Automatic Knowledge Graph Construction from Events in News Articles and Public Sentiment from Twitter

Gustavo Panez Velazco B.Sc., M.B.A.

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Future Networked Systems)

Supervisor: Declan O'Sullivan

Assistant Supervisor: Fabrizio Orlandi

August 2019

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Gustavo Panez Velazco

August 8, 2019

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Gustavo Panez Velazco

August 8, 2019

Acknowledgments

I would like to acknowledge the support of my supervisors in providing guidance for this dissertation.

I would also like to thank my parents, my sister and my girlfriend for their continuous support during my Master's studies.

Gustavo Panez Velazco

University of Dublin, Trinity College
August 2019

Summary

Knowledge graphs can model information and support question answering use cases better than traditional information retrieval systems. Having as motivation the goal of helping users find better answers to event related questions, this dissertation introduces NewsTextAnalyzer, a system that automatically builds a knowledge graph from assertions and events extracted from a large corpus of political news articles. The system employs a lightweight NLP pipeline to uplift information from unstructured data and provides structure to that information in order to create a knowledge graph. In addition, NewsTextAnalyzer recovers the sentiment that the public expressed on Twitter as a reaction to specific events reported in the news, and associates such sentiment to specific assertions that exist in the graph. This ability of NewsTextAnalyzer to not only report what happened but also how people reacted to it, differentiates it from similar systems.

NewsTextAnalyzer leverages open information extraction methods to recover knowledge from the articles. Furthermore, NewsTextAnalyzer can recognize and resolve entities, specifically politicians mentioned in the articles' text and link their mentions to their resource representation in DBpedia. This capability allows NewsTextAnalyzer to support complex queries that require event information from its local graph and factual information that exists in DBpedia. In addition, the system can detect temporal expressions, normalize them, and associate them to appropriate assertions. NewsText-

Analyzer also provides a user interface through which users can build queries to surface event information, including the associated sentiment. For example, users can retrieve all the events in which a politician was involved, or all the events concerning a type of politician, such as Senators.

This dissertation provides information about background and related work in information extraction and knowledge graph construction approaches, details about the design and implementation of the NewsTextAnalyzer system, a qualitative evaluation of its capabilities compared to similar systems, directions for future work to improve the performance of the system, and the conclusions and findings of the project.

Contents

List of Figures

Chapter 1

Introduction

Information retrieval (IR) is usually defined as the task of finding text documents from within collections to satisfy a user's information need . Search engines are applications that implement IR techniques and apply them to a large corpus [1], the most famous search engine being Google's. Typical use of a search engine involves the user typing a query to express his information need and the system returning a ranked list of relevant documents. To be able to conduct the search in a reasonable amount of time, search engines index the collection of documents in an offline step. Indexing is the process of calculating the importance of words within the documents and storing such information in special data structures such as lookup tables and inverted indexes which are later used during the querying process. A retrieval model is the method that governs how word importance is computed. Most retrieval models utilize statistics derived from the frequency a word occurs within a document and the frequency it occurs over the entire corpus [2].

The prevalence and popularity of search engines on the Internet demonstrates that IR systems work well when users are looking for documents about a topic. However, in other situations, users are looking for answers to questions they have. In such cases, the search engine puts considerable amount of work onto users, who have to go through the search result list and inspect each document [3] to see if they can get an answer to their question. This becomes even more difficult for users when there is information overload, that is when there are many documents related to the question that do not

clearly provide an answer.

Some users' questions could be complex, such as "Which Democratic Senators left office during President Obama's term?". In such cases, it is possible the search terms used in the query are not present in the documents that hold the answer [4]; hence, the search results would not satisfy the user, who will need to spend time in thinking in another way to express his information need. In addition, it is possible that a specialized corpus does not even have documents with important pieces of the query. For example, a corpus about U.S. financial transactions, such as mergers and acquisitions, might not have documents with information about the Federal Trade Commission (FTC), the organization that reviews and approves such operations. Given the query "What were the deals that were rejected when Joseph Simmons was Chairman of the FTC?", it would be difficult for a search engine to surface useful documents just based on the information of said corpus. These limitations arise from the fact that IR systems tend to be isolated units that do not "read" or "understand" what is said in documents, they do not extract meaning from the data, which in turn hampers their ability to satisfy question and answering use cases [5].

On the other hand, Semantic Web[1] technologies and tools have been designed with the specific purpose of modeling complex relations between entities through knowledge graphs. Knowledge graphs definitions vary but the Journal of Web Semantics defines them as "large networks of entities, their semantic types, properties, and relationships between entities" [6]. One could think of a knowledge graph as a structured representation of a set of facts or assertions, see Figure 1.1 for a graphical representation. Thus, while IR systems leverage statistics derived from words and documents, knowledge graphs go deeper and capture meaning.

Knowledge graphs can hold heterogeneous data because they model facts using triples, simple constructs composed of a subject, a predicate and an object. That is, a subject can have pieces of information associated to it via many different predicates. On knowledge graphs, one can find resources and literals. Resources are identified

[1] `https://www.w3.org/standards/semanticweb/data`

through the use of Uniform Resource Identifiers (URI)[2]. Literals can be of different types such as strings or dates. Because resources have an identifier, triples can be created to describe relationships between resources [7].

By using triples, knowledge graphs can describe diverse facts such as "New York City is part of the U.S." or "Donald Trump's birth date is June 14, 1946", without the need of defining a schema in advance as relational databases do. For example, the fact that Donald Trump was born in New York City, graphically represented in Figure 1.1, could be formally described with the triple depicted in Figure 1.2.

The triples that form a knowledge graph can be serialized through Semantic Web technologies such as the Resource Description Framework (RDF) [8] and the Terse RDF Triple Language (Turtle) [9]. Moreover, triplestores [7], purpose-built databases that can handle RDF data, allow applications to write and read data from the graph through the use of the SPARQL Protocol and RDF Query Language (SPARQL) [10], a SQL-like query language. Triplestores have the ability to find relation patterns within the graph, enabling them to answer complex queries. Triplestores usually expose a query service or endpoint to which applications can send SPARQL queries over the HyperText Transfer Protocol (HTTP)[3]. The triplestore parses the query and navigates the graph of information to find fragments of the graph that comply with a set of conditions [11]. In addition, it is even possible for a system to reason over the information contained in a knowledge graph, which would allow discovering facts that are not explicitly stated in it [7].

