2. Background

This chapter provides an overview of the main concepts related to this dissertation. Section 2.1 introduces the main concepts, elements and the current state of the Semantic Web. Section 2.2 presents an overview of the field of natural language processing, including state of art techniques we applied in this work. Finally, Section 2.3 presents the theory of the Logistic Regression and how we apply it to classification problems.

2.1. The Semantic Web

2.1.1. An Architecture for the Semantic Web

The idea behind the Semantic Web was first presented by Berners-Lee et. al. (2011) which described the evolution of a Web of documents published for human consumption to one that included extra data for computers to manipulate. That extra data should be conceived through a semantic theory such that the information contained in the documents could be interpreted.

The main problem that the Semantic Web aims to solve is the interoperability between systems. The semantic theory provides a set of interpretable symbols such that it offers logical connections between data sources.

The architecture proposed for the Semantic Web in (Berners-Lee, T. et. al. 2011) is depicted in Figure 1. The Coding layer provides character encoding (Unicode) and referencing (URI) mechanism. The Structure layer defines XML as the standard for document exchange and RDF for data representation. The Inference layer adapts resources available on the Web to the definitions stated on the Structure Layer.

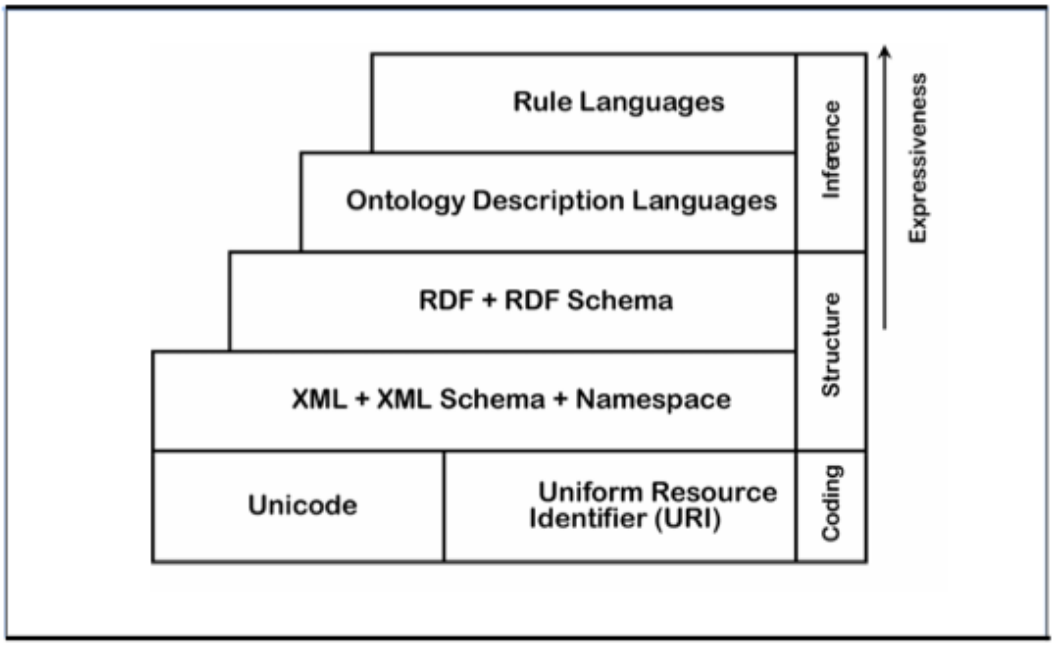

Figure 1: An architecture for the Semantic Web (Available at: http://www.w3.org/2000/Talks/1206-xml2k-tbl/Overview.html)

We discuss in details each layer and the components of the Semantic Web, along with the Linked Data Principles which state rules to achieve the interoperability between systems and data sources. Finally, we overview the Linked Open Data Project, which covers existing datasets available under open licenses in the Semantic Web.

2.1.2. The Coding Layer

The Coding Layer sets standards for character encoding in the Semantic Web and defines a referencing pattern for uniquely identify resources. The Unicode Standard is the universal character encoding standard for written characters and text. It is maintained by the Unicode Consortium and their mission is to create a specification consistent enough to provide ways of encoding multilingual text such that it can be exchange internationally creating a foundation for global software (Allen et al. 2012). As default encoding of HTML and XML, the Unicode Standard is a natural choice for the Semantic Web since the Structure Layer of its architecture is based on XML.

The referencing mechanism defined in the Coding Layer is based on *Uniform Resource Identifiers* (URI). As they identify resources they are an

essential part of the Semantic Web. (Berners-Lee et al. 2005). URIs are independent regarding the context of interpretation, therefore, by theory they have a global scope. Then, once a resource is referenced by an URI, anyone can retrieve a representation of such resource and link to it. Figure 2 illustrates the use of an URI to identify a resource and a possible representation when retrieving the same resource using the URI.

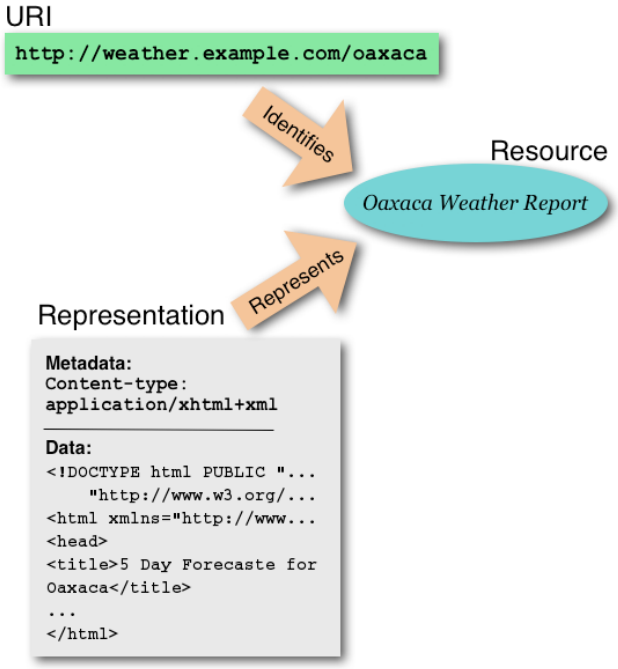

Figure 2: An example of the referencing mechanism using URI (Ian Jacobs 2004).

