

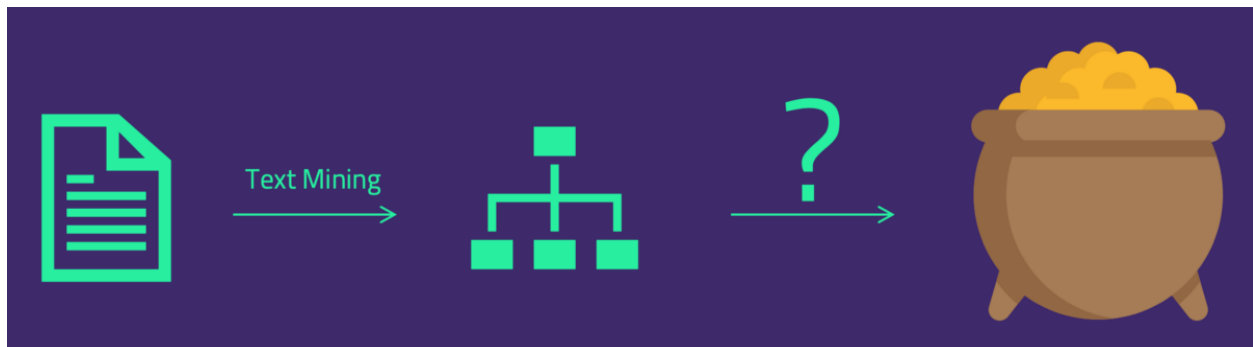
Text Mined Knowledge Graphs

Text is the medium used to store the tremendous wealth of scientific knowledge regarding the world we live in. However, with its ever-increasing magnitude and throughput; analysing this unstructured data has become an impossibly tedious task. This has led to the rise of Text Mining and Natural Language Processing (NLP) techniques and tools as the go-to for examining and processing large amounts of natural text data.

Text-Mining is the automatic extraction of structured semantic information from unstructured machine-readable text. The identification and further analysis of these explicit concepts and relationships help in discovering multiple insights contained in text in a scalable and efficient way.

Some of the various text-mining/NLP techniques include; structure extraction, tokenisation, acronym normalisation, lemmatisation, de-compounding, and identifying language, sentences, entities, relations, phrases and paragraphs.

But once we have this so called “structure semantic information”, what do we do then? Do these text mining techniques simply produce the insights we are trying to uncover?



Well, it is not that simple. Even after extracting some information from text, there is a long way to go to convert that into, first, knowledge, and then, into a valuable insight. That insight may be a new discovery or confirming and verifying a previous hypothesis in the form of creating new links in existing knowledge we have about our domain. Let us look at why going beyond text-mining is not an easy task.

What are the Challenges of Going Beyond Text Mining?

While working with a number of NLP tools, I found several challenges between the output of a text-mining/NLP tool and the insights I was looking for. These can be summarised as follows:

- 1. Difficult to ingest and integrate complex networks of text mined outputs**

Text mining a single instance of text is easy. We could also just read the text and extract the knowledge contained in it ourselves. However, what do we do when we have hundreds, thousands or millions of independent text instances in our corpus. In this case it becomes extremely hard to uncover and understand the relationships between the knowledge extracted

(text mined outputs) from each text against each other. To be able to do such analyses, we need to integrate all the outputs in one place — which is not as easy as it sounds.

2. Difficult to contextualise knowledge extracted from text with existing knowledge

Second, not only do we want to analyse knowledge extracted from text, but we want to go beyond that, to see how the information extracted relates to all the other data we have. These data would have its own format or structure; making it impossible to compare it with our original NLP output. This leads to the difficulty of contextualising relationships between various groups of disparate and heterogeneous data.

3. Difficult to investigate insights in a scalable and efficient way

Finally, due to the magnitude of text that can be extracted, it becomes extremely tedious to generate or investigate insights in a scalable way. Of course, valuable insights can be discovered manually for single instances of text, but such an approach is impossible to scale across millions of text instances. Moreover, in most cases, doing this manually is simply practically impossible. What do we do then?

How do we Address These Challenges?