Linking data across knowledge graphs allows triplestores to resolve queries that require information distributed among various knowledge bases that can be administered by different institutions. In this fashion, different organizations can create specialized knowledge graphs and interlink the overlapping resources. For example, a knowledge graph about medications and the pharmaceutical companies that produce them, could interlink to companies in another knowledge graph about enterprises listed in the New York Stock Exchange and their market value. The interlinking could enable answering

[2] `https://tools.ietf.org/html/rfc3986`
[3] `https://tools.ietf.org/html/rfc7231`

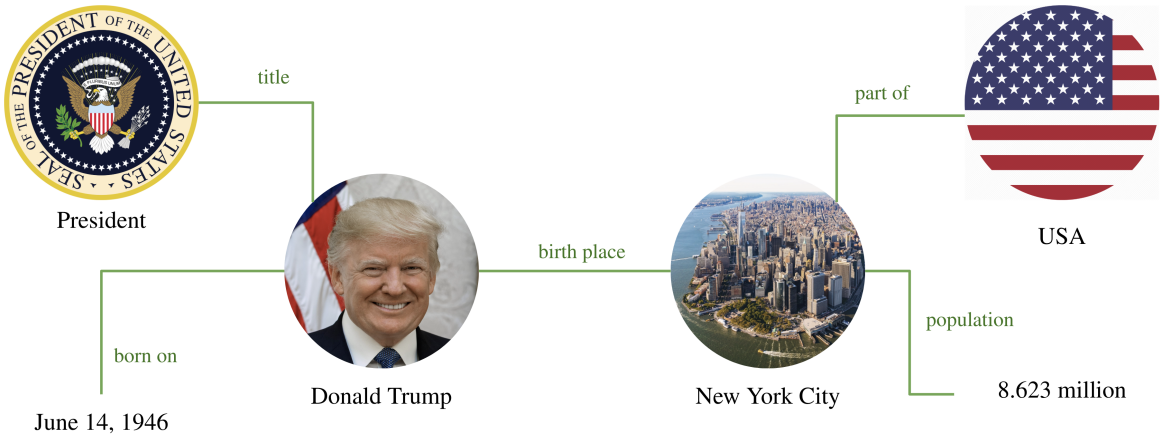


Figure 1.1: Representation of a small knowledge graph

Subject	<http://dbpedia.org/resource/Donald_Trump>
Predicate	<http://dbpedia.org/ontology/birthPlace>
Object	<http://dbpedia.org/resource/New_York_City>

Figure 1.2: Example of a formal triple

queries by using information stored in the different graphs, e.g., "How the market value of pharmaceutical companies changed when certain medication was introduced?". This interlinking capability allows the enrichment and expansion of knowledge graphs as new information surfaces or new requirements arise but reduces repetition when possible.

One of the milestones in the adoption of knowledge graphs by the technology industry occurred when in 2010 Google acquired Metaweb Technologies[4]. At that time, Metaweb had developed a knowledge graph called Freebase [12] that aimed to be a massive database of the world's knowledge. Freebase contained data from sources such as Wikipedia[5], but also information contributed and verified by a community of editors. Freebase would become a core component of the Google Knowledge Graph, introduced in 2012, and the Google Knowledge Graph in turn would become an important signal that now Google Search uses to decide which information to surface to users as search

[4] https://googleblog.blogspot.com/2010/07/deeper-understanding-with-metaweb.html
[5] https://www.wikipedia.org/

results[6]. Since then, many technology companies, such as Amazon, Microsoft, Apple, and Facebook, have developed knowledge graphs to power software designed to satisfy question and answering use cases, such as intelligent assistants. For example, to the question "Who is the president of the U.S.?" an intelligent assistant could conclude that the likely answer is "Donald Trump" by using the knowledge graph depicted in Figure 1.1 to realize that:

- President is a kind of title that people can have
- Countries, such as the U.S., have cities, such as New York City
- Donald Trump was born in the U.S. because he was born in New York City
- Donald Trump is the likely answer to the query because he has the "President" title and is American

Initially knowledge graphs were constructed manually or through the transformation of structured data sources, such as relational databases [13]. However, in the last decade, researchers have built systems that construct knowledge graphs from structured and semi-structured data automatically, such as DBpedia [14]. Because many of these systems use Wikipedia as source, the resulting knowledge graphs hold encyclopedic information. That is, the Wikipedia-based knowledge graphs mostly contain general world knowledge and facts that hold true at a specific period of time but does not capture the history of changes.

To extend the kind of information ingested into knowledge graphs, researchers have been looking to capture events, where events are defined as actions that happened at a certain point in time. Popular semi-structured data sources to gather events from include curated Wikipedia pages such as the Wikipedia Current Events Portal (WCEP)[7]. However, because the event coverage of such sources is limited, researchers developed systems that could detect and extract events from news articles, such as

[6] `https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html`

[7] `https://en.wikipedia.org/wiki/Portal:Current_events`

XLike [15]. Every day, major publications produce high quality journalism pieces that cover events of interests all around the world in almost real time. These events evolve as journalists learn new details about them and update their initial stories. To be able to extract knowledge from the articles' texts, most of the advanced automatic knowledge graph construction systems utilize sophisticated Natural Language Processing (NLP) pipelines that could take more than a minute to process a single news article.

1.1 Motivation

Event information captured by the aforementioned systems include participants, date, location and provenance. An indirectly related but interesting component of an event is how people reacted to it. Such sentiment could be captured from social media sites and associated to events extracted from news. However, it appears that a system that automatically builds a knowledge graph from news articles and enriches it with sentiment data has not been designed yet. Thus, the main motivation behind the project is helping users get better answers to their event related questions than what a typical IR system could offer by surfacing the event's sentiment.