Considering the Semantic Web aims, URIs provide groundings for both object and relations referencing. When processing data, applications of the Semantic Web process at the bottom layer only URIs.

2.1.3. The Structure Layer

The Structure layer introduces XML as the standard for document exchange and RDF for data representation.

2.1.3.1. XML + XML Schema + XML Namespaces

The *Extensible Markup Language* (XML) is a general purpose markup language that is known for its simplicity to parse, to use and to specify specific purpose markup languages (Bray, T. et al. 2008). XML was developed by a

special working group from the World Wide Web Consortium (W3C). The current W3C Recommendation dates February 2013.

An example of application of XML is showed in Figure 3. In that, the definition of a markup language for defining books is applied. In that case, books are uniquely represented by the key 'ISBN' and contains two attributes 'title' and 'author'.

```
<Books>
    <Book ISBN="0553212419">
        <title>Sherlock Holmes: Complete Novels...
        <author>Sir Arthur Conan Doyle</author>
    </Book>
    <Book ISBN="0743273567">
        <title>The Great Gatsby</title>
        <author>F. Scott Fitzgerald</author>
    </Book>
    <Book ISBN="0684826976">
        <title>Undaunted Courage</title>
        <author>Stephen E. Ambrose</author>
    </Book>
    <Book ISBN="0743203178">
        <title>Nothing Like It In the World</title>
        <author>Stephen E. Ambrose</author>
    </Book>
</Books>
```

Figure 3: An example of XML for representing books.

Any markup language can be defined using XML. One important markup language is the XML Schema that is defined by a set of well-defined constraints on XML. The main ones are Document Type Definitions (DTDs), Relax-NG, Schematron and W3C XSD (XML Schema Definitions). W3C defined DTDs and XSD as its primary schemas languages.

XML Schema can be used for applying a vocabulary of elements and attributes to a document, for associating types such as integers, float, strings, etc to values in the document, to constrain where elements can appear in the document, to provide documentation for the document giving a formal description, etc. The term *validation* in XML Schema context refers to the process of validating a given document to one XML Schema. This step is an important process of quality assurance.

In order to give an example of one application of XML Schema, we shall discuss the concept of XML Namespaces. In the exchange of XML documents, name conflicts may occur. In general, when trying to use different XML

documents in a same application, the semantics behind names defined by the creator of the document can be ambiguous. Consider the example in the Figure 4 containing an example of an XML representing two documents: the first one at the top is a part of an HTML file describing an HTML table element and the second one at the bottom is an XML describing attributes about one table, a furniture.

```
<table>
  <tr>
    <td>First cell</td>
    <td>Second cell</td>
  </tr>
</table>


<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

Figure 4: Two stretches of XML files. At the top an HTML table and at the bottom values about a table (furniture)

If the two stretches of XML files in Figure 4 were used by the same application, a conflict with the name ‘table’ would happen since the word ‘table’ is being interchangeably used across different semantic contexts. XML Namespaces are sets of names defined, generally, in a same context.

XML Namespaces are implemented using prefixes before the name stated for elements in a XML document. Namespaces are defined using the *xmls* attribute. In the example of Figure 4, namespaces would solve the conflict problem. Its implementation is showed in Figure 5.

Figure 5 describes the application of two namespaces *h* and *f*, for HTML tags and for furniture respectively. Each namespace is declared at the begging of the file linking for a resource on the Web were that namespace is defined formally giving semantics for all elements that it defines. The purpose is to give a unique name for each XML Schema.

We are now able to give a complete example of a XML Schema application. Figure 6 shows an example of XML document representing the overview of one shipping order. According to the document, one shipping order is consisted by an order ID represented by the key attribute *orderid*, the name of the person who created the order represented by the element *orderperson*, the name of the costumer represented by the element *shipto*, containing a name, address, city and country, and a sequence of items represented by the element *item*. Every item is defined by the elements *title*, *note*, *quantity* and *price*.

```
<root xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="http://www.w3schools.com/furniture">

<h:table>
  <h:tr>
    <h:td>First Cell</h:td>
    <h:td>Second Cell</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

Figure 5: Use of namespaces in an XML document with name conflicts

In order to be valid for a given application, the document in Figure 6 must be valid against a schema. In that example, the schema is defined in the file *shiporder.xsd*, which comprises a file containing an XML Schema. A complete and valid XML Schema for the XML document in Figure 6 is showed in Figure 7.

The XML Schema in Figure 7 document is defined by the declaration of the element *xs:schema*, defined by the namespace formalized by the W3C. Every XML Schema file must reference the same namespace to be considered a valid XML Schema.

An XML Schema can be interpreted as a tree where every node is constituted of attributes and children nodes with the same definition. In Figure 7, the root element *shiporder* is declared with only one attribute named *orderid* and type *xs:string*. Its children are defined by a sequence of elements inside a complex

type element declaration. Note that the same tree hierarchy presented in Figure 6 is declared in the tree fashion in Figure 7.