With this in mind, we can think of potential solutions that address these challenges. Based on my research, I suggest this methodology:

1. Integrate and ingest complex networks of text mined output into one collection database

To solve the first challenge, we need a method to easily accumulate text mined output into one collection — in other words, a text mined [knowledge graph](#).

2. Impose an explicit structure to normalise all data

To enable the intelligent analysis and integration of data, while also maintaining data integrity, we need to impose an explicit structure on all the data we want to analyse. Not only will this help to contextualise the concepts themselves, but also the relationships between them. This translates into having a higher-level data model to encompass the various types of data and consolidate their presence in the knowledge graph. This will then also allow us to validate the data at the time of ingestion. The data model will act as an umbrella over all data types allowing us to contextualise all relationships within and between them.

3. Discovering new insights using automated reasoning

In order to extract or infer as much information as possible from our knowledge graph, we need some sort of [automated reasoning](#) tool to propagate our domain expertise throughout the entirety of the data. This will enable us to ask questions from our knowledge graph and get the right answers with their explanations — where other traditional methods would fail.

Having identified solutions to the previously listed challenges, I wondered whether there was any one technology out there, that encompassed all three points?

Well, to our luck, [Grakn](#) solves all of these.

If you're unfamiliar with it, Grakn is an intelligent database in the form of a knowledge graph to organise complex networks of data. It contains a [knowledge representation system](#) based on [hyper-graphs](#); enabling the modelling of any type of complex networks. This knowledge representation system is then interpreted by an automated reasoning engine, which performs reasoning in real-time. This software gets exposed to the user in the form of a flexible and easily understood query language — [Graql](#).

Building a Text Mined Knowledge Graph?

But how do we actually go about building a text mined knowledge graph using Grakn?

Step 1: Identify Text

To get started, we first need to know the text we are mining and want to store. In this article, I want to focus specifically on biomedical data. Therefore, the types of unstructured text data that I looked at included:

1. [Medical Literature](#)
2. [Diagnostic Test Reports](#)
3. [Electronic Health Records](#)
4. [Patient Medical History](#)
5. [Clinical Reports](#)

Out of the above listed text corpuses. I chose to look specifically into medical literature. Specifically, [PubMed](#) article abstracts.