1.2 Contribution

This project aims to address the gap described before by proposing NewsTextAnalyzer, a system that integrates sentiment data and event information into a single representation. It uses an NLP pipeline to extract event information quickly from news articles, and calculates and integrates sentiment from public reaction on Twitter[8] to a granular event reported by the news. In addition, NewsTextAnalyzer links data to DBpedia, which allows it to answer questions that require event information from NewsTextAnalyzer and factual information that NewsTextAnalyzer has not observed but exists on DBpedia.

[8] `https://twitter.com`

1.3 Research Questions

The questions that the project aims to answer include:

- How can events be quickly uplifted from a large corpus of unstructured data such as hundreds of thousands of news articles?
- How can events be represented in a knowledge graph to support temporal-related queries?
- How can sentiment about specific events be captured from social media and integrated into the knowledge graph at a per-event level basis?

1.4 Context

To ground the project in a specific use case, NewsTextAnalyzer focuses on event question and answering within the political space. The project explores how a knowledge graph could be used to better understand political events. A system such as NewsTextAnalyzer could potentially help political science researchers that want to understand how public opinion shifted as a result of policy or government administration changes. One could imagine that a system with wide coverage over many news publications could provide more up to date opinion sentiment about a politician than a poll. Granted, such information might be noisy but could be gathered at a fraction of the cost. For example, the campaign team of a presidential candidate could use a system such as NewsTextAnalyzer to assess if the candidate's speeches and policy announcements are making a difference on how people feel. Moreover, political science historians could use this system to analyze the events that changed perception the most about a politician during his career or identify all the events in which a politician was involved. From a more general perspective, a system such as NewsTextAnalyzer could support curiosity questions coming from regular Internet users, such as "Which were the major cabinet resignations during Theresa May's premiership?".

1.5 Document Structure

This document is organized in the following way: Chapter 2 presents background information and related works about information extraction and knowledge base construction approaches which are central to this project. Chapter 3 explains the most important design decisions made with respect to the NewsTextAnalyzer system. Chapter 4 goes into the implementation details of the different system components including its NLP pipeline, its enrichment module and its simple querying interface. Chapter 5 provides a qualitative evaluation between NewsTextAnalyzer and similar systems developed by research teams in the last 5 years. Chapter 6 describes areas of opportunity to improve the system with future work. Lastly, Chapter 7 presents the conclusions.

Chapter 2

Background and Related Work

This chapter covers different approaches for relation extraction, a subtask of information extraction, as well as a new paradigm called open information extraction. Relation extraction and open information extraction are critical methods that support uplifting knowledge from text, e.g., detecting what happened or identifying who conducted the action. In turn, the extraction of knowledge is an important step of the process of automatically building a knowledge graph.

In addition, this chapter presents examples of systems that automatically build knowledge bases from structured, semi-structured and unstructured data. The typical knowledge graph construction process involves additional tasks to knowledge extraction such as entity disambiguation. These systems also describe different ways to model events and their properties, which this project uses as inspiration to represent the events it extracts, including the sentiment attached to them.

2.1 Relation Extraction

In [16], Jurafsky and Martin define information extraction as the task of converting unstructured information, specifically text, into structured data. The goal of information extraction is to extract meaning from sentences, usually in a limited way. Through years of research, specific subtasks have been defined as part of information extraction

such as named entity recognition (NER), coreference resolution, entity linking and relation extraction. The named entity recognition task consists in finding words within a sentence that refer to a subset of classes or entities [16]. Some systems are built with the goal of identifying very specific entities, such as diseases, medications, and biological reactions, but the vast majority of named entity recognizers detect much more general entity classes such as "person", "organization", and "location". Coreference resolution consists in identifying words or tokens that refer to the same entity in a document [16]. Entity linking or entity resolution is a complementary task to coreference resolution and consists in resolving the identified entities to their representations in a database [16]. Lastly, relation extraction is the task of determining if a relation exists between two entity mentions in a sentence, and if one exists, categorizing the relation into a class [16], such as "capital of" or "born in".

In the last couple of years, advances in the entity recognition subtask have made vast contributions to solve the problem, with modern English entity recognizers achieving over 90% F-measure, according to Yadav and Bethard in [17]. Publicly available NLP tools such as Stanford CoreNLP[1] and Apache Open NLP[2] usually incorporate the latest ideas from published by the research community allowing their entity recognizers to perform well too. On the other hand, relation extraction is a more challenging task than entity recognition and still requires more research to achieve accuracy and recall levels comparable to entity recognition. For that reason, the next subsections of this chapter survey the literature to provide examples of different relation extraction approach and methods.

2.1.1 Supervised Approaches

In supervised relation extraction, there is a set of specific relations of interest defined in advance that are provided to the system. Supervised approaches require training data in which each pair of entity occurrences have been labeled with one of the specific relations of interest. For cases in which a relation between entity mentions does not

[1] `https://stanfordnlp.github.io/CoreNLP/`
[2] `https://opennlp.apache.org/`

exist in the training data, a special relation marker could be used to represent such fact. Supervised approaches tackle the relation extraction task as a classification one. They build multi-class classifiers in which each class represents one of the predefined possible relations of interest.

In [18], Kambhatla used the Automatic Content Extraction (ACE) data from [19] as training set. The ACE data contains many manually annotated sentences in which entity mentions and their relations have been labeled. Kambhatla trained maximum entropy models that used lexical, syntactic and semantic features derived from the training sentences to categorize pair of mentions into one of 49 classes. In terms of syntactic features, the author relied on the mentions' tokens, i.e., words, and the words between them. Other syntactic features included Part of Speech (POS) tags and chunk labels. The system devised by the author also performed NER on the sentence and used the mentions entity types as features. The most advanced features were derived from dependency parse trees. A dependency parse tree is constructed by examining a sentence and identifying relations between head words and modifier words within phrases. A head word is a word within a phrase that determines to which syntactic category the phrase belongs. For example, in the sentence "American Airlines canceled the evening flights to Los Angeles", the head word for the noun phrase "the evening flights to Los Angeles" is "flights" while "evening" and "to Los Angeles" are just noun modifiers applied to "flights". Furthermore, the action "canceled" has the noun phrases "American Airlines" and "the evening flights to Los Angeles" as dependents, with the former acting as subject and the latter as object. This kind of analysis allowed the author to incorporate features that indicated whether the entity mentions where in the same noun phrase or verb phrase, or even the path that connected the mentions.