Special declarations such as number of occurrences of children are also available in the XML Schema specification. In Figure 7, for example, the element named by *item* is explicitly declared to have no limit of occurrence. The default value for limit of occurrences is 1.

```
<?xml version="1.0" encoding="UTF-8"?>

<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>
```

Figure 6: Example of an XML document representing a shipping order

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="orderperson" type="xs:string"/>
      <xs:element name="shipto">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="item" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="note" type="xs:string"/>
            <xs:element name="quantity"
type="xs:positiveInteger"/>
            <xs:element name="price" type="xs:decimal"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="orderid" type="xs:string" use="required"/
>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Figure 7: Example of an XML Schema that is valid for the XML Document in Figure 6

2.1.3.2. **RDF + RDF Schema**

The Resource Description Format (RDF) is a standard data model defined by W3C. The basic idea behind the RDF is that a resource is described using statements in the form subject-predicate-object expressions. In this context, such statements are known as triples. A subject is the current resource being described, predicates are defined by an URI referencing its formal definition and an object can be a value or an URI referencing other resource.

A triple can be classified by the type of its object. Two classes are defined: datatype property or object property. A datatype property is a triple that its object is a basic value such as integer numbers, strings, float point numbers, etc. An object property is a triple that its object is a URI referencing another resource.

There are several standards to write out an RDF. Those standards are called format serialization syntaxes for RDF. The two most common standards are the Notation 3 (N3) standard and the RDF/XML standard.

In Figure 8, an RDF about a resource describing a resource is serialized in N3 format. In this example there are three triples. The first two ones give geolocation information about a resource identified by URI <*http://www.puc-rio.br*>. The third triple states a title for another URI <*http://www.inf.puc-rio.br*>. The last triple states that the two objects relate to each other by the predicate *edu:hasDept*, which indicates that the subject has a department referenced by the object. The last triple is an object property and the first three triples are datatype properties.

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix edu: <http://www.example.org/> .

<http://www.puc-rio.br> geo:lat "22.97S" ; geo:long "43.23W" .
<http://www.inf.puc-rio.br> dc:title "Departamento de
Informática" .
<http://www.puc-rio.br> edu:hasDept <http://www.inf.puc-rio.br> .
```

Figure 8: Example of an RDF resource in N3 format

The same example in Figure 8 is given in Figure 9 in the RDF/XML format. They describe the same resources with the same number of triples. In the RDF/ XML format triples are grouped by object.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:geo="http://www. w3.org/2003/01/geo/wgs84_pos#"
    xmlns:edu="http://www.example.org/">

    <rdf:Description rdf:about="http://www.puc-rio.br">
        <geo:lat>22.97S</geo:lat>
        <geo:long>43.23W</geo:long>
        <edu:hasDept rdf:resource="http://www.inf.puc-rio.br"/>
    </rdf:Description>

    <rdf:Description rdf:about="http://www.inf.puc-rio.br">
        <dc:title>Departamento de Informática</dc:title>
    </rdf:Description>
</rdf:RDF>
```

Figure 9: Example of an RDF resource in RDF/XML format

In Figure 8 and 9, a graph model is serialized. This graph is depicted in Figure 10.

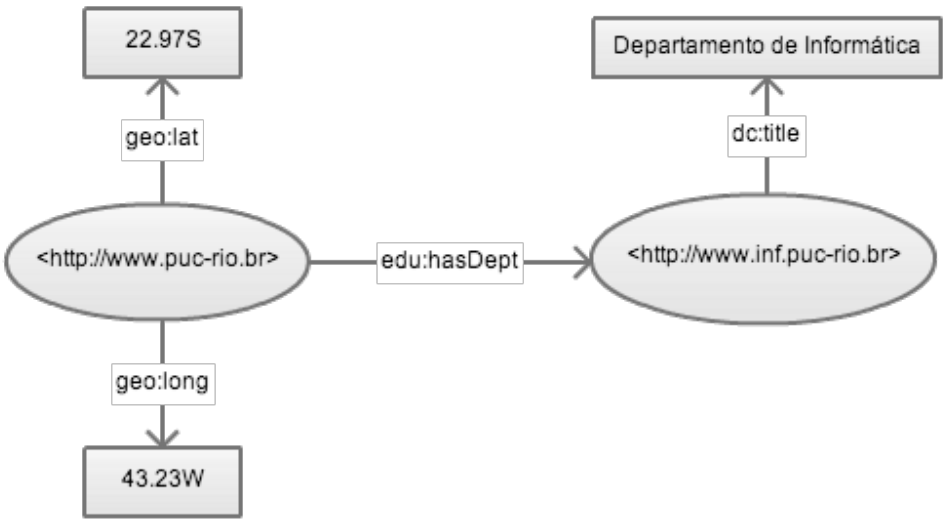

Figure 10: The graph model described in the RDF examples of Figure 8 and 9

In order to structure RDF resources, a set of classes with certain properties which are not specific for any application were published by the W3C as a schema called RDF Schema or RDFS. It provides a framework for describing application-specific classes and properties. The entire definition of RDF Schema is given by Brickley, et al. (2014). The following list gives examples of main constructs from RDFS:

- *rdfs:Class* declares a class for one given resource
- *rdf:Property* defines classes of properties for resources
- *rdfs:domain* declares the class of subjects
- *rdfs:range* declares the class which a object of a triple must belong to
- *rdf:type* states that a resource is an instance of a given class
- *rdfs:subClassOf* defines hierarchy of classes

An example of the use of constructs of RDF Schema is given in Figure 11. In that example, a class *animal* is defined and a subclass *horse* of *animal* is also declared. The use of RDFS is made defining an XML Namespace *rdfs*.