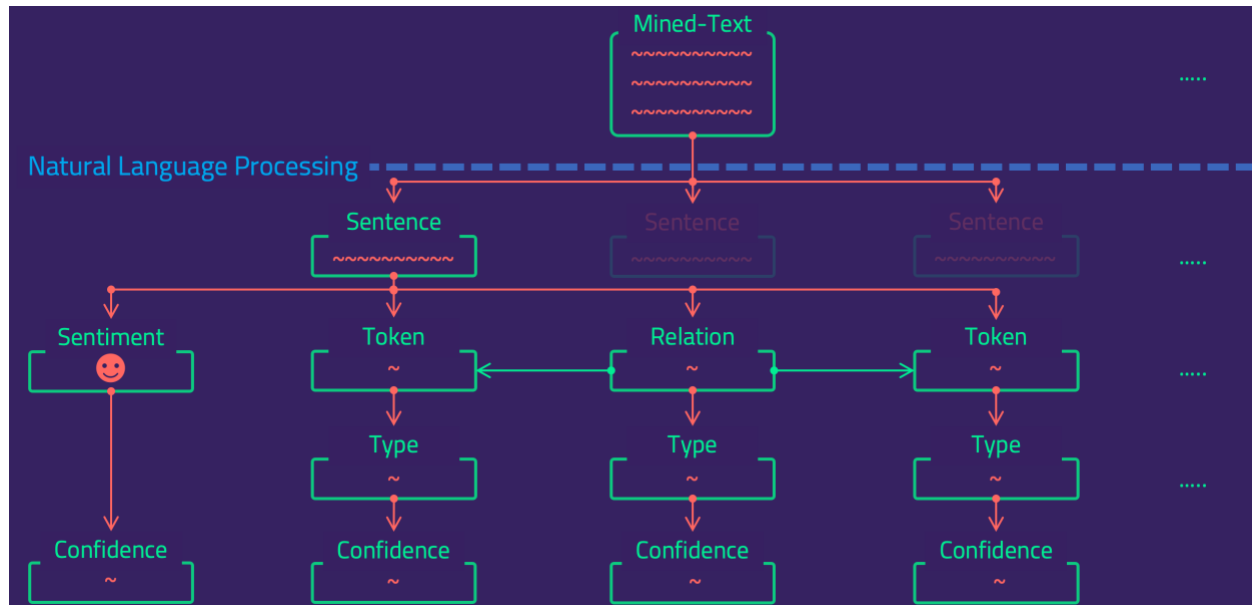
Step 2: Identify Text mining/NLP tool

While looking into this space, I found it easy to integrate [Stanford's CoreNLP API](#) to allow me to train and mine text. The script to mine text can be found on our GitHub repo [here](#). Other tools that may be used instead may include; [NLTK](#), [TextBlob](#), [gensim](#), [spaCy](#), [IBM Watson NLU](#), [PubTator](#), [LitVar](#), [NegBio](#), [OpenNLP](#), and [BioCreative](#).

Step 3: Model Your Data

Thanks to CoreNLP, we now have raw text mined data, and we can move onto data modelling. To this end, Grakn utilises the [entity-relationship model](#) to group each concept into either an entity, attribute, or relationship. This means that all we have to do is to map each concept to a [schema concept type](#), and recognise the relations between them. Let us look at an example to demonstrate how we would go about doing this.

In order to start modelling our CoreNLP output, we first need to know what it actually looks like. The most basic mining extracts sentences from a body of text. Those sentences have a sentiment, tokens (which make up the sentence), and relations between certain tokens. We also get a confidence measure of each type that the tool identifies. A visual representation of this is shown below:



As we can see, the structure of our output is already graph-like. Traditional methods strive to encapsulate this rich information into tables, however that strips away a whole dimension of information and is counter-productive. It feels more natural to keep this graph-like structure to build an integrated complex knowledge graph of everything we have extracted from the text corpus.

We can easily map the output to [Graql](#):

```
define

# Entities

mined-text sub entity,
  has text,
  plays container;

sentence sub entity,
  has text,
  has sentiment,
  plays contained,
  plays container;

token sub entity,
  has lemma,
  has type,
  plays contained,
  plays object,
  plays subject;

# Relations

containing sub relation,
  relates contained,
  relates container;

mined-relation sub relation,
  has type,
  relates contained,
  relates object,
  relates subject;

# Attributes

text sub attribute, datatype string;
lemma sub attribute, datatype string;
sentiment sub attribute, datatype string;
type sub attribute, datatype string;
```

We start by recognising `mined-text`, `sentence` and `token` to be entities, having attributes such as `text`, `sentiment`, `lemma` and `type`. We can also see the roles they play in the relations we will define next.

We can identify two relations. One is a `containing` relation which is between something that is `contained` and something that is the `container`. We can look at the entities to see that the `mined-text` entity plays the `container` for a `sentence`, a `sentence` plays the `contained` with respect to the `mined-text` and as the `container` with respect to a `token`. Lastly, the `token` only plays the `contained` with respect to the `sentence`.

The second relation is the `mined-relation` which relates the `tokens` as `objects` or `subjects` and also has the `contained` relation to show which `sentence` that `mined-relation` was extracted from. The relation `mined-relation` also has an attribute `type`, to represent the type of relation.