In [20], GuoDong et al. built on top Kambhatla's research, incorporated new syntactic features to train a support vector machine (SVM) classifier, and were able to achieve over 55% F-measure in extracting 24 ACE relation subtypes, a three point increase compared to Kambhatla's method. The new features that allowed the classifier to make better predictions included incorporating those derived via phrase chunking. GuoDong et al. identified the phrase head words between the mentions and classified them in 3 types before including them as features; the types were first word, last,

and other. In addition, they introduced the phrase head words before and after the mentions, and the headwords of the entity mentions themselves. The authors also compiled lists of special entities, such as country names or relative relations, and introduced features that indicated if one of the mentions were within such lists. Through the use of these list-based features, GuoDong et al. were more successful in correctly identifying location-based relations, e.g., "citizen of", and social relations, e.g., "parent of".

Nguyen et al. employed a supervised approach to extract relations from Wikipedia through SVM classifiers [21]. First, the authors identified entity mentions through analysis of the sentences' syntax and by inspecting their chunk tags. Then, they resolved all references to the same entity via an ad-hoc coreference resolution method. With this information, SVM entity classifiers can correctly assign an entity type, e.g., "organization", to an entity mention. The features the SVM entity classifiers use include the category and parent categories of the source Wikipedia article, the most frequent pronouns, and the singular nouns that appear on the first sentence of the article. The entity types themselves then become features of a relation extraction classifier. Nguyen et al. believed that there are keywords that are strong indicators of relations, e.g., the presence of the word "cofounder" provides a significant signal that the founder relation might be expressed in a sentence. To find potential high signal keywords, they used a semi-automatic method that extracts entity mentions that have a known relation from the summary sections of Wikipedia articles, and used those entity mentions to search for even more sentences within Wikipedia. Using dependency parsing, Nguyen et al. processed the recovered sentences, evaluating the words in the path between the pair of entity mentions and ranking those words using a Term Frequency - Inverse Document Frequency (TF-IDF) based model. The TF-IDF model calculates the importance of words based on how many times a term appears on a document but also on the term specificity, that is how uncommon a term is in a corpus [1]. Nguyen et al. manually selected the words with highest relevance score for each relation of interest and obtained the final high signal keyword lists. During training and classification, the authors expanded the idea of dependency tree into a core tree. A core tree contains paths from elements in the dependency tree to keywords that appeared in the sentence, which allows identification of relations between entities even if they are stated in an indirect way. For example, from the sentence "Steve Ballmer joined Microsoft and eventually

became CEO", their system can identify the relation "CEO" between "Steve Ballmer" and "Microsoft", even if the word CEO does not appear between those two mentions.

In [22], Chan and Roth explored the fact that relations between entity mentions in the ACE 2004 corpus are categorized in five groups: premodifier, possessive, preposition, formulaic, and verbal. For example, in the expression "American President", an adjective or proper noun modifies the head word of the noun phrase; thus, it is categorized as a premodifier relation. In contrast, in the expression "President of the United States", there is a preposition that is attached to the head word, and thus it is categorized as a preposition relation. The authors found that if they were able to predict the possible relation category between entity mentions, e.g., premodifier, they would be able to discard those that were unlikely to have a valid relation. In essence, estimating the relation category, allowed Chan and Roth to build a subsequent classifier that receives a validated list of entity mentions which have a high probability to have a relation, reducing false positives, and increasing F-measure. To identify the relation categories, the authors created a list of patterns for the four first categories. The patterns use complex regular expressions based on words and POS tags to create pattern matchers that can be applied to sentence fragments.

All the previous supervised methods rely on a classifier that requires the right set of features in order to make a good prediction. Hence, most of the challenges for relation extraction within this family of methods involve getting the right amount and quality of labeled data, which usually demands hiring experts to annotate sentences. Moreover, the training data also needs to include negative examples, and provide a balanced distribution of examples across all relation classes [23]. In addition, these methods require careful and consistent analysis of the features added or removed from the model and their effects in the prediction accuracy, i.e., they demand considerable feature engineering. One of the major disadvantages of supervised methods is the requirement to explicitly define the relations of interest which in turn makes it hard to adapt the classifiers to recognize additional relations in the future.

2.1.2 Semi-Supervised Approaches

Semi-supervised approaches have the capability to extract relations without requiring large amounts of labeled data. Most of them rely on a set of rules or small set of examples that allows them to explore a database to obtain additional training, possible noisy, examples. Two of the most important approaches are DIPRE and Snowball, which are described below.

Brin followed a bootstrapping technique in [24] to extract book-author relations from the web just by providing a handful of valid book-author pairs. The author explored the fact that patterns and relations are linked. A good set of patterns can extract good examples of a target relation between entity mentions. Then, these good examples of a target relation could be analyzed to discover new patterns, at which point the entire process is repeated. In this way, the initial input to a system of this nature is only a small number of very good examples that showcase a relation that is stated in different ways in a vast corpus. In the case of [24] such corpus was a subset of the Internet composed of 24 million web pages. Brin built a system called DIPRE that took only five seed pairs that expressed the book-author relation, e.g., ("William Shakespeare", "The Comedy of Errors"). Then, the system analyzed the corpus and obtained occurrences of the seed pairs, including the web page URL and the surrounding text of the sentence that contained the occurrence. The system used a set of hand-crafted rules to extract patterns from these occurrences by exploring the similarity between the URLs and the similarity of the text that lied between the entity mentions. The patterns selected made a tradeoff: they would result in fewer false positives because of the patterns' specificity but would also result in lower coverage or recall. The author reasons that such compromise is acceptable for a corpus that is as vast and redundant as the web, in which the same relation is probably expressed repeatedly across different web pages. DIPRE was able to identify over 15,000 distinct book-author relations by using the method described above.

