Improving content discovery through combining linked data and data mining techniques

Ross Fenning

December 22, 2015

CONTENTS

CHAPTER
ONE

INTRODUCTION

Media companies produce ever larger numbers of articles, videos, podcasts, games, etc. – commonly collectively known as "content". A successful content-producing website not only has to develop systems to aid producing and publishing that content, but there are also demands to engineer effective mechanisms to aid consumers in finding that content.

Approaches used in industry include providing a text-based search, hierarchical categorisation (and thus navigation thereof) and even more tailored recommended content based on past behaviour or content enjoyed by friends (or sometimes simply other consumers who share your preferences).

1.1 Problems

There are several technical and conceptual problems with building effective content discovery mechanisms, including:

- Large organisations can have content across multiple content management systems, in differing formats and data models. Organisations face a large-scale enterprise integration problem simply trying to gain a holistic view of all their content.
- Many content items are in fairly opaque formats, e.g. video content may be stored as audio-visual binary data with minimal metadata to display on a containing web page. Video content producers may not be motivated to provide data attributes that might ultimately be most useful in determining if a user will enjoy the video.
- Content is being published continuously, which means any search or discovery system needs to keep up with content as it is published and process it into the appropriate data structures. Any machine learning previously performed on the data set may need to be re-run.

1.2 Hypothesis

The following hypotheses are proposed for gaining new insights about an organisation's diverse corpus of content:

- Research and software tools around the concept of *Linked Data* can aid us in rapidly acquiring a broad view (perhaps at the expense of depth) of an organisation's content whilst also providing a platform for simple enrichment of that content's metadata.
- We can establish at least a naïve mapping of an RDF graph representing a content item to an attribute set suitable for data mining. With such a mapping, we can explore applying machine learning – particularly unsupervised learning – across an organisation's whole content corpus.
- Linked Data and Semantic Web *ontologies* and models available can provide data enrichment beyond attributes and keywords explicitly avaiable within content data or metadata.
- We can adapt established machine learning approaches such as clustering for data published continuously in real time.
- Many content-producers currently enrich their web pages with small amounts of semantic metadata to provide better presentation of that content as it is shared on social media. This enables simple collection of a full breadth of content with significantly less effort than direct integration with content management systems.

CHAPTER

TWO

BACKGROUND

This chapter discusses some of the existing research and technologies around machine learning, RDF and combining them. It also covers some of the advantages of using linked data and RDF in an enterprise setting and what tools and approaches are well-defined enough that a corporation could build on top of them rapidly.

Data mining activities such as machine learning rely on structuring data as *feature sets*[2] – a set or vector of properties or attributes that describe a single entity. The process of *feature extraction* generates such feature sets from raw data and is a necessary early phase for many machine learning activities.

The rest of this chapter will show:

1. that extracting feature sets from RDF[1] graphs can be done elegantly and follows naturally from some previous work in this area; and

2. that the RDF graph is a suitable and even desirable data model for content metadata in terms of acquiring, enriching and even transforming that data ahead of feature extraction.

2.1 Data Mining

TODO

2.2 RDF and Feature Extraction

The RDF graph is a powerful model for metadata based on representing knowledge as a set of subject-predicate-object *triples*. The query language, SPARQL, gives us a way to query the RDF graph structure using a declarative pattern and return a set of all variable bindings that satisfy that pattern.

For example, the SPARQL query in Listings 2.1 queries an RDF graph that contains contact information and returns the names and email address of all "Person" entities therein.

[1]http://www.w3.org/TR/PR-rdf-syntax/

Notably, Kiefer, Bernstein and Locher[3] proposed a novel approach called SPARQL-ML – an extension to the SPARQL[8] query language with new keywords to facilitate both generating and applying models. This means that the system capable of parsing and running standard queries must also run machine learning algorithms.