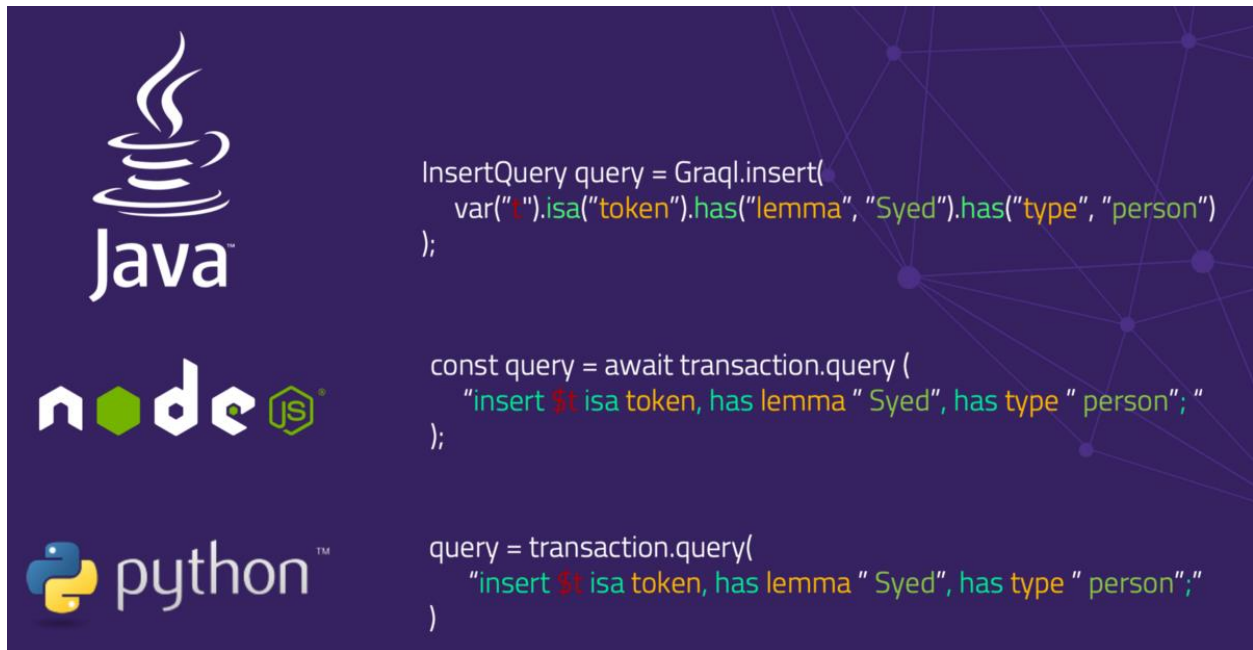
Finally, we also need to specify the datatypes of each attribute we have defined, which can be done as the last four lines of the schema above.

Step 4: Migrate into Grakn

Now that we have the data, and a structure imposed on this data, the next step is to migrate

this into Grakn to convert it into knowledge. Please note there are many different ways to do migration, but here I would like to specifically touch on how we would go about using [Java](#), [NodeJS](#) and [Python](#).

For this, we can easily use any of these languages to insert instances of extracted information compliant with the schema we modelled. The image below depicts how to insert a single instance of a `token` with a `lemma` and `type` into Grakn using any of these three languages:



To learn more about [migrating data](#) into Grakn, make sure to read this article: [Modelling & Migrating Big Biological Data with Grakn](#).

