feature extraction, distant supervision, and any other task necessary to create the variables on which it will then perform inference, and, if needed, to generate the training data used for learning the factor weights. The tasks to perform during extraction are specified by defining extractors, which can also be user-

defined functions (UDFs). The results of extraction are stored in the application database and will be then used to build the factor graph according to rules specified by the user.

Factor graph grounding

DeepDive uses a factor graph to perform inference. The user writes SQL queries to instruct the system about which variables to create. These queries usually involve tables populated during the extraction step. The variable nodes of the factor graph are connected to factors according to inference rules specified by the user, who also defines the factor functions which describe how the variables are related. The user can specify whether the factor weights should be constant or learned by the system (refer to the 'Writing inference rules' document).

Grounding is the process of writing the graph to disk so that it can be used to perform inference. DeepDive writes the graph to a set of five files: one for variables, one for factors, one for edges, one for weights, and one for metadata useful to the system. The format of these file is special so that they can be accepted as input by our sampler.

Weight learning

DeepDive can learn the weights of the factor graph from training data that can be either obtained through distant supervision or specified by the user while populating the database during the extraction phase. The main general way for learning the weights is maximum likelihood.

The learned weights are then written to a specific database table so that the user can inspect them during the calibration of the process.

Inference

The final step consists in performing marginal inference on the factor graph variables to learn the probabilities of different values they can take over all possible worlds. DeepDive uses our high-throughput DimmWitted sampler to perform Gibbs sampling, i.e., to go through many possible worlds and to estimate the probabilities. The sampler takes the grounded graph (i.e., the five files written during the grounding step) as input, together with a number of arguments to specify the parameters for the learning procedure. The results of the inference step are written to the database. The user can write queries to analyze the results. DeepDive also provides calibration data to evaluate the accuracy of the inference.

Knowledge base construction

Knowledge base construction (KBC) is the process of populating a knowledge base (KB) with facts (or assertions) extracted from data (e.g., text, audio, video, tables, diagrams, ...). For example, one may want to build a medical knowledge base of interactions between drugs and diseases, a Paleobiology knowledge base to understand when and where did dinosaurs live, or a knowledge base of people's relationships such as spouse, parents or sibling. DeepDive can be used to facilitate KBC.

As a concrete example, one may use DeepDive to build an application to extract spouse relations from sentences in the Web. Figure 1 below shows such an application, where the input consists of sentences like "U.S President Barack Obama's wife Michelle Obama ...", and DeepDive's output consists of tuples in an `has_spouse` table representing the fact that, for example, "Barack Obama" is married to "Michelle Obama". DeepDive also produces a probability associated to the fact, representing the system's confidence that the fact is true (refer to the 'Probabilistic inference' document for more details about this concept).

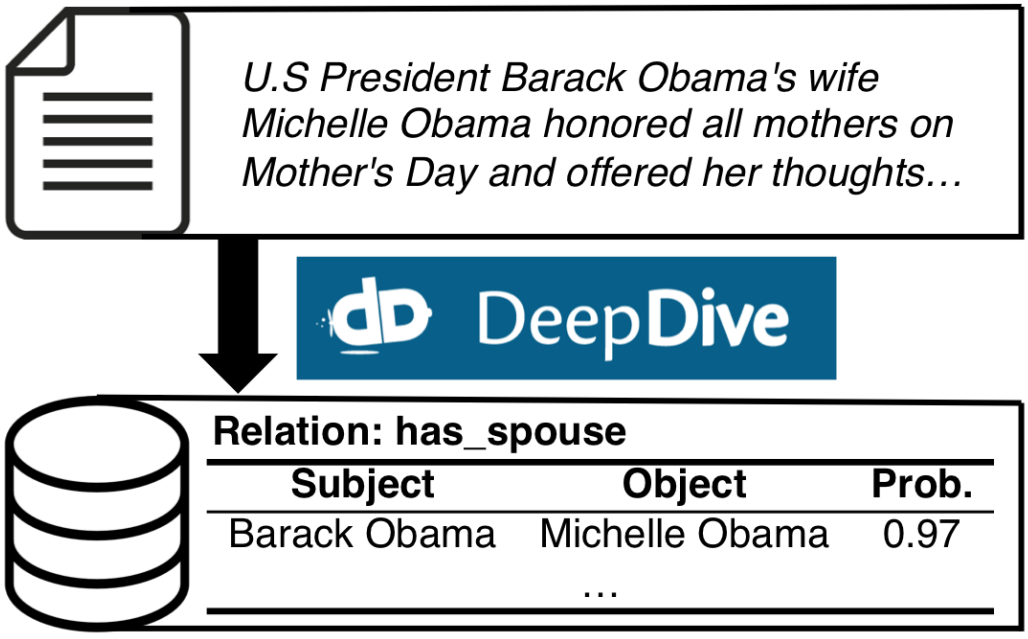

Figure 1: An example KBC application

KBC terminology

KBC uses specific terms to refer to the objects it manipulates. The following is a list of the most important ones