Their work involved developing an extension to the SPARQL query engine for *Apache Jena*[2] that integrates with systems such as *Weka*[3]. A more suitable software application for enterprise use might focus solely on converting RDF graphs into a neutral data structure that can plug into arbitrary data mining algorithms.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
  ?person foaf:mbox ?email.
}
```

Listing 2.1 : Example SPARQL query for people's names and email addresses

If we consider an RDF graph, g, to be expressed as a set of triples:

$$(s, p, o) \in g$$

this query could then be expressed as function $f : G \to (S \times S)$ where G is the set of all possible RDF graphs and S is a set of all possible strings. This allows the result of the SPARQL query to be expressed as a set of all SELECT variable bindings that satisfy the WHERE clause:

$$q(g, n, e) = \exists p.(p, type, Person) \in g \ \wedge (p, name, n) \in g \wedge (p, mbox, e) \in g$$

$$g \in G \models f(g) = \{(n, e) \subseteq (S \times S) \mid q(g, n, e)\}$$

This could be generalised to express a given feature set as vector $(a_1, a_2, ..., a_n)$:

$$g \in G \models (a_1, a_2, ..., a_n) \in f(g)$$

and in the case where all $a_k \in f(g)$ are literal (e.g. string or numeric) values, we can thus consider a given SPARQL query to be specific function capable of feature extraction from any RDF graph into sets of categorical or numeric features.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?topic
WHERE {
  ?article rdf:about ?topic .
}
```

Listing 2.2 : SPARQL query to determine what

[2]https://jena.apache.org/

[3]http://www.cs.waikato.ac.nz/ml/weka/

This might allow a query that extracts a country's population, GDP, etc. provide feature extraction for learning patterns in economics, for example. However, this is limited to features derived from single-valued predicates with literal-valued ranges. It is not clear how to formulate a query that expresses whether or not a content item is about a given topic.

In the RDF model, it would be more appropriate to use a query like that in Listings 2.2 where for a given ?*article* identified by URI, we can get a list of URIs identifying concepts which the article mentions. Such a query might be expressed as function $f' : G \to \mathcal{P}(U)$ where U is set of all URIs such that:

$$g \in G \models f'(g, uri) = \{t \mid (uri, about, t) \in g\}$$

An approach of generating attributes for a given resource was proposed by Paulheim and Fürnkranz[5]. They defined specific SPARQL queries and provided case study evidence for the effectiveness of each strategy.

Their work focused on starting with relational-style data (e.g. from a relational database) and using *Linked Open Data* to identify entities within literal values in those relations and generated attributes from SPARQL queries over those entities.

For a large content-producer, there is a more general problem where many content items do not have a relational representation and the content source is a body of text or even a raw HTML page. However, the feature generation from Paulheim and F urnkranz proves to be a promising strategy given we can acquire an RDF graph model for content items in the first place.

2.3 RDF in the enterprise

TODO

CHAPTER
THREE

SYSTEM DESIGN

In this chapter, a system is inductively derived and concretely design to make use of multiple strategies for:

1. gathering (meta)data about all of an organisations content items;
2. extracting metadata not explicitly modelled in source content management systems;
3. further enriching that metadata with information not explicitly present in the content item itself; and
4. applying machine learning to that content metadata to gain new insights about that content.

Initially, a business context is described to produce a design for a system that could be a applied within a media or content-producing organisation. This context will guide all design decisions.

3.1 Context

3.2 Use Cases

3.3 Data Pipeline

A core subsystem in the overall system is a conceptual data pipeline whose input is a URI or IRI identifying a content item published on an organisation's website and the output is feature sets ready for applying machine learning.

In this section, a theoretical pipeline is inductively defined in steps such that an application of this pipeline would choose to implement some subset of all potential pipeline stages as appropriate for the relevant problem domain.

In Chapter 4, aa system is engineered that implements as many of these pipeline stages as possible such that a running instance of the application can configure which components to use and which not to use. Then in Chapter 5, an evalution of the system is given while it is running each component in isolation to demonstrate which of the theoretically-defined processes in this chapter appears to be most effective in generating feature sets specifically for clustering web content.

3.3.1 Definitions

This system requires some initial definition of some data structures in use:

IRI

The input to the system is a character string conformant to the IRI syntax defined in RFC 3987[1]. This allows more generality offered by URIs[2] but is trivially made compatible with systems that use URIs through the conversion algorithm defined in section 3.2 of RFC 3987. Note that the public URL by which the public can read or otherwise consume the content is a valid identifier, but we are not restricted to that.