Step 5: Discover and Interpret New Insights

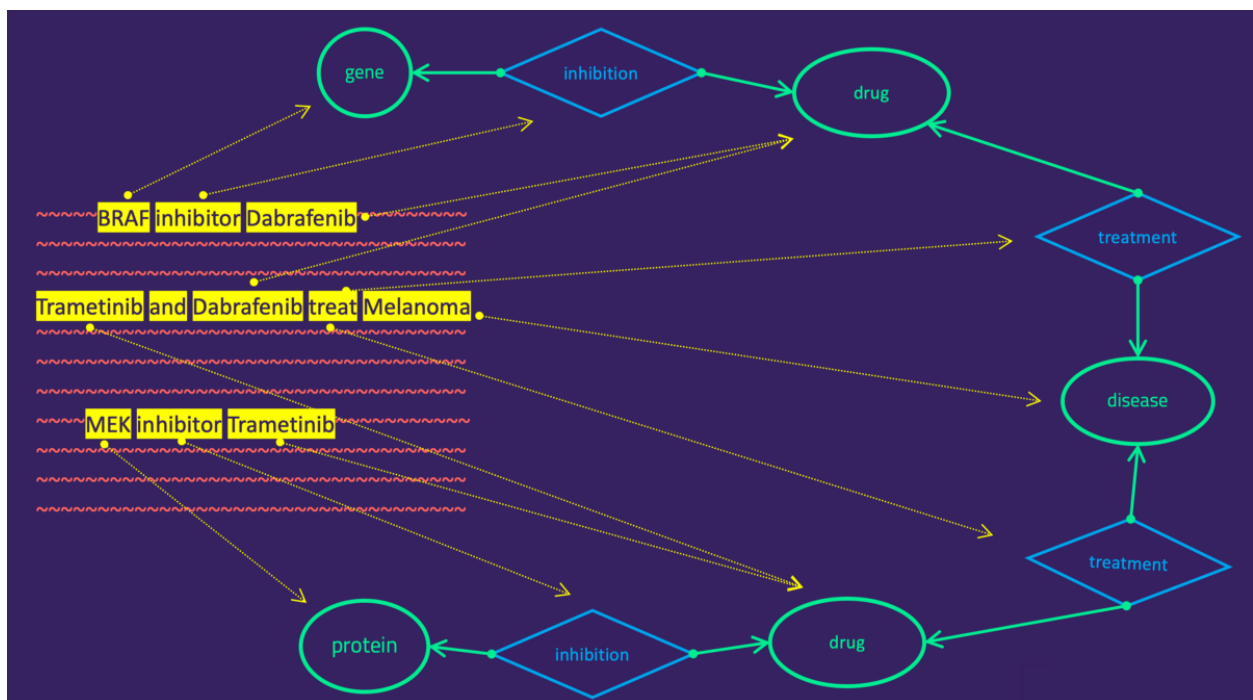
After migration, we can start to discover new insights. Discovering insights refers to finding new data that may be valuable to what we are trying to accomplish. In order to do that, we need to first look or ask for something. In other words, we start with a question. These questions can range from asking within the text, or even more complex ones regarding other data which is augmented by the text mined output.

Let us look at some examples, and see how our text mined knowledge graph may provide answers to them:

Question 1: What knowledge is extracted from a PubMed article?

```
match
  $p isa pubmed-article, has pmid "123";
  $r ($p, $k) isa knowledge-extraction;
get;
```

Answer 1:



The answer we get back are the various concepts that were mined from the abstract of the PubMed article we are interested in. The figure above shows an example of this where the entities extracted were:

1. Gene — BRAF
2. Drug — Trametinib
3. Drug — Dabrafenib

4. Disease — Melanoma
5. Protein — MEK

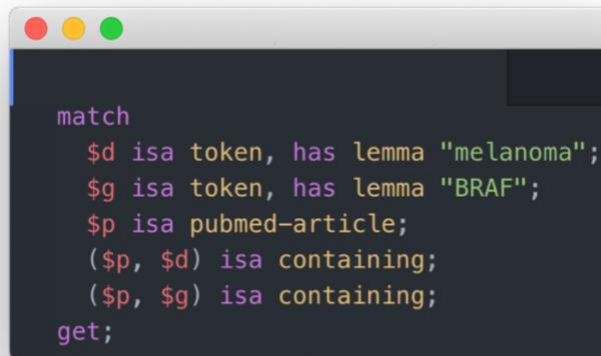
We also can see some mined-relations that were extracted:

6. Inhibition — between BRAF and Dabtrafenib
7. Inhibition — between MEK and Trametinib
8. Treatment — between Dabtrafenib and Melanoma
9. Treatment — between Trametinib and Melanoma

These entities and relations were extracted due to the training performed on our CoreNLP tool. More information regarding this can be found [here](#).

