Generating Knowledge Graph (KG) from unstructured Text Data coupled with Reinforcement Learning (RL)

By Biswapratap Chatterjee

The problem

EY has a lot of very important but unstructured text data like invoices, broker statements etc. Harnessing information from them has always been a daunting task. Information has been harnessed until now either by manual effort or by very strict rule-based parsers. Recently, there has been considerable effort for automatic extraction of information from unstructured text data using machine learning.

Current State of the Art Solution

The core of the current ML effort is around supervised classification of information into pre-labelled classes like – invoice date, client address, invoice date etc. The text information is converted into contextual embeddings like Sequence and Position of the information with respect to the document. Although, there has been some success in this approach, but one can clearly point out two major shortfalls in this approach.

1. Finite set of class labels.
2. Manual Labelling of classes.

Finite set of class labels means the solution is heavily limited by the different variety of classes a human can uniquely identify. Even though the model has a rich contextual embedding of the information but due to the finite set of class labels it is impossible to identify any new information that comes in the way of the model.

Manual labelling of classes is a very well-known in supervised learning problem. Normally, supervised approaches are taken for problems where there is a rich abundancy of already existing labelled data. But, the moment manual labelling becomes part of the solution it defeats the purpose of using supervised learning. Apart, from this we can find two more issues related to manual labelling –

1. The model can never become better than the human who has manually labelled the data. Any mistakes or bad quality labelling done by the human gets sucked and propagated throughout the model.
2. It's entirely impractical to expect manual labelling complex text information in huge numbers in finite time. And with low number of labelled information, the sample population for model train is too small, which poses a huge risk of overfitting the model.

Proposed Solution

The proposed solution is an entirely novel approach. Instead of supervised learning, we propose an entirely unsupervised learning solution coupled with reinforcement learning**. The goal is to extract maximum correct information from any kind of unstructured text with zero human intervention.** Although, the goal may sound to be just too perfect to achieve, we believe we can achieve good quality output following the hypothesis.

The Hypothesis

The vector math analogy of word vector embeddings in Word2Vec by Mikolov et al lays the foundation of the hypothesis.

Say, a word $\mathbf{w_1}$ has a vector embedding $\mathbf{e_1}$ calculated using genism Word2Vec, and similarly another word $\mathbf{w_2}$ has a vector embedding $\mathbf{e_2}$.

So, the relationship vector between w_1 and w_2 can be calculated as –

$$\mathbf{r_{12} = e_1 - e_2}$$

where, r_{12} is the relationship vector between w_1 and w_2.

Hypothesis 1

If w_1 and w_2 represents a valid piece of information like –

w_1 = *Invoice Number*

w_2 = *123456789 (sample invoice number)*

Then, such a pair must exist across multiple invoice documents, with a similar relationship vector r_{12}.

Hypothesis 2

Using a valid relationship vector r_{12} as a seed we can generate other valid latent relationships within invoice documents using an actor-critic based reinforcement learning agent.

Seed Relationship

What is a seed relationship? A seed relationship is something that we already know like “invoice number is”. A seed relationship helps boot-strap the algorithm. How? As we know for a seed relationship –

$w_1 - r_{seed} \rightarrow w_2$

where, w_1 will be Invoice Number n-gram string, w_2 will be the invoice number by value and r_{seed} will be the relationship vector representing “invoice number is”. So, in the first invoice document after getting the word vector embeddings, we get the word vector representation of the n-gram string “Invoice Number” and the actual invoice number by value (say) “123456789” –

```
text = PDFExtractor.extract('Invoice101.pdf')
embeddings, word_corpus = Word2VecEmbedder.get_embeddings(text)
w1 = embeddings['Invoice Number']
w2 = embeddings['123456789']
r_invoice_number_is = w1 - w2
```

Here, “r_invoice_number_is” is the seed relationship vector. Since, it’s natural for every invoice document to have an invoice number in a similar context, so we believe that if we apply the

"r_invoice_number_is" vector in all the remaining invoice documents we will get their corresponding invoice numbers –

w_j = w_i - r_invoice_number_is

where w_j = all invoice numbers by value and w_i = all invoice number n-gram strings.