Agichtein and Gravano augmented Brin's approach in [25] by introducing a step that performs NER on sentences, and by generating patterns that included named entity types. Compared to DIPRE, the Snowball system developed by the authors, only

evaluated relations between entity mentions clearly defined. For example, the relation "headquarters" would only be evaluated between entities of types "organization" and "location". Like DIPRE, Snowball takes an initial set of seed tuples, e.g., ("Microsoft", "Redmond"), and then searches for occurrences of the tuples in a large corpus. It obtains the NER tags of the mentions and discards those that do not comply with the expected type. For example, if "Redmond" was found in a sentence but used in the context of a last name, it would be labeled as "person" instead of "location" and this sentence would be filtered out. Snowball analyzes the sequence of words between the entity mentions as well as the text left of the first mention and right of the second one. But in contrast to DIPRE, Snowball does not group occurrences just by selecting sentences with the same literal text in between their entity mentions. Instead, Snowball uses the Vector Space Model (VSM) to represent the text that provides context to the occurrence. In VSM, a text fragment is represented by a vector of numbers, usually calculated based on the frequency of the terms in the text fragment and in the entire corpus [1]. Modeling text as vectors enables the calculation of statistical similarity between two pieces of text using methods such as cosine similarity [1]. By incorporating VSM, Snowball gains robustness and is able to group occurrences in which their connecting text only has minor differences, such as punctuation symbols, but that hold the same valid relation, something that DIPRE is not capable of. Snowball uses a clustering algorithm to group the occurrences of seed tuples found in the corpus by matching the similarity of the vectors that describe the context or connecting text between the mentions, if such similarity is over a preset threshold. At this point, Snowball has effectively identified a new pattern for each cluster, where a pattern is defined by five elements: the two entity types that are common to all occurrences of the cluster, and the three vectors that represent the text left and right to the entity mentions and the text between them. The latter component of the pattern, the vectors, are taken from the respective cluster centroid calculated by the clustering algorithm as the average of the vectors of all items that form the cluster. Another contribution of Agichtein and Gravano is that they filter out patterns and tuples based on a set of rules and confidence score. For example, they use the new generated patterns to find new occurrences and discard the patterns that could not retrieve more than a set of minimum occurrences. In addition, they check if any of the new tuples extracted by the new patterns actually contradict previous findings. For example, if a pattern

extracts the tuple ("Microsoft", "New York") it would receive lower confidence score, given that from previous iterations, specifically from the seed tuple, Snowball knows that Microsoft headquarters are in Redmond. The use of a NER tagger, the abstraction of the context via VSM, and the process of estimating tuple and pattern confidence allows Snowball to achieve significantly higher recall and precision than DIPRE.

One problem with the previous two approaches is that both expect to find entity mentions that express the relation near each other. However, in natural language, it is common to use pronouns and other noun phrases to refer to subjects mentioned earlier in a piece of text. In those cases, the seed tokens and respective NER tags will be too far apart and might even belong to different sentences, preventing the systems from detecting and extracting the relation. In [26], Gabbard et al. leveraged coreference resolution to address this deficiency. The authors introduce a coreference step that replaces no-name references, like possessives, by names observed before, allowing the identification of candidate tuples that would have been missed otherwise. As expected, this approach allowed the authors to achieve higher recall that previous bootstrap-based approaches.

2.1.3 Unsupervised Approaches

The initial set of seed examples selected greatly influence the precision and recall achieved by semi-supervised methods, such as DIPRE and Snowball. However, no guidelines exist as to what the characteristics of good seed examples are or the optimal number of seed examples. In comparison, unsupervised approaches can extract relations without requiring an initial set of seed examples and without defining a set of relations of interest. Thus, unsupervised approaches are useful when the focus of the extraction is broad, i.e., there is not just a handful of relations of interest but thousands, and when it is not possible to know the most important relations expressed in the corpus a priori.

In [27], Hasegawa et al. propose a completely unsupervised relation extraction system. Their method first uses an ad-hoc NER tagger that can recognize hundreds of

entity types. The NER tagger process the sentences within a domain-specific corpus and identifies entity types and their locations. For example, in a corpus of financial transactions, it could find mentions such as "Google", "Nest", "Amazon" and "Zappos", and tag these as entities of the "company" type. For each possible pair of instances, it explores the corpus and tries to find co-occurrences, that is, text fragments in which both entities appear near each other, within a predefined maximum distance. As a result of this operation, the system obtains many sentences that relate both entities. For example, some sentences that the system might have obtained could be "Google announced the acquisition of Nest in an all-cash deal", "Google buys Nest for 3.4 billion dollars", "Amazon buys Zappos for 900 million dollars" and "Amazon closed the acquisition of Zappos". All of the words found between a pair of the same mentions, e.g., "Google" and "Nest", are accumulated. Using VSM, specifically TF-IDF vectors, the authors model the accumulated list of words to represent the context of a specific pair. Once all pairs have gone through this process, Hasegawa et al. cluster pairs that have the same entity types according to the cosine similarity between their accumulated connecting words. Given that there is no way to know in advance the number of clusters, a hierarchical clustering technique is used. For example, given the four expressions mentioned before, the system would have accumulated them in two different groups because of their entity mentions, and during the clustering process, both groups would have been clustered together because of the high cosine similarity between their aggregate connecting text. The resulting cluster would represent entities of type "company" in which the first one acquired the second one. The label for the cluster, that is the name of the relation identified, would be calculated based on the most common words among the connecting text of the sentences that belong to the clusters. The disadvantages of this method include the fact that some relations are expressed with text that does not appear between entity mentions causing the extractor to miss them, and that because many distinct relations can exist between the same pair of entity types, the extractor might cluster them incorrectly. For example, a pair of companies might have appeared in the context of an acquisition but later appeared in the context of one being a subsidiary of the other. The clustering method used by the authors could end up grouping both sentences believing they express the same relation.