Feature Set

The final output of this data pipeline is a data structure analogous to a relation or tuple per IRI fed into the system. Every IRI should have a literal value against all possible columns or fields. For binary fields, (e.g. the presence of absence of a concept tag), a more pragmatic structure might be a list of tags positively associated with the IRI rather than explicitly assigning *false* to all tags to which the content item does not pertain. This is analogous to a spare matrix when dealing with a large number of dimensions.

Named RDF Graph

The structure used throughout most of the data pipeline is that of an RDF graph. This is used for all the benefits outlined in Section 2.3 such as ease of transformation and combining of data sets. Named graphs are used such that all data acquired are keyed back to the IRI of the content item being processed. This also allows all graphs to be combined in a *triplestore* if needed to allow SPARQL queries across the combined data for all content items. This can be modelled as a data structure in many programming languages, but where a serialisation is used (e.g. examples shown here or to send the data between components), the JSON-LD[9] syntax will be used.

3.3.2 Identity Graph

```json
{
  "@id": "http://example.com/entity/1",
  "@graph": []
}
```

Listing 3.1 : Identity graph for a content item in JSON-LD syntax

With the knowledge only of a content item's IRI, we are arguably only able to produce an empty named RDF graph. Such a graph for an example IRI `http://example.com/entity/` is illustrated in JSON-LD syntax in Listings 3.1.

[1]http://tools.ietf.org/html/rfc3987
[2]http://tools.ietf.org/html/rfc3986

The most naïve feature set we can generate from such an RDF graph is clearly a singleton relation `("http://example.com/entity/")` where a single *IRI* field has the value `"http://example.com/entity/"`. It is also clear that a set of one-dimension feature vectors with unique values in each is not suitable for any form of machine learning activity. This does, however, illustrate a baseline for a working software application that is – at least in the syntactic sense – transforming IRI inputs to feature sets outputs. Such a *null* feature generator is depicted in Figure 3.1.

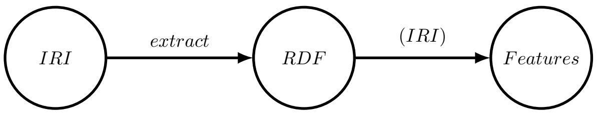

Figure 3.1: Null feature generator

Note that Figure 3.1 shows all three data structures involved despite having no functional use. We can also see top-level definitions of the process where we first *extract* semantic information in RDF from a content item indentified by IRI and then *generate* features therefrom. More useful models can now be inductively defined by adding atomic subcomponents that may each add value to the overall transformation.

There are three clear axes along which we can improve this pipeline: *extract* more RDF data knowing only an item's IRI, expand or *enrich* an existing RDF graph and then improve how we *generate* features for data mining. In the first instance, we can consider the former and add a single pipeline stage for expanding the RDF graph.

3.3.3 RDF Extraction

Tim Berners-Lee outlined four rules[1] for Linked Data, rule number three of which states "When someone looks up a URI, provide useful information, using the standards". If we assume that many pages have embedded some semantic web or RDF data, then a simple extraction strategy would be to deference the content item's IRI via an HTTP GET and pass the content to a parser capable of extracting RDFa, microformats, etc.

Many tools such as the RDFLib[3] provide functionality for taking a URL and returning an RDF graph of all data found when fetching the resource it represents, so this is arguably an ideal first choice in attempting to learn something about a content item from its IRI.

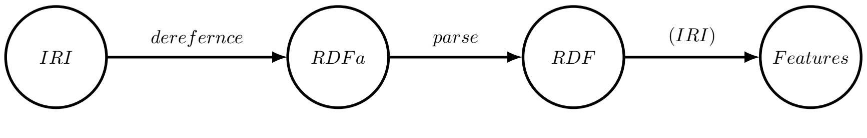

Figure 3.2: Semantic web data extraction

[3]https://github.com/RDFLib/rdflib

Figure 3.2 depicts the pipeline with a simple dereference step added. Note that the feature set generated is still the singleton relation with only the IRI value. Thus the next step should be to add a step that improves the feature set generation.