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xml:base="http://www.animals.fake/animals#">

<rdfs:Class rdf:ID="animal" />

<rdfs:Class rdf:ID="horse">
  <rdfs:subClassOf rdf:resource="#animal"/>
</rdfs:Class>

</rdf:RDF>
```

Figure 11: Example of the use of constructs of RDF Schema to define a class hierarchy

2.1.4. The Inference Layer

The Inference Layer in the Semantic Web architecture uses RDF and RDF Schema as mechanisms to describe resources on the Web in a form of ontology descriptions. More semantics can also be achieved with the use of expressive rule languages. Rules are of the form of an implication between an antecedent and consequent. The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

2.1.4.1. Ontology Description Languages

The term *ontology* was defined in philosophy to distinguish the study of "being" from the study of various kinds of beings in natural sciences (Breitman et al. 2006).

In the Semantic Web literature, the most quoted definition for ontology is the one presented by Gruber (1993) which states that "an ontology is a formal, explicit specification of a shared conceptualization". In our context, this means that an ontology must describe an abstract model which is clearly well defined and is stated in a machine processable format.

The W3C Consortium has a more succinct definition stating that: "Ontology is a term borrowed from philosophy that refers to the science of describing the kinds of entities in the world and how they are related" (McGuinness et al. 2004).

Their stated propose suggests that ontologies should provide descriptions for the following elements:

- Classes (or "Things") in the various domains of interest;
- Relationships among "Things";
- Properties (or attributes) that "Things" possess.

In this work we also assume that an ontology defines a concept hierarchy or taxonomy. In the context of computer science, a taxonomy is defined as "the classification of information entities in the form of a hierarchy, according to the presumed relationships of the real-world entities that they represent" (DaConta et al. 2003). A taxonomy classifies terms hierarchically using only the father-son relationship.

Formally, we assume that an ontology defines a set $T \subseteq C \times C$, for the set of classes C, where $T(C_1, C_2)$ indicates that C_1 is a subconcept of C_2. Figure 12 depicts an example of taxonomy for the kingdoms of life defined in biology. In order to formally describe an ontology, several ontology description languages were defined based on RDF and RDF Schema: SHOE, DAML, Oil, DAML+Oil and OWL.

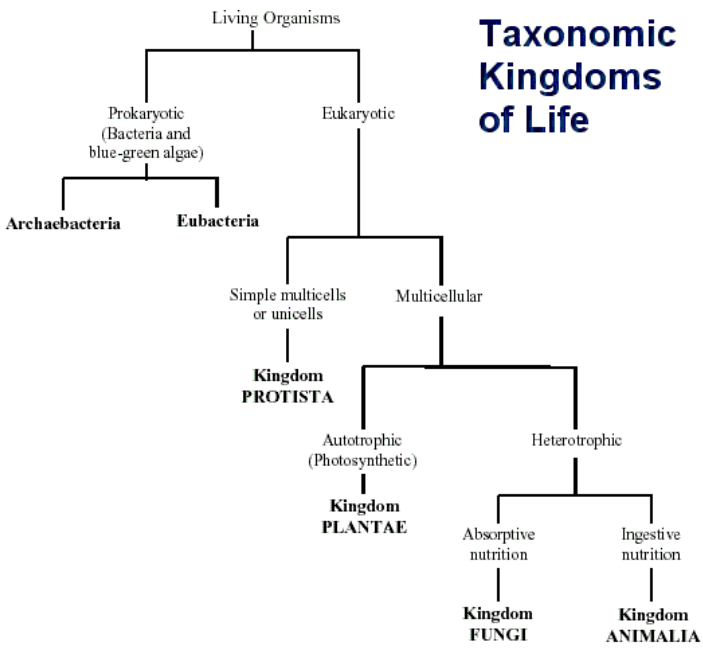

Figure 12: An example of taxonomy of the kingdoms of life

The first ontology description language was proposed at the University of Maryland and it was called Simple HTML Ontology Extension (SHOE). Its basic principle were to use an extension of the HTML language with new tags to semantically annotate Web pages.

Later, released in August 2000, the language DARPA Agent Markup Language (DAML) was published. Sponsored by the Defense Advanced Research Projects Agency (DARPA), the DAML was an extension of RDF and RDF Schema in alignment of the concepts of the Semantic Web.

Meanwhile, the European Community proposed an ontology description language based on description logic. As a result of the On-To-Knowledge Project the language was named Ontology Inference Layer (Oil). This language motivated the development of inference mechanisms to verify the consistency of an specification written in Oil.

In early 2001, DAML and Oil were combined into one description language named DAML+Oil. The effort in joining those two languages was submitted to W3C to be evaluated as a candidate for the ontology description language for the Semantic Web. The committee responsible for defining the standard language was the Web Ontology Working Group (WebOnt).

In February 2004, the WebOnt group finally published the language for describing ontologies in the Semantic Web, the Web Ontology Language (OWL). The OWL language provides additional vocabulary based on description logic to XML and RDF + RDF Schema.

The OWL language has three sublanguages: OWL Lite, OWL DL and OWL Full. The OWL Lite is designed for a primary use consisting of implementations of classification hierarchy and simple constraints. The main purpose of this language is to provide a quick and simple migration from thesauri and other taxonomies to the Semantic Web ontology description language. The OWL DL supports applications that require a maximum of expressiveness while

guaranteeing completeness and decidability. For that reason, the constructs in OWL DL which are all defined by OWL can only be used under logical restrictions. On the other hand, the OWL Full language guarantees the maximum expressiveness of OWL in addition of RDF constructs but there is no guarantee of computability (completeness and decidability). In that sense, an ontology for example can augment the definition of pre-defined vocabulary of RDF or OWL.

Figure 13 gives an example of an OWL file. This example defines an ontology for transport. In this ontology a top class called “Transport” is defined and two sub-classes of “Transport” are declared: “AirTransport” and “LandTransport”. As sub-classes of “LandTransport” the classes “Bus” and “Car” are defined. Note that OWL are just an extension of RDF and RDF Schema.

2.1.4.2. **Rule Inference**

The main purpose of rule inference in the Semantic Web is to find new relationships. All the data presented in the Semantic Web are basically constituted by resources and relationships among them. An inference in this context is the capability of automatic algorithms to generate new relationships among the existent resources based on the data or additional information in the form of vocabulary, e.g. a set of rules.

Inference approaches rely on knowledge representation techniques. In fact, such representations must be expressed in rules that define a general mechanism on discovering and generating new relationships based on the existing ones using for example logic programming. W3C recommends the Rule Interchangeable Format (RIF) for exchanging rules among systems.

The RIF Working Group has been created in 2005 and is focused in two types of dialects: the logic based dialects and dialects for rules with actions. The logic based dialects include languages to express first-order logic often restricted to Horn logic or non-first-order logic being based on logic programming languages. The rules with actions dialects consists basically on the production of rule based system based on existing systems (e.g., Jess, Drools and JRules) and

for systems based on reactive rules of the format event-condition-rules (e.g., Reaction RuleML and XChange).

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE rdf:RDF [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY ns_transport "file://www.ibm.com/WSRR/Transport#">
]>

<rdf:RDF
  xmlns:xsd="&xsd;"
  xmlns:rdf="&rdf;"
  xmlns:rdfs="&rdfs;"
  xmlns:owl="&owl;"
  xmlns:ns_transport="&ns_transport;"
>
  <owl:Ontology rdf:about="&ns_transport;TransportOntology">
    <rdfs:label>A transport classification system.</rdfs:label>
    <rdfs:comment>Cars and buses and some superclasses.</
rdfs:comment>
  </owl:Ontology>

  <owl:Class rdf:about="&ns_transport;Transport">
    <rdfs:label>Transport</rdfs:label>
    <rdfs:comment>Top-level root class for transport.</
rdfs:comment>
  </owl:Class>

  <owl:Class rdf:about="&ns_transport;LandTransport">
    <rdfs:subClassOf rdf:resource="&ns_transport;Transport"/>
    <rdfs:label>Land Transport.</rdfs:label>
    <rdfs:comment>Middle-level land transport class.</
rdfs:comment>
  </owl:Class>

  <owl:Class rdf:about="&ns_transport;AirTransport">
    <rdfs:subClassOf rdf:resource="&ns_transport;Transport"/>
    <rdfs:label>Air Transport.</rdfs:label>
    <rdfs:comment>Middle-level air transport class.</
rdfs:comment>
  </owl:Class>

  <owl:Class rdf:about="&ns_transport;Bus">
    <rdfs:subClassOf rdf:resource="&ns_transport;LandTransport"/>
    <rdfs:label>Bus.</rdfs:label>
    <rdfs:comment>Bottom-level bus class.</rdfs:comment>
  </owl:Class>

  <owl:Class rdf:about="&ns_transport;Car">
    <rdfs:subClassOf rdf:resource="&ns_transport;LandTransport"/>
    <rdfs:label>Car.</rdfs:label>
    <rdfs:comment>Bottom-level car class.</rdfs:comment>
  </owl:Class>

</rdf:RDF>
```