Algorithm

Assume there are n documents of type T in total.

Step 1: Create a list of seed relationship vectors applicable for document of type T, say $T\_R\_LIST_{seed}$.

Step 2: Create an empty knowledge graph say T_KG.

Step 3: Create a reinforcement learning model (actor-critic) say T_RL model.

Step 4: For every relationship vector r_i in $T\_R\_LIST_{seed}$.

Step 5: Create a set of tuples S_i = [(w_{pk}, w_{qk}) . . .], 0 < k < n (n = total number of documents) such that word vector w_{pk} of document k is related to a corresponding w_{qk} of same document k in a relationship vector r_{ik} which is **very similar** to the relationship vector r_i i.e. $w_{pk} - w_{qk} = r_{ik} \sim r_i$. The similarity score calculation algorithm between r_i and r_{ik} is defined in detail later.

Step 6: Create edges $w_{pk} - r_i \rightarrow w_{qk}$ for each (w_{pk}, w_{qk}) pair in S_i and add them to T_KG.

Step 7: Generate a new relationship vector r_{i_new} from the seed relationship vector r_i via the actor NN (initialized with random weights) of the RL model.

Step 8: Create a set of tuples S_{i_new} this time for the new relationship vector r_{i_new} by finding out pairs of (w_{pk}, w_{qk}) in every document k, (0 < k < n) such that $w_{pk} - w_{qk} = r_{i\_new\_k} \sim r_{i\_new}$.

Step 9: If the size of S_{i_new} is above a threshold value $V_{threshold}$ then S_{i_new} can be considered as a new valid latent relationship vector present inside documents of type T, else S_{i_new} is a bogus relationship vector.

Step 10: If S_{i_new} is valid relationship vector then feedback a reward scalar to the T_RL model's actor and critic NNs and create edges $w_{pk} - r_{i\_new} \rightarrow w_{qk}$ for each (w_{pk}, w_{qk}) pair in S_{i_new} and add them to T_KG, else feedback a punishment scalar to the T_RL model's actor and critic NNs.

*(**Important**: The feedback back-propagation into the actor and critic NNs is done via stochastic gradient descent, because it is not necessary that the S_{i_new} produced by the actor NN is at a distance from the S_i which can be expressed as a regular function, linear/non-linear. The S_{i_new} can very possibly be at random distances from the S_i.)*

Step 11: Goto Step 7, until no new relationship vectors are obtained.

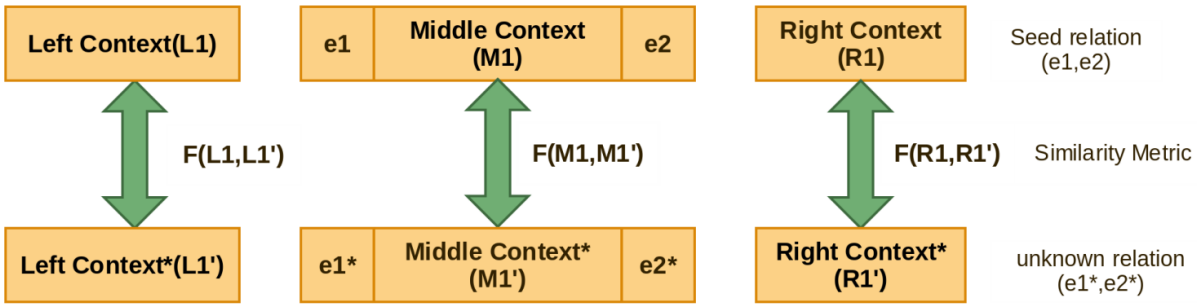
Step 12: Goto Step 4

Step 13: The final T_KG is a well-directed graph of all possible information hidden inside documents of type T.