3.3.4 Feature Set Generation

Paulheim and Fürnkranz[5] described a number of SPARQL queries for generating feature sets from RDF data, which could inspire a simple query such as that shown in Listings 3.2. This query generates a boolean `true` value for any properties that match and implies `false` for those that do not.

```
SELECT ?p ?v
WHERE { ?iri ?p ?v . }
```

Listing 3.2 : Generates field `content_?p_?v` with value `true`

As also noted by Paulheim and Fürnkranz, this overlooks the *open world assumption*[7]. However, application of clustering algorithms on binary data can employ asymmetric distance metrics such as Jaccard similarity coeffcient[10], which notably avoids deriving similarity from negative values. That is, two content items lacking a particular property will contribute no information about their (dis)similarity. Thus we safely avoid inadvertently grouping together one item that genuinely lacks the property with another that indeed has the property, but we lack the positive assertion thereof in the data extracted.

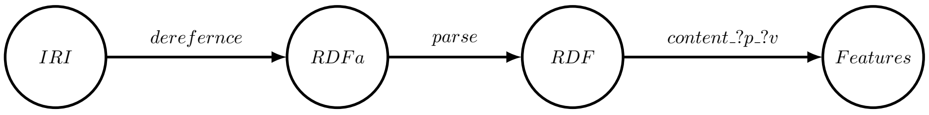

Figure 3.3: Semantic web content extraction with basic SPARQL feature generation

The basic pipeline in Figure 3.2 can thus be augmented with this basic feature extraction to produce the pipeline depicated in Figure 3.3.

3.3.5 RDF Enrichment

The third and final direction in which this data pipeline can be improved is in terms of data enrichment. A simple strategy here is to repeat the deferencing used in Section 3.3.3, but for each IRI found as the object of a triple in which the initial IRI is the subject. Formally:

$$g \in G \models \exists p, o.(IRI, p, o) \in g \rightarrow g' = deref(o)$$

That RDF graphs can be modelled as mathematical sets as in Section 2.2 means we can express a graph enriched this way as a *union* of the initial graph with each graph returned from all dereferencing:

$$g \in G \models g' = g \cup \bigcup \{deref(o) \mid (IRI, p, o) \in g\}$$

Figure **??** shows the data pipeline with this additional enrichment stage. Note that now we have potential for some stages being executed in parallel.

So far in this section, we have inductively built up a data pipeline from a "null" base working with only identity graphs to a simple pipeline capable of *extracting* an RDF graph, *enriching* it and then *generating* features from it.

What needs to be proven through experimentation is *which* of these provides the information required for effective data mining. As part of this experimentation, we can now look at further techniques and approaches to try.

3.3.6 Improving Extraction

In order to gain a larger set of RDF data at the start of the pipeline, we can derive further ways to get information about a content item given only its IRI as input.

Rizzo and Troncy[6] defined a framework called NERD capable of combining multiple entity extraction systems to provide a unified way of identifying – and disambiguating – named entities within a given body of text. With such a system, we can create a second, parallel RDF extraction strategy that creates a graph of triples in the form:

$$(IRI, \textit{rdf:about}, \textit{Entity})$$

where *Entity* is an IRI representing a concept or entity believed to be found in the content's textual content. A data pipeline complementing the RDFa-based extraction is depicted in Figure 3.5. Note the ability to apply a simple set union to the result of each extraction as with the enrichment.

Another strategy can be to infer a relationship between two content items where one contents an HTML link to another. It is not always possible to derive precise semantics of such a link (unless the publisher has kindly provided a `rel` attribute), but a weak relationship such as:

$$(IRI_1, ex : related, IRI_2)$$

might prove – through experiementation – to be useful enough for data mining insights.

The fourth and final extraction strategy explored is acquiring metadata from bespoke Content Management Systems and other internal APIs. This is generally the only option used in enterprises settings, as discussed in Section 2.3. This is likely to be the richest source of information where an enterprise has typically preferred bespoke integrations against non-hypermedia interfaces, so experimentation should help quantity or qualify the value such a direct integration adds (perhaps to consider it in combination with the cost of bepsoke, repeated integration projects).