Figure 13: An example of OWL file describing an transport ontology

2.1.5. The Linked Data Principles

One main concern that is not covered by the architecture of the Semantic Web is on how data is published on the Web. The main principle of the Semantic Web is that resources are linked through the definition of well-known relationships. The principles for publishing data are proposed as a Web architecture note by Berners-Lee et. al. (2007). They are called The Linked Data Principles:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, it provides useful information, using the standards (RDF, SPARQL).
4. Include links to other URIs, so they can discover more resources.

The first principle states that any resource in the Web, either the identification of an object or concept, must be referenced by URIs. In this fashion, URIs must reference conceptualizations such as people, places or even abstract ideas as types of relationship, e.g., "knowing somebody".

The second principle guarantees the proper identification of resources referenced by URIs. It states that an URI can be dereferenced over the HTTP protocol into a description of a resource it makes reference to. In the classical Web HTTP URIs are simply used as a simple retrieval mechanism.

The third principle applies standards over the content format retrieved by the dereferencing mechanism. It enables different applications to process Web content. This principle recommends the use of RDF as a standard representation (Heath et al. 2011).

The last principle states that resources must be connected to each other if a semantic link is applicable by one known relationship. In fact, connected resources participate in a same context. For example, if there are resources

describing a country and its capital isolated, they must be connected by the relationship of "is capital of". Therefore, the hyperlinks in Linked Data, known as RDF hyperlinks, connect not only documents, but any type of thing on the Web as well (Heath et al. 2011).

Following the Linked Data Principles, the process of publishing data guarantees machine readability, the explicit meaning of the data, and the possibility to link such data to external datasets (Heath et al. 2011). The set of data published following those principles become part of a single global data space called the Web of Data (Heath, 2009).

2.1.6. The Linked Open Data Project

In February 2007, the *W3C Linking Open Data Project*[1] started under the leadership of Chris Bizer and Richard Cyganiak with the aim of identifying existing datasets available under open licenses and to publish them following the Linked Data Principles (Heath et al. 2011). This project has the participation of universities such as FU Berlin, MIT, KMi/The Open University, Universities of Pennsylvania, Leipzig, London, Hannover, Galway, among others and companies such as OpenLink Software, Talis, Zitgist and BBC (Heath, 2008).

The state of such efforts in September 2011 is depicted in Figure 14. The metadata collected is curated by the Data Hub[2] members. In Figure 14, each node represents a dataset published following the Linked Data Principles and each vertex represents links between resources of datasets. The thickness of every vertex represents the number of links between resources. This set of datasets was named the Linked Open Data Cloud (LOD Cloud).

Currently, the LOD Cloud comprehends 884 datasets[3] and is maintained by the LOD within the Comprehensive Knowledge Archive Network (CKAN[4]), a generic catalog that lists open-license datasets.