- An **entity** is an object in the real world, such as a person, an animal, a time period, a location, etc. For example, the person "Barack Obama" is an entity.
- **Entity-level data** are relational data over the domain of conceptual entities (as opposed to language-dependent mentions), such as relations in a knowledge base, e.g. "Barack Obama" in Freebase.
- A **mention** is a reference to an entity, such as the word "Barack" in the sentence "Barack and Michelle are married".
- **Mention-level data** are textual data with mentions of entities, such as sentences in web articles, e.g. sentences in New York Times articles.
- **Entity linking** is the process to find out which entity is a mention referring to. For example, "Barack", "Obama" or "the president" may refer to the same entity "Barack Obama", and entity linking is the process to figure this out.
- A **mention-level relation** is a relation among mentions rather than entities. For example, in a sentence "Barack and Michelle are married", the two mentions "Barack" and "Michelle" has a mention-level relation of `has_spouse`.
- An **entity-level relation** is a relation among entities. For example, the entity "Barack Obama" and "Michelle Obama" has an entity-level relation of `has_spouse`.

The relationships between these concepts (also known as the data model) is represented in Figure 2 below.

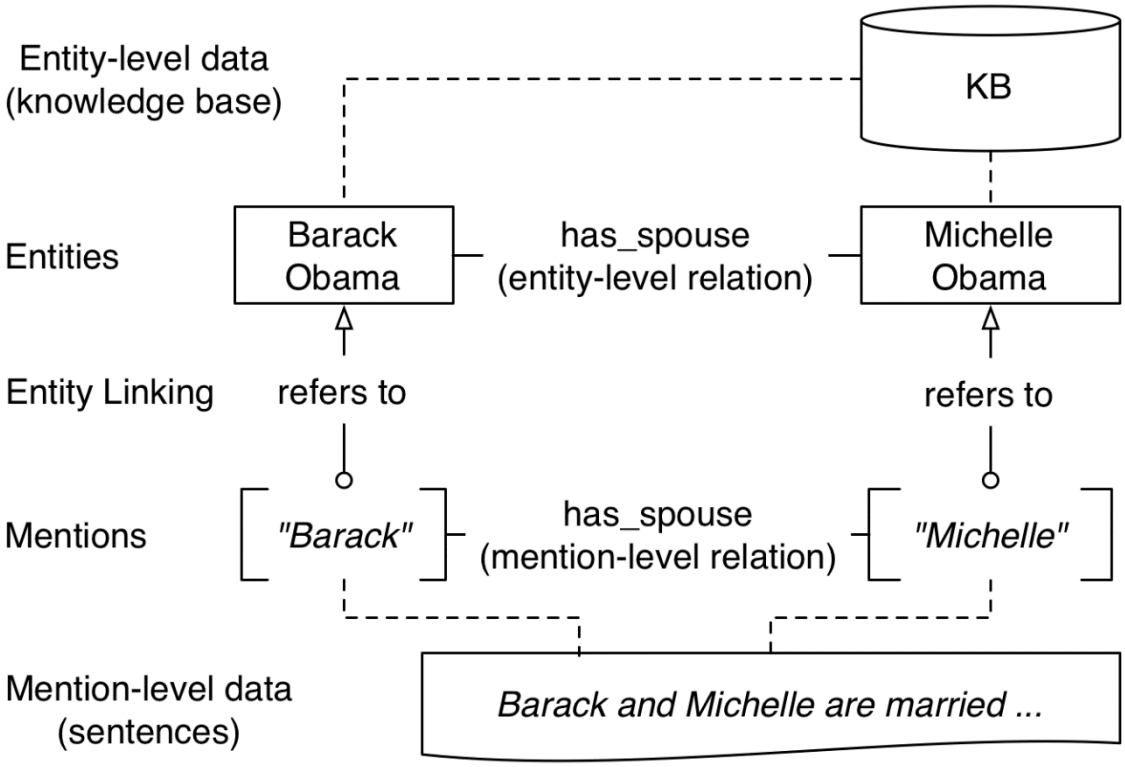

Figure 2: Data model for KBC

KBC data flow

In a typical KBC application, the **input** of the system is a collection of raw articles in text format, while the **output** of the system is a database (the KB) containing the desired (entity-level or mention-level) relations.

As explained in the System overview, the steps to obtained the output are the following:

1. data preprocessing
2. feature extraction
3. factor graph generation by declarative language
4. statistical inference and learning

Specifically:

1. In the **data preprocessing** step, DeepDive takes input data (articles in text format), loads them into a database, and parse the articles to obtain sentence-level information including words in each sentence, POS tags, named entity tags, etc.
2. In the **feature extraction** step, DeepDive converts input data into relation signals called **evidence**, by running extractors written by

developers. Evidence includes: (1) candidates for (mention-level or entity-level) relations; (2) (linguistic) features for these candidates.