Etzioni et al. described an unsupervised system, called KnowItAll, capable of ex-

tracting information from the web instead of from a particular corpus [28]. Because of the scale of the web, the authors reasoned that deep sentence syntactic analysis was not the right approach to solve the task at the web scale. Instead, KnowItAll starts with an ontology and a set of predefined generic templates from which extraction patterns are derived. KnowItAll is composed of three main modules: Extractor, Search Engine Interface, and Assessor. The Extractor uses the ontology to generate class specific extraction patterns from a set of templates. For example, a generic template could be "NP, such as NPList", where NP represents a noun phrase, and NPList a list of noun phrases. Hence, an extraction pattern to capture the hyponymy relation between the elements of a list and a specific class from the ontology, like "Country", can be derived as "countries, such as NPList". The Extractor will use these class-specific extraction patterns to find matches within sentences of a document passed as input. In addition, it processes the sentences to obtain their chunk and POS tags, and uses this information to apply syntactic constraints and filter out spurious extractions. For example, it validates that the head word of each of the noun phrases in the list are proper nouns. Lastly, the Extractor generates keyword queries for each particular extraction pattern, by using the literals involved in it. In the case of the extraction pattern mentioned before, the literals "countries" and "such as" become the keyword query "countries such as".

The second KnowItAll component, Search Engine Interface, takes the keyword queries generated by the Extractor and sends them to many search engines, including Google. In a parallel fashion, the Search Engine Interface, visits each of the web pages listed in the results returned by the search engines. It downloads the web page content, which is later presented to the Extractor component.

Finally, the Assessor computes a score about a candidate extraction obtained by the Extractor, e.g., ("country", "Italy"). The score is based on hit counts obtained from the search engines to discriminant phrase queries. KnowItAll generates discriminant phrases from the extraction patterns and a particular instance of a class. In this context, a discriminant phrase could be the keyword phrase and the instance appearing contiguously, e.g., "countries such as Italy". For other classes, such as "Scientist", a discriminant phrase that includes the class and the instance contiguously, e.g., "scientist

Albert Einstein", would be more appropriate. The abundance of particular discriminant phrases in search results signal to KnowItAll that the candidate extraction is valid. Discriminant queries related to valid extractions result in a higher search engine hit count, which in turn result in higher confidence score.

KnowItAll utilizes bootstrapping to generalize the extraction patterns in following iterations of the process. After it collects enough high-quality instances of classes using the extraction patterns derived from the generic templates, KnowItAll trains discriminators using the extractions with the highest confidence score. It then selects the best discriminators for each class and use the Search Engine Interface to obtain new instances of the classes, which are then filtered by confidence score, and the process repeats.

In a subsequent work [29], Etzioni et al. conducted a detailed analysis of KnowItAll. They proposed enhancements to allow the system to extend its coverage of the relations it can extract. One of those enhancements is the inclusion of a Pattern Learning module. The authors explain that a crucial deficiency of KnowItAll is that the generic templates need to be flexible to work across classes but that in turn prevents the system from learning domain-specific patterns. For example, a good extraction pattern to identify movies would be "the film NP starring" but this template would not generalize well to other classes, such as "album"; thus, it could not be incorporated in the initial set of generic templates. In essence, the authors recognize that the relations KnowItAll can extract are generic in nature, e.g., part of, hyponymy, etc. To solve this, the authors propose extending the use of the search engines to obtain context about instances of classes. The new version of KnowItAll takes instances of classes captured by the Extractor and validated by the Assessor, and uses them as keywords within queries sent to the search engines. The documents returned in response to these queries are examined and the context, words around the occurrence of the instance within a resulting document, are gathered as patterns. In these new patterns, the occurrence of the instance is replaced by a placeholder associated to the class name. Because of the way these patterns are generated, they are domain-specific and can increase coverage. For example, if "Point Break" is a validated instance of the class "Film", then a query for "Point Break" might return documents including the phrase "parts of the film Point

Break were shot at", from which the pattern "parts of the film <FILM>were shot at" could be derived. Furthermore, the authors came up with a method to estimate the expected precision and recall of the extractions these patterns could achieve, which allows the new version of KnowItAll to filter out most of the low-quality patterns.

A second enhancement presented by Etzioni et al. in [29] is the introduction of a component called List Extractor, which results in a significant increase in coverage over baseline KnowItAll. The researchers explain that on web pages it is common to find lists that have instances of a specific class because many web pages auto-generate content from databases, so they impose a structure to the information they expose online. The List Extractor takes some validated instances of a class, queries the search engines, and examines the documents returned by them. The component parses the HyperText Markup Language (HTML)[3] of the page looking for list structures. It then learns which class is the list about and extracts the list items. The number of lists in which a particular item appears is tracked too, and after validation, this information is used to incorporate new instances. For example, the List Extractor might prepare a query with the keywords "Point Break", "The Godfather", "Rocky" to get information about the class "Film", find many lists in which not only the previous films appear but also the terms "Casablanca" and "Gone with the Wind", allowing the system to discover new valid films. It is important to note that the authors also came up with a way to learn patterns that indicate the presence of a list of instances on a web page. They call these patterns wrappers. The wrappers can be assessed in terms of quality and even new ones can be discovered over time. Through these enhancements, Etzioni et al. improved recall by 4 times over the original version of KnowItAll when only extractions with a confidence score over 0.8 were considered.

In [30], Yan et al. proposed a similar clustering approach to Hasegawa's but applied it to a more structured corpus, specifically Wikipedia, which enabled them to identify stronger relations. The authors created a list of related concepts by exploring how Wikipedia pages were linked to others. The main title of a Wikipedia entry is considered a main concept or entity, while pages linked to it are considered secondary ones. Their method identifies sentences from Wikipedia in which pairs of concepts appear

[3] `https://www.w3.org/TR/html401/`

together. Dependency parse patterns of such sentences are mined to represent intrinsic context. To gather additional features, Yan et al.'s method also uses a Web Content Collector component that issues queries that include the concept pair to a search engine. Upon obtaining the search results, it analyzes the snipped text included in each entry to extract additional context that might not have been stated in Wikipedia. From this text, the system selects a keyword for each concept pair as a simplified representation of the context. Then, the system applies a two-phase clustering process to group concept pairs. First, the system clusters concept pairs by similarity of the context provided by the dependency parse trees that link the concepts. Once stable clusters are formed, they are considered as initial points for a second k-means clustering algorithm that uses the context retrieved for each concept pair from the search engine. As expected, the use of additional information not present in the corpus, i.e., Wikipedia, allows this method to improve recall, as certain relations are not directly captured by dependency trees, but are paraphrased and expressed in more direct ways on the web.