---

[1] http://linkeddata.org/

[2] http://datahub.io/

[3] http://datahub.io/dataset?tags=lod

[4] http://ckan.org/

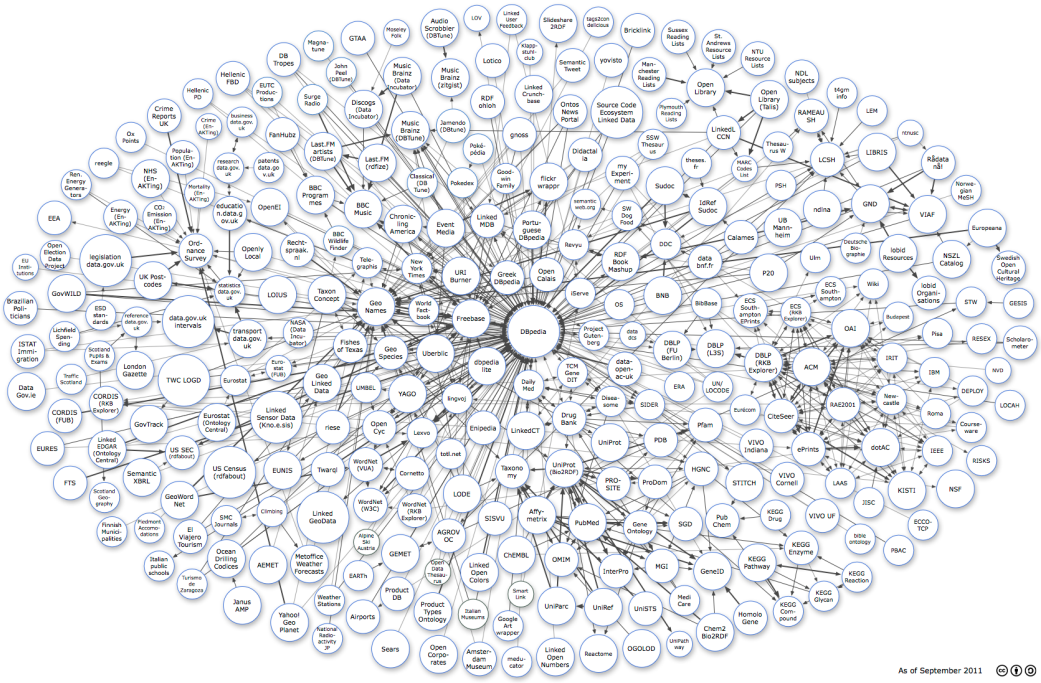

Figure 14: The LOD Cloud Diagram at September 2011[5]

2.2. **Natural Language Processing**

Natural Language Processing (NLP) is a research field that studies and develops algorithms able to understand and manipulate natural language text or speech for a plethora of applications. The foundations of NLP lie in a wide range of disciplines such as computer and information sciences, linguistics, mathematics, electrical and electronic engineering, artificial intelligence and robotics, psychology, etc. (Chowdhury 2003). In the context of this work, we use two natural language processing techniques: sentence boundary disambiguation and part-of-speech tagging.

2.2.1. **Sentence Boundary Disambiguation**

The Sentence Boundary Disambiguation (SBD) is a problem associated with the disambiguation of sentence boundary markers in natural text. As a result, from a given text, a list of sentences can be extracted for further processing.

[5] http://lod-cloud.net/

A sentence is a fundamental and well-known unit in theoretical and computational linguistics. Other sub-units in linguistics are constrained by the abstract concept of sentence, which they are confined by sentence boundaries. The conceptualization of such boundaries makes possible the processing and understanding of sentences. Although they serve well for this purpose, boundary graphemes, which are the smallest semantically distinguish unit in a written language, occur more often for more than one single purpose. One example of sentence boundary, the period, can be used for abbreviations, initials, numbers and ellipses (Nunberg, 1990) for a linguistic discussion of punctuation and the ambiguity of the period). Those examples can occur at the same time inside a sentence, for example, what makes sentence detection a disambiguation problem.

Consider the following sentence extracted from an article in Wikipedia[6]:

"On December 6, 2013, Apple Inc. launched iBeacon technology across its 254 U.S. retail stores."

Although we can notice four uses of a period in the above sentence, only the final period is a sentence boundary marker. This is an example of how complex the sentence boundary disambiguation task can be.

The main strategy used by all the approaches on SBD is detecting abbreviations, initials, numbers and ellipses and, by exclusion, the remaining occurrences of periods are sentence boundaries. Surveys on SBD systems use an standard English dataset: sections 3-6 of the Wall Street Journal (WSJ) corpus (Marcus et al., 1993) containing around 27,000 examples.

Most of the approaches are based on lists of hand-crafted regular expressions and abbreviations. The Alembic system (Aberdeen et al., 1995), using this approach, achieves an error of 0.9%. Even though the error rate is very low, this approach is very dependent on the language and genre.

Supervised Machine Learning (SML) methods were also proposed to solve this problem. Palmer and Hearst (1997) presented Satz which is based on part-of-speech features as input to a neural net classifier using held-out training on the

[6] http://en.wikipedia.org/wiki/Apple_Inc.

WSJ dataset. Their features were generated using a 5000-word lexicon and a list of 206 abbreviations. They achieved an error rate of 1.0%.

Also using SML, another classifier was proposed by Reynar, J. et al. (1997) using a maximum entropy method. Named mxTerminator, it was trained on nearly 1 million words of additional WSJ extracting simple lexical features of words to the left and right of the candidate period. It achieved an error rate of 1.2%.