3. In the next step, DeepDive uses evidence to **generate a factor graph**. To instruct DeepDive about how to generate this factor graph, developers use a SQL-like declarative language to specify inference rules.

4. In the next step, DeepDive automatically performs **learning and statistical inference** on the generated factor graph. learning, the values of factor weights specified in inference rules are calculated. These weights represent, intuitively, the confidence in the rule. During the inference step, the marginal probabilities of the variables are computed, which, in some cases, can represent the probability that a specific fact is true.

After inference, the results are stored in a set of database tables. The developer can **get results** via a SQL query, **check results** with a calibration plot, and perform **error analysis** to improve results.

As an example of this data flow, Figure 3 demonstrates how a sentence "U.S President Barack Obama's wife Michelle Obama..." go through the process (In this figure, we only highlight step 1--3):

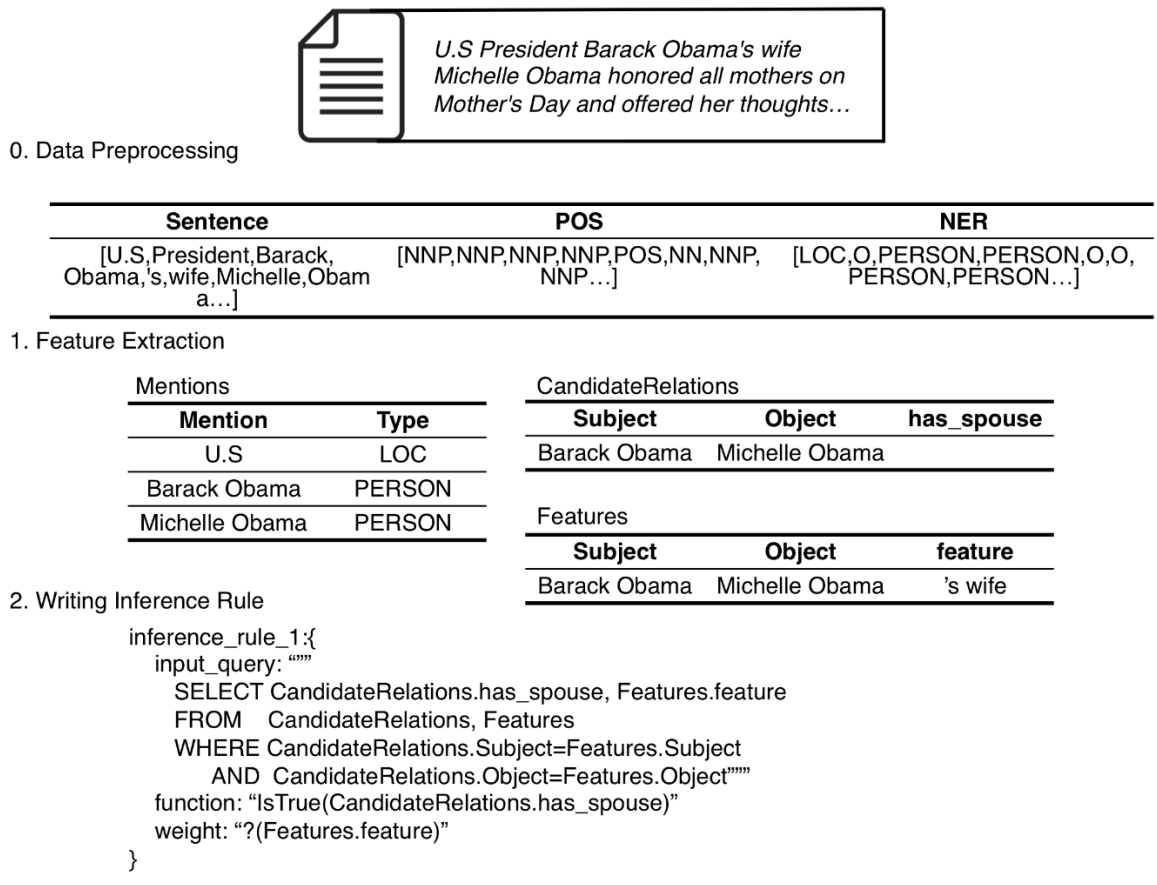

Figure 3: Data flow for KBC

In the example above:

1. During data preprocessing, the sentence is processed into words, POS tags and named entity tags;
2. During feature extraction, DeepDive extracts (1) mentions of person and location, (2) candidate relations of `has_spouse`, and (3) `feature` of candidate relations (such as words between mentions).
3. In factor graph generation, DeepDive use rules written by developers (like `inference_rule_1` above) to build a factor graph.