2.2 Open Information Extraction

Open information extraction has the same goal as information extraction but with additional characteristics: the sole input is a corpus composed of articles in natural language, the open information extraction system performs a single pass over the corpus to discover relations stated on the text, and the system can work at the scale of the web [31]. Thus, in open information extraction, the system does not receive a list of relations of interests, it does not receive a list of the relation arguments, nor a list of the relation arguments' types. Most open information extraction systems are self-supervised, that is they use heuristics to generate labeled data to train extractors. The goal of the relation extraction task was to identify the relation between entity mentions marked in a sentence, in case such relation actually exists. In comparison, the goal of the open information extraction task is to product a set of triples as a result of analyzing a sentence. All the entries of the triple are text fragments. The triple indicates that two entities in the triple, usually the first and third entry of the triple, have the relation specified in the remaining entry, usually the second one. One could

consider that, in a way, open information extraction performs entity mention detection and relation extraction at the same time. Open information extraction is an active area of research that has gained interest over traditional relation extraction among the research community in the last couple of years. This section describes four approaches that are often cited in the literature: TextRunner, O-CRF, WOE and ReVerb.

2.2.1 TextRunner

In [31], Banko et al. reached a similar conclusion as Etzioni et al. in [28]; that is that methods that rely on NER taggers to perform relation extraction will not work with the vast, heterogeneous and multi-domain text that one finds on the web. Thus, to build a system that can capture knowledge at that scale, one cannot depend on deep linguistic features for the extractor. Furthermore, the authors recognize that even unsupervised methods, e.g., KnowItAll, within the traditional information extraction space still rely on entity or pattern seeds that allow them to bootstrap the extraction process. In comparison, Banko et al. envision a system that is domain-independent and that does not require prior information about relations. Banko et al. propose TextRunner, a system made of three components: a self-supervised learner, a single-pass extractor, and a redundancy-based assessor. The self-supervised learner takes the Penn Treebank [32], a small corpus composed of approximately 2,500 articles from the Wall Street Journal that have been manually annotated and generates a classifier that estimates if a candidate extraction is likely to be correct or not. The learner processes the sentences from the corpus and obtains its dependency parse tree. Using that tree, the learner finds the base noun phrases within the sentence, which it considers possible entities. Then, for each pair of base noun phrases, the learner finds the words that connect such phrases by navigating the dependency tree. This set of connecting words become the relation and along with the base noun phrases, form a candidate extraction. The learner evaluates a list of conditions on the extractions; if all of them are met, the extraction is considered valid; case contrary, the extraction is marked as invalid. Some of the conditions evaluated include that the dependency path between candidate entities is shorter than a predefined length, that the dependency path from entity 1 to entity 2 does not cross a sentence boundary, and that the entities are not just pronouns.

Once all extractions have been labeled as valid or invalid, the learner transforms the extractions into features that are used later to train a Naive Bayes classifier, which is the final output of the learner. The set of features derived from the extraction include the POS tags of the relation, POS tags to the left and right of the entities, the number of tokens in the relation, whether the entities are proper nouns, among others.

The second component of TextRunner, the extractor, goes through a large web corpus, and processes each sentence, obtaining its POS tags and its chunk tags to identify noun phrases and possible entities. The extractor also examines the text between the noun phrases and follows a set of heuristics to simplify the connecting text and obtain the relation. Some of the heuristics used by the extractor include removing prepositional phrases and single-token words such as adverbs. As a result of this process, the extractor obtains candidate extractions that the classifier evaluates. Those that are classified as positive are persisted by TextRunner.

The last component of the TextRunner system, the assessor, normalizes the relations by removing verb and noun modifiers. Then, it groups all positive extractions that have the same entities and the same normalized relations, and count the number of sentences in which such base extraction was present. These counts are used to estimate the probability that the extractions contain a valid relation between the entities.

When TextRunner and KnowItAll were given a corpus of 9 million web pages, both produced a similar number of extractions about relations of interest given to KnowItAll. However, the error rate of the extractions produced by TextRunner was 30% less than the error rate of those produced by KnowItAll. From a software engineering perspective, TextRunner is also more efficient than KnowItAll; the former can capture almost the same number of extractions than the latter, but with a total runtime in the range of tens of hours, instead of tens of days over the corpus described before. Factors that enable TextRunner's efficiency include its capability to execute a distributed extraction process, and its ability to extract relations in a single pass. Moreover, in the same time that it took KnowItAll to capture extractions for given relations of interest, TextRunner is able to capture orders of magnitude more extractions about broader relations. Furthermore, TextRunner does not depend on search engines, which tend to

impose querying rate limits that could hamper the performance of an open information extraction system.

2.2.2 O-CRF

Building on the lessons learned from TextRunner, Banko and Etzioni introduced the O-CRF system in [33]. O-CRF incorporates a couple of changes to TextRunner that allows it to extract triples from binary relations at high precision, almost double the recall of, and 63% F-measure gain versus TextRunner. The authors propose reframing the relation extraction problem from a classification one to a sequence labeling one by incorporating a conditional random field model instead of a Naive Bayes classifier. Conditional random fields could be thought of as a model that when given a sequence of inputs, outputs the most likely sequence of labels for such input. Similar to TextRunner, O-CRF also has a self-supervised learner. O-CRF uses a set of relation-independent rules to extract triples from a small corpus. Some of the rules allow the extraction of valid triples, which act as positive training data, and other rules purposely extract invalid triples that serve as negative training examples. Features similar to those used in TextRunner, e.g., chunk and POS tags, etc., are extracted from these training examples, and are used to train a conditional random fields model. As a result, the model learns to identify words that explicitly state a relation between entity mentions. Once the model is prepared, O-CRF can start processing a large web corpus. First, it extracts sentences, chunks them, and identifies noun phrases. If there is a text fragment with two noun phrases connected by a number of words smaller than a predefined length, the text fragment is considered a candidate extraction pending evaluation. O-CRF then anchors the noun phrases as the beginning and end of the input sequence, and the conditional random field model trained before is used to generate a sequence of labels. If the model believes there is a relation in the words connecting both noun phrases, then the beginning and end of such relation will be marked with special labels in the output sequence. A particular characteristic of O-CRF is that it has a second mode of operation, similar to traditional relation extraction because it can receive relations of interest and only focus on extracting those if necessary. To be able to work in this mode effectively, when the output sequence provided by the conditional

random fields model has labels marking a potential relation, O-CRF passes the labeled relation to a Synonym Resolution component based on [34]. The Synonym Resolution component, called Resolver, is able to find similar text fragments that express the same relation, which allows O-CRF to increase its recall in this second mode of operation. This is a particular capability that O-CRF possesses but TextRunner does not.