The assertion explored here is that such bespoke integrations can *complement* cheaper work such as extracting RDFa with pre-built tools (and thus be developed one-by-one after the initial release of an application such as this data pipeline). Note that repeated custom integration projects means that each data source requires a different application be developed (as opposed to the reuse of a single RDFa parser or HTML link scraper). This also means we are not necessarily comparing like-for-like if we introduce only one at a time. It also makes it difficult to evaluate data mining of a diverse content corpus if an integration against an API provides additional metadata for only, say, 10% of that corpus.

These challenges aside, it is clear that bespoke integrations have a clear place in this data pipeline being applied in a real enterprise setting. Now that we have a complete set of theoretical stages for *extraction*, the remaining improves lie now in the *enrichment* and *feature generation* stages.

3.3.7 Improving Enrichment

In addition to enriching through dereferencing linked entities, it is proposed to explore the following options:

- inferring relationships to hypernyms as defined by Wordnet[4];
- inferring facts based using rules derirved from expert domain knowledge;
- using RDFS and OWL to generate new triples with well-established Ontology rules.

In the first approach, we can consider a relationship rule such as:

$$(IRI, ex:related, ex:Dog)$$

and *infer* the fact:

$$(IRI, ex:related, ex:Animal)$$

and produce an enriched graph containing all additional facts inferred in this way.

When dealing with proper nouns and named entities, inferring facts based on domain knowledge may be more appropriate. For instance, the rule in n3 syntax:

```
{
  ?article ex:takesPlaceIn ?city .
  ?city a ex:City .
  ?city ex:capitalOf ?country .
} -> { ?iri ex:takesPlaceIn ?country }
```

might be useful ot help cluster articles that take place in the same country – even if the countries are not always explicitly mentioned therein. Such an inference requires knowledge about cities and countries to write and domain experts for different types of content might be able to offer more nuanced rules.

An example for BBC content might be to infer that all articles written under the *Newsround* brand is suitable for children or that programmes that have broadcast times during the day are also suitable for children.

The third and final proposed improvement makes use of standard tools to find *closures* using, e.g. RDFS, ontology rules. With this approach, we can infer that entities that have a given type or class also have their superclasses and supertypes. This gives us similar inference to hypernyms, but with knowledge present in well-established ontologies.

An obvious example might where two content items have been identified as related to the same concept – so they would be candidates for clustering together

– but when RDF data are extracted, it is found that two different URIs have been used for each:

$$(IRI_1, \textless http://dbpedia.org/property/related\textgreater, ex1 : entity)(IRI_2, \textless http://dbpedia.org/property/related\textgreater,$$

In the RDF graph for IRI_2, say, we might find the source had provided an `owl:sameAs` assertion such as:

$$(ex2 : anotherEntity, owl : sameAs, ex1 : entity)$$

This is possible in the case where the second item's data source uses its own set of identifiers for entities, but has chosen to provide equivalences to a more standard set of identifiers (e.g. DBpedia). With this information, our data pipeline can infer:

$$(IRI_1, \textless http://dbpedia.org/property/related\textgreater, ex1 : entity)(IRI_2, \textless http://dbpedia.org/property/related\textgreater,$$

and the feature generation stage might provide the common features `dbprop_related_ex1_entity=true` for both content items.

3.3.8 Improving Feature Generation

The feature generation outlined so far relies solely on boolean values indicating whether or not a given content item is related by some property to some entity. This recreates the concept of *tagging* where a given object is either associated or not associated with a series of *tags*.

One of the advantages of the RDF graph model is that we are not constrained necessarily to properties and attributes directly applicable to the entity. We could imagine adding a level of indirection to the query in Listings **??** to create the query in Listings **??**.

```
SELECT ?p1 ?p2 ?v
WHERE {
  ?iri ?p1 ?o .
  ?o ?p2 ?v .
}
```

Listing 3.3 : Generates field `content_?p1_?p2_?v` with value `true`

With the this query, features of the form `content_?p1_?p2_?v` can be generated. An example of this might be where a television programme content item has information about the actors that appeared therein, e.g. `ex:hasActor`, and furthermore we have information about those actors such as where they were born, e.g. `ex:bornIn`. With the path created by following both of these predicates, it is possible to create features for a television programme such as `content_ex:hasActor_ex:bornIn_Edinburgh` and we can potentially find similarity between programmes where the actors were born in the same city.