Unsupervised systems were also adopted to solve the SBD problem. Two systems are notable. The first one, Plunkt (Kiss et al., 2006), uses a log-likelihood based heuristics to infer abbreviations and common sentence starters from a large corpus. Although it presents an error rate of 1.65% on the WJS dataset, Plunkt is easily adaptable but requires a large unlabeled corpus of the domain of discussion. Plunkt is the system presented in the NLTK library (Loper et al., 2002) for Python systems.

The second system, presented by Mikheev (2002), is based on heuristics to decide whether words correspond to abbreviations and names or not. The original proposal achieves an error rate of 1.41% but it can be improved by the addition of extra tuned lists from other news groups achieving rates of 0.45% and in combination with POS-based supervised systems give the best error rate of 0.31% until 2009.

Gillick (2009) based his work on a supervised machine learning method. He used Support Vector Machine with linear kernel to generate a classifier using features based on words at the left and right of the period candidate. After a study of combinations of features proposed by previous systems, they found that popular features, such as lists of abbreviations, increase the error rate and should not be considered. Their classifier, named Splitta, use eight features and achieved an error rate of 0.25% on the WSJ corpus. Although this system achieved the best accuracy, tests were only made with the English language.

2.2.2. **Part-of-Speech Tagging**

Part-of-Speech (POS) tagging consists of assigning grammatical classes to each word in natural language sentences. This is one of the most well studied

problems in Natural Language Processing. In fact, state-of-the-art POS tagging accuracies in a large range of languages are near 95%, English being 97.50% (Sogaard et al., 2011).

For the English language, the most commonly used tag set is the Penn Treebank (Mitchel et al., 1994) containing 36 tags of variations of verbs, nouns, adjectives, adverbs, pronouns, interrogatives, etc. For example, consider the following sentence:

"They refuse to permit us to obtain the refuse permit"

A POS tagging for the given sentence follows bellow.

('They', 'PRP'), ('refuse', 'VBP'), ('to', 'TO'), ('permit', 'VB'), ('us', 'PRP'), ('to', 'TO'), ('obtain', 'VB'), ('the', 'DT'), ('refuse', 'NN'), ('permit', 'NN')

In this example, PRP stands for personal pronoun, VBP for verb in the 3rd person in the singular present, TO for the word *to*, VB for verb in the base form, DT for determiner and NN for noun in singular form. Notice that the word *refuse* in the example above illustrates a case of ambiguity. The ability to handle ambiguity is the main aspect that defines the accuracy of a POS tagger system.

The majority of taggers applies machine learning algorithms on the Penn Treebank WSJ corpus, that contains over one million annotated words from a 1989 Wall Street Journal material. All taggers in the literature use lexical information as features for their classifiers.

For the English language, POS taggers have been developed using several supervised machine learning techniques, most of which uses Hidden Markov Models (HMM) (Brants 2000), maximum entropy models (Ratnaparkhi 1996; Toutanova et al., 2000; Toutanova et al., 2003), conditional random fields (Lafferty et al. 2001), perceptron learning (Collins, 2002) and bidirectional sequence classification (Shen et al., 2007). The state-of-the-art, however, applies a semi-supervised method based on a condensed nearest neighbor classification (Sogaard et al., 2011).

The three POS tagger systems with best accuracy in the literature are the SCNN (Sogaard et al., 2011) with 97.50% of token accuracy, LTAG-spinal (Shen

et al., 2007) with 97.33% and the Stanford Tagger 2.0 (Manning, 2001) with 97.32%.

2.3. Classification Methods

2.3.1. Overview

Classification is the problem of identifying to which category from a set of categories a new observation belongs on the basis of a training set of data containing observations whose category is known. In the machine learning context, an observation is called instance and category is called class.

Classifiers are separated into two categories: binary classifiers and multi-class classifiers. This categorization considers the number of classes or set of categories a problem has. If the set of classes contains only two classes, for example, a spam or not a spam, sick or not, etc. A classifier is considered a binary classifier. If the set of classes contains more than two classes, a classifier is considered a multi-class classifier.

Every instance is interpreted by classifiers as a feature vector representing measurable properties of the instance. Every property is called a feature, also known as explanatory variable in the statistics context. The vector space associated with feature vectors is called the feature space.

There are many classification algorithms (Kotsiantis 2007). The simplest one is the nearest neighbor. Its derivative, *k*-nearest neighbors, is more popular. They are examples of memory-based or instance-based classification methods which basically compare new observations with previous instances seen in the training set, instead of performing explicit generalization. Non-memory-based methods, which create an explicit generalization model, include neural networks, decision-trees , Bayes classifier, SVM's etc.

In this work, we use a non-memory-based method called Logistic Regression. We apply its multi-class variation, called Multinomial Logistic Regression.

$x^{(i)}$

$$h(x^{(i)}) = y^{(i)}, 1 \le i \le m$$

2.3.2. Logistic Regression

$(x^{(i)}, y^{(i)})$

$\{(x^{(i)}, y^{(i)}) | 1 \le i \le m\}$

To establish a notation, consider a training set with *m* classified examples of data. Associated with each example *i* we have its feature vector denoted by χ and its class by γ. An *instance* is a pair $(x^{(i)}, y^{(i)})$ and, hence, we have a set of instances $\{(x^{(i)}, y^{(i)}) | 1 \le i \le m\}$ representing our training set.

We denote the feature space by χ and the space of classes by γ. The learning process is represented by a function $0 \le h_\theta(x^{(i)}) \le 1$, called the *hypothesis*, defined by the logistic function or sigmoid function as:

$$h_\theta(x^{(i)}) = g(\theta^T x^{(i)}) = \frac{1}{1 + e^{-\theta^T x^{(i)}}}$$

$$\dim(\theta) = \dim(\chi) = n$$

where

$$g(z) = \frac{1}{1 + e^{-z}} \qquad h_\theta(x^{(i)}) = y^{(i)}$$

and θ is a vector such that, $h_\theta(x^{(i)}) = y^{(i)}$, called parameter of the model. Figure 15 depicts a plot of the sigmoid function. The learning process constitutes an iterative model to find the optimal θ that best satisfies $h_\theta(x^{(i)}) = y^{(i)}$.