Relationship Vector Similarity Algorithm

As mentioned above we need a very highly efficient relationship vector similarity algorithm to cluster or group word vectors together belonging to either a seed relationship vector or a latent relationship vector. We use the following algorithm to calculate the similarity –

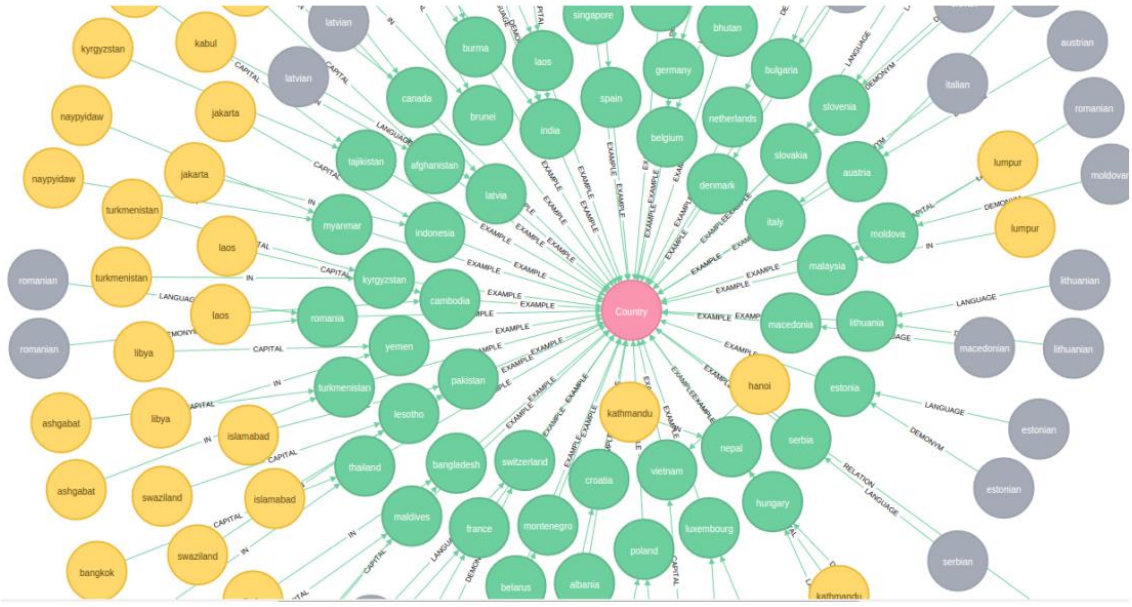
We find the similarity using a context-based method. The method tries to find the similarity between two relationships based on the context in which the linked entities occur. We try to find the left, middle and right context for the relationships and try to find the similarities between these contexts.



The context is represented by averaging out the word vectors that constitute the context and we calculate the cosine distance between the context to get an estimate of the similarity.

Knowledge Graph Usage

As you can get an idea from the algorithm that the KG is incrementally created. With every new latent relationship vector discovered a new layer of vertices gets added to the KG. So, by the time the algorithm gets completed we have a huge KG specifically created from the n documents of type T. Therefore, every document type has a unique KG for itself. The RL model also gets trained simultaneously to effectively discover valid latent relationship vectors for every document type. Therefore, as any new document comes into the system the RL model uses it to discover all existing relationships and any new hidden relation if any.



Knowledge Graph Query Language

As shown above, a sample KG, we can very easily develop a query language on top of this directed graph. Suppose we want to find the price of all items for an invoice number –

SELECT price OF items FOR invoice_number = 123456789

The KG query processing unit starts from the “invoice_number_is” relationship vector to “item_belongs_to” relationship vector to the “has_price” relationship vector for getting the prices of all items that belongs to the given invoice_number.

Risks and Limitations

There are two major risks that can be identified –

1. Hypothesis 1 is a well proven concept in the Word2Vec paper. The Hypothesis 2, although looks rational is unproven and new. So, the Hypothesis 2 is a major risk.
2. Since we are using Reinforcement Learning, there is a high risk that with a slow learning rate the model will converge very slowly and with a high learning rate the model will underfit. So, the learning rate selection is a also another major risk.