Perhaps a third step in the predicate path followed can give us even more useful features. The example above could be expanded to `content_ex:hasActor_ex:bornIn_ex:cityIn_Scotlan` to allow the more general ability to cluster programmes with Scottish actors, for instance.

Appropriate experimentation should show whether more value is gained by adding these additional levels of indirection.

3.3.9 Maximal Data Pipeline

In this section, a data pipeline was inductively built up from a base, identify pipeline with suggestions for potential improvements in different stages. An application of all the ideas discussed so far might look like that depicated in Figure3.6.

3.4 Technical Architecture

$$\begin{tikzcd}
 & & & RDF_1 \arrow[dr, "\cup"] & & \\
IRI \arrow[r, "dereference"] & RDFa \arrow[r, "parse"] & RDF \arrow[ur, "dereference/parse"] \arrow[r, "dereference/parse"] \arrow[rr, bend right, "\cup"'] & RDF_2 \arrow[r, "\cup"] & RDF' \arrow[r, "content\_?p\_?v"] & Features
\end{tikzcd}$$

Figure 3.4: Semantic web content miner with addtional dereferencing of linked entities

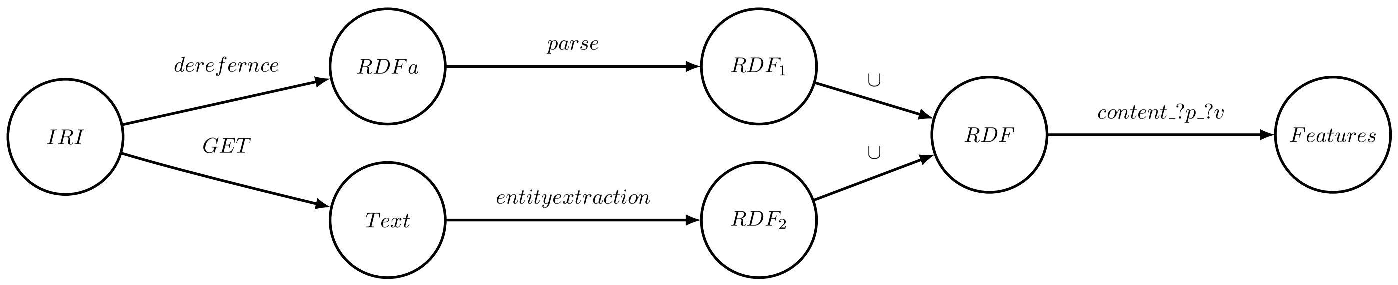

Figure 3.5: Named entity extraction in addition to semantic web extraction

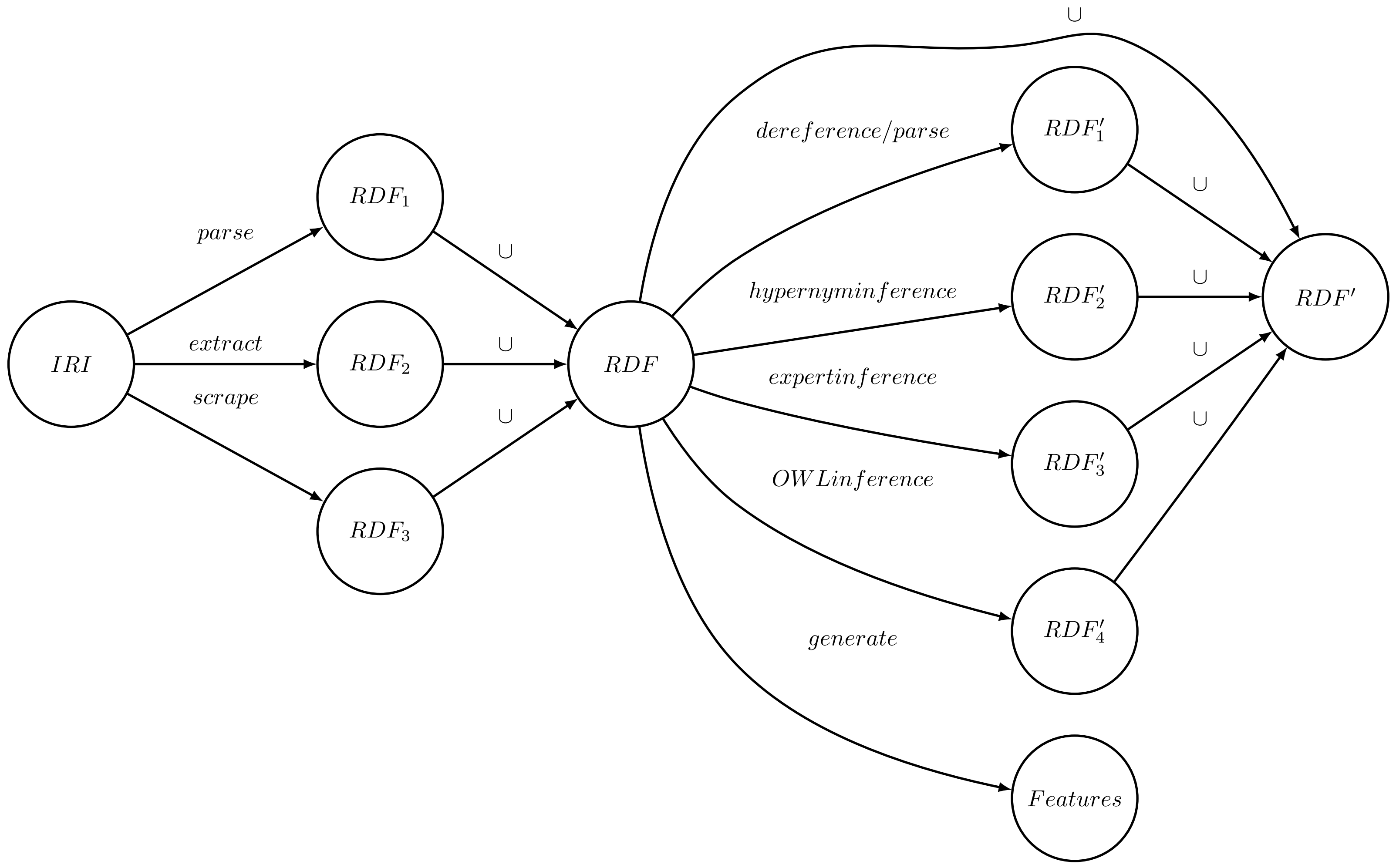

Figure 3.6: Maximal Data Pipeline

CHAPTER

FOUR

IMPLEMENTATION

CHAPTER

FIVE

EVALUATION

CHAPTER

SIX

CONCLUSION

BIBLIOGRAPHY

[1] Tim Berners-Lee. Linked data-design issues (2006). *URL http://www. w3. org/DesignIssues/LinkedData. html*, 2011.

[2] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[3] Christoph Kiefer, Abraham Bernstein, and André Locher. *Adding data mining support to SPARQL via statistical relational learning methods*. Springer, 2008.

[4] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[5] Heiko Paulheim and Johannes Fümkranz. Unsupervised generation of data mining features from linked open data. In *Proceedings of the 2nd international conference on web intelligence, mining and semantics*, page 31. ACM, 2012.

[6] Giuseppe Rizzo and Raphaël Troncy. Nerd: a framework for unifying named entity recognition and disambiguation extraction tools. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 73–76. Association for Computational Linguistics, 2012.

[7] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach, 2010.*

[8] Toby Segaran, Colin Evans, and Jamie Taylor. *Programming the semantic web*. " O'Reilly Media, Inc.", 2009.

[9] Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, and Niklas Lindström. Json-ld 1.0. *W3C Recommendation (January 16, 2014)*, 2014.

[10] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.