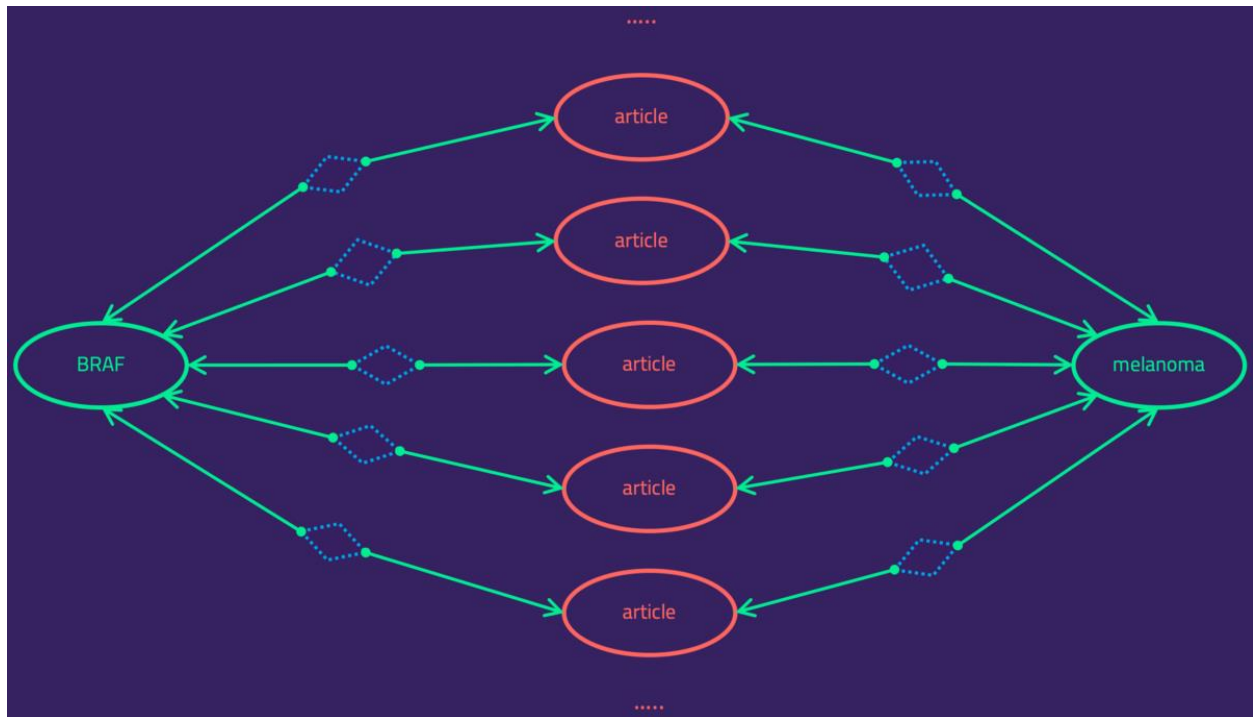
Let us now try asking a harder question:

Question 2: Which PubMed articles mention the disease Melanoma and the gene BRAF



```
match
  $d isa token, has lemma "melanoma";
  $g isa token, has lemma "BRAF";
  $p isa pubmed-article;
  ($p, $d) isa containing;
  ($p, $g) isa containing;
get;
```

Answer 2:



The answer we receive from Grakn is a list of all PubMed articles that match the condition provided. A great application of such a query can be proposed in the Precision Medicine domain where we want to link individual patients to medical literature relevant to their personal biological medical case. If you're interested in precision medicine and knowledge graphs, checkout this [article](#).

These queries are useful, however now it begs to question—how can it leverage our text mined knowledge graph to augment our existing knowledge? How can we use what we mined to expand our insights and apply them to other domains or fields in life sciences? The next question shows a demonstration of this:

Question 3: Which drug is related to the disease Melanoma?

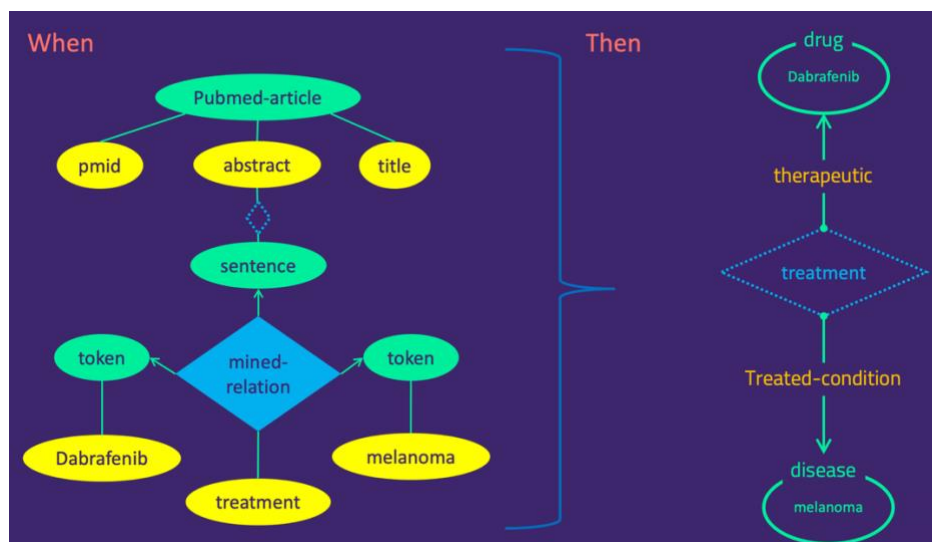
```
match
  $di isa disease, has name "melanoma";
  $dr isa drug;
  $r ($di, $dr);
get; limit 1;
```