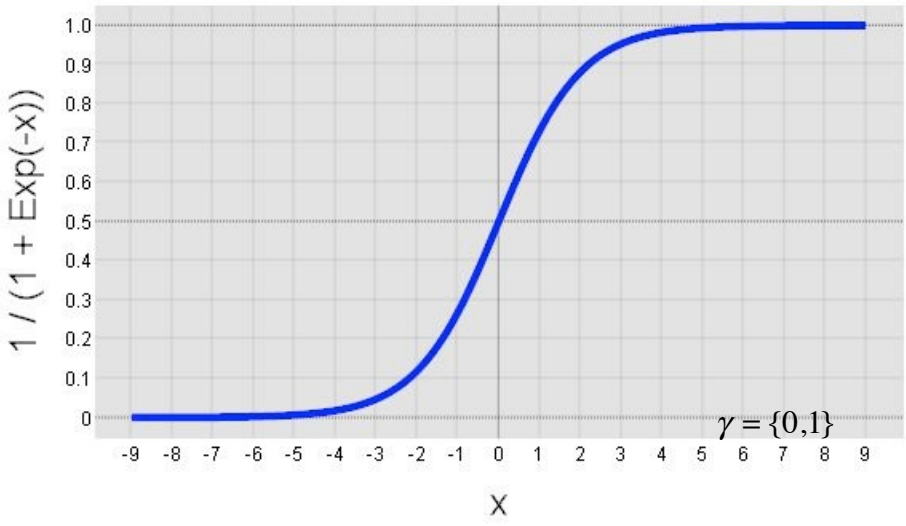

Figure 15: An example of sigmoid function

Assume that our model generates a binary classifier, i.e., $y^{(i)}$. Then, if $h_\theta(x^{(i)}) < 0.5$, predict $y = 0$ and if $h_\theta(x^{(i)}) \ge 0.5$, predict $y = 1$. Also, assume that:

$$P(y^{(i)} = 1 \mid x^{(i)}; \theta) = h_\theta(x^{(i)})$$
$$P(y^{(i)} = 0 \mid x^{(i)}; \theta) = 1 - h_\theta(x^{(i)})$$

where P represents the probability that a class $(x^{(i)}, y^{(i)})$ is 1 or 0, conditionally to the features and parameters $\{(x^{(i)}, y^{(i)}) \mid 1 \le i \le m\}$.

Note that the statements above can be written as: $X_{(m,n)}$

$$P(y^{(i)} \mid x^{(i)}; \theta) = (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$$

for a given instance $x^{(i)}$ $\vec{y}$ $y^{(i)}$.

Considering all instances θ from the training set, let us define a feature matrix by $P(\vec{y} \mid X; \theta)$ where each line represents a feature vector θ of dimension *n*. Also, let us define a class vector , where each dimension is represented by .

So, given , we want to calculate as a function that describes the total probability that a model defined by fits the instances of our training set. This function can be written as:

$$L(\theta) = L(\theta; X, \vec{y}) = P(\vec{y} \mid X; \theta)$$

We assume that instances are independent from each other. Hence, we can write: θ

$$L(\theta) = \prod_{i=1}^{m} P(y^{(i)} \mid x^{(i)}; \theta)$$
$$= \prod_{i=1}^{m} (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$$

$L(\theta)$ $L(\theta)$ $L(\theta)$ $l(\theta)$

Here we apply the principle of maximal likelihood to choose that maximizes .

In fact, instead of maximizing , we can maximize any strictly increasing function of . In order to simplify derivations, we choose to maximize the *log likelihood function* , defined as: $l(\theta)$ θ θ $l(\theta)$

$$l(\theta) = \log(L(\theta))$$
$$= \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

α

In order that find θ that maximizes , we will use the gradient ascent algorithm, which starts with an initial guess for and, at every iteration, changes so that increases until convergence is reached. The update at every iteration is given by:

$l(\theta)$ h_θ $P(\vec{y} \mid X;\theta)$

$$\theta_{j+1} = \theta_j + \alpha \nabla_{\theta_j} l(\theta)$$

In the equation above, is the learning rate. Taking derivatives, we write the update rule as:

$$\theta_{j+1} = \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$

h_θ

Iterations occur until convergence is reached, i.e., successive values of vary by a very small number.

At the end of the process, we have a binary regressor that maximizes .

2.3.3. **Multi-class Perceptron**

A multi-class perceptron can be computed using a binary regressor based on Logistic Regression. The classifier is calculated using a one-vs-all strategy. Using the same definitions described for the binary class Logistic Regression, calculate:

$$h_\theta^{(j)}(x^{(i)}) = P(y = j \mid x^{(i)}; \theta)$$

for each class *j*. Therefore, in this case:

$$P(y^{(i)} = j \mid x^{(i)}; \theta) = h_\theta^{(j)}(x^{(i)})$$
$$P(y^{(i)} \neq j \mid x^{(i)}; \theta) = 1 - h_\theta^{(j)}(x^{(i)})$$

On a new observation *x*, we apply the one-vs-all strategy choosing the class *j* such that returns the biggest value of hypothesis function:

$$j = \arg\max_j h_\theta^{(j)}(x)$$

2.4. Summary

This chapter presented the main concepts related to the solution proposed in the dissertation. Section 2.1 presented the architecture of the Semantic Web. Section 2.2 gave an overview of the principles and related work to two problems in the Natural Language Processing field: sentence boundary detection and part-of-speech tagging. Section 2.3 introduced classification methods and described the Logistic Regression methods for the binary and multi-class classifiers.