Answer 3:



Even though Grakn gives us a correct answer to our question, this data was actually never ingested into Grakn — no connections exist between diseases and drugs. So, how did we get this relevant answer?

In short — Grakn's automated reasoner created this answer for us through [automated reasoning](#). As this type of reasoning is fully explainable, we can interpret any inferred concept to understand how it was inferred/created. Below you can see how this explanation looks like. In the next section, I will dive deeper into how we created the logic and rules that allowed Grakn to infer these relationships.



The relation we saw above, was the product of first order logic. This can be modelled as a when and then scenario.

The when in this case refers to a sub-graph of our text mined knowledge graph — i.e., whenever any sub-graph like the one on the left is found, the relation on the right is created.

This logic propagates throughout the knowledge graph creating new valuable connections, which is done by written rules in Graql. The rule that I used for the inference above, is as follows:

```
treatment-rule sub rule,
when {
  $p isa pubmed-article, has abstract $a;
  ($a, $s) isa containing;
  ($s, $token1, $token2) isa mined-relation, has relation-type "treatment";
  $token1 has lemma $l1;
  $token2 has lemma $l2;
  $drug isa drug, has name $drug-name;
  $drug-name == $l1;
  $disease isa disease, has name $disease-name;
  $disease-name == $l2;
},
then {
  (therapeutic: $drug, treated-condition: $disease) isa treatment;
};
```

The above rule states that when:

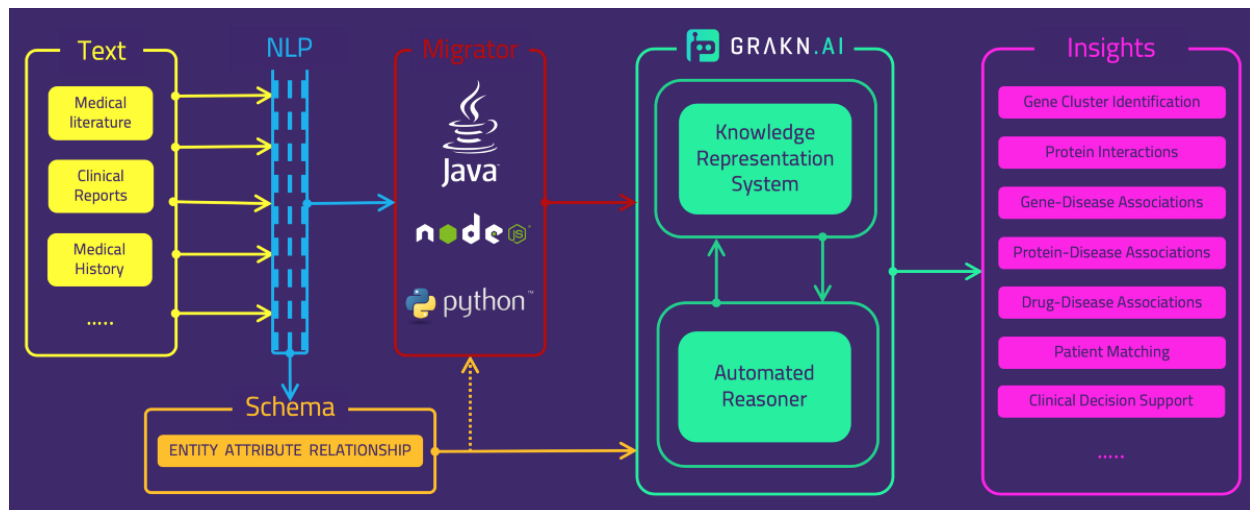
1. There exists a PubMed article \$p with an abstract \$a
2. There exists a sentence \$s contained in abstract \$a
3. There exists a mined-relation extracted from sentence \$s which has the type “treatment”
4. The tokens taking part in that mined-relation have lemmas
5. Those lemmas have the same value as a drug and a disease

If this is true, then create a treatment relation between the drug and disease. We can also see from the rule above that it is agnostic to the disease or drug; hence creating treatment relations between each drug and disease that match the given conditions.

It should be noted that the rule above is a demonstration of how automated reasoning can be used to create a high-level abstraction over complex insights which scale through the data. It by no means compares to the full implementation of how an end-user application would look like.

How do All the Pieces Fit Together in One Architecture

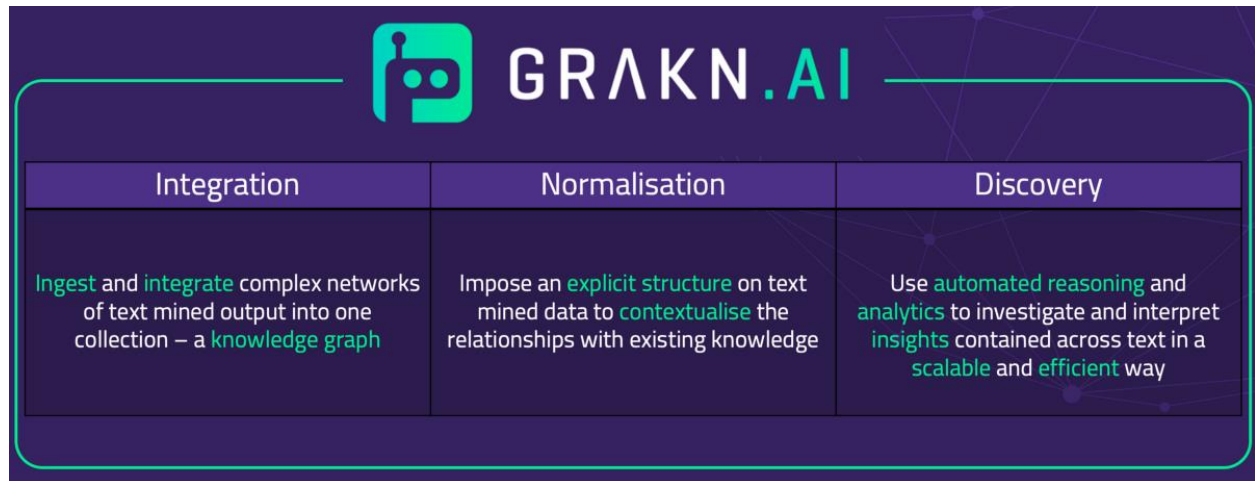
Now, let us take a step back, and look at how all of the components of building a text mined knowledge graph piece together.



We start with the text which can come from multiple sources and in various formats. An NLP tool is used to mine the text and produce some sort of output with a structure. That structure is used to create a [schema](#) (high level data model) to enforce a structure on the raw NLP output. Once that is done, we use one of [Grakn's clients](#) to migrate the instances of NLP output into Grakn making sure every insertion adheres to the schema. Grakn stores this in its knowledge representation system, which can be queried for insights to discover complex concepts or even test out hypotheses. These insights can already be in the graph or even be created at the time of query by the reasoning engine; for example, discovering gene cluster identifications, protein interactions, gene-disease associations, protein-disease associations, drug-disease associations, patient matching, and even for clinical decision support.

Conclusion

So, we know that *Text Mining* is extremely promising in several domains, and in this article, we have specifically looked at the biomedical domain. We understand that there is a lot of work required from the point of extracting information from text to actually discovering something useful. I hope to have shown that Grakn can help to bridge that gap. In summary, Grakn helps to solve the three key challenges in text mining when it comes to going beyond text mining.



For more information, please visit grakn.ai or contact us at enterprise@grakn.ai.