2.2.3 WOE

Wu and Weld proposed WOE, an open information extraction system than uses Wikipedia instead of the Penn Treebank to build a dataset that is used to train extractors in a self-supervised fashion [35]. WOE is made of three components: preprocessor, matcher, and learner. The preprocessor takes Wikipedia articles' HTML and obtains its POS and chunk tags. The preprocessor also creates lists of synonyms for the main concepts of the article, and if possible, for the values listed in the Wikipedia article's infobox. A Wikipedia infobox is a kind of template, similar to an attribute-value table, that certain type of Wikipedia pages include on the right side of their content; see Figure 2.1 for an example. To create the lists, the component employs two techniques: redirect page inspection and backlink inspection. With the former, the preprocessor examines redirection pages that take users to the article under examination, and extracts the title of the redirect pages, which become synonyms. With the latter, the preprocessor explores Wikipedia pages that have links to the article under processing, finds the links, and extracts the anchor text used by such pages. For example, using the first technique, the preprocessor finds that the Wikipedia page "Einstein" redirects to "Albert Einstein". The second technique would have found the same synonym if the preprocessor examines the page "University of Zurich".

The second WOE component, the matcher, takes the lists of synonyms and the tuples of attributes and values captured from the article's infobox, and searches for sentences that contain a reference to both: the article's main subject and a value from one of the tuples. To extend coverage, the matcher uses the lists of synonyms to find additional sentences. The matcher also employs a set of heuristics to broaden the scope of sentences it associates to a particular tuple. For example, if no sentence is retrieved

Figure 2.1: Example of an infobox

for an exact match or a synonym match for the article's subject then the matcher tries to find sentences that match a prefix or suffix of the subject. Additional heuristics, such as assuming that the most common pronoun refers to the subject, are applied in cascading fashion if necessary. The matcher also follows a set of rules to discard sentences that might not be of good quality, e.g., verifying that the subject and value are headwords of noun phrases in the sentence.

The WOE system actually comes with two different learning extractors, WOEpos and WOEparse. WOEpos trains a conditional random field model with the sentences annotated by the preprocessor and selected by the matcher, similar to O-CRF. In contrast, WOEparse extracts the dependency parse tree of the candidate extraction, and uses features based on the dependency path to construct a pattern learner. Such pattern learner will eventually be able to examine the dependency path between two noun phrases and decide if a relation is stated in it. It is worth noting that WOEparse applies a series of transformations to the dependency parse trees before using their

features to train the pattern learner. For example, WOEparse discards or adds tokens to the shortest dependency path depending on their function in the sentence. It also replaces the lexical tokens with their corresponding POS tags, and converges families of POS tags into single label, such as mapping POS tags for proper noun, proper plural noun, singular noun, and plural noun to just noun. The result of these transformations is called a generalized core path by the authors and becomes a pattern. After all of this additional processing, WOEparse persists the patterns to a database and calculates their frequency over the given set of labeled sentences. According to Wu and Weld, from a set of more than 250,000 labeled sentences, WOEparse learned more than 15,000 different patterns, with 185 of them appearing in 100 different sentences each.

WOEpos achieves an improvement of more than 18% of F-measure over TextRunner, while WOEparse's gain is greater than 72% over the same baseline. To achieve such increase in F-measure, WOEparse sacrifices runtime: it takes WOEparse almost 30 times as much time to process a sentence when compared to TextRunner and WOEpos. An interesting observation that Wu and Weld make is that even if WOE was trained on a specific kind of corpus, Wikipedia, it performs well on other kind of sources too, such as news articles. That is because the WOE system uses non-lexical features, such as POS, chunk tags, and the generalized core paths, instead of explicit tokens, i.e., words, which is also true for TextRunner. An additional interesting finding of the authors, is that as the length of a sentence increases beyond 10 tokens, the F-measure of WOEparse decreases at a smaller rate than that of WOEpos, and it never goes below 0.5 even for sentences made of more than 40 words. Nevertheless, that comes at the cost of runtime: WOEparse processing time grows quadratically with respect to sentence length, while WOEpos runtime is linear.

2.2.4 ReVerb

Fader et al. introduced a rule-based approach to open information extraction in [36]. Previous methods used heuristics or distant supervision to gather a dataset of annotated sentences that contain relations between entities, and used such sentences to learn

an extractor. During the extraction phase, the extractor was utilized to evaluate the context connecting two entities, usually noun phrases, and tag the tokens that formed the relation, in case the extractor believed one existed there. One problem with the previous approaches is that the resulting datasets are noisy, and they tend to provide a large number of example sentences for a small set of relations. Furthermore, because these methods frame the problem as a sequence-labeling one, their models first anchor candidate arguments of the relation, typically noun phrases, then try to extract a relation. This could derive in the extractor considering a noun phrase that is part of the relation as an argument, resulting in uninformative extractions. Lastly, because the extractors make decisions on what tokens are part of the relation, word by word, when early decisions are wrong, the identified relation tends to be incorrect.

To combat these problems, Fader et al. proposed ReVerb, a system that can identify most of the English verb binary relations. Instead of generating a training dataset to learn an extractor, ReVerb uses simple manually crafted rules to extract triples. Moreover, compared to previous open information extraction systems, ReVerb first identifies possible relations in a holistic manner, and only then attempts to find the correct arguments for those relations.

ReVerb identifies candidate relations and arguments by examining POS and chunk tags. For ReVerb, a sequence of tokens that starts with a verb is a potential candidate relation. ReVerb applies syntactic rules to filter out uninformative or incoherent extractions that other systems such as TextRunner often surface. Uninformative extractions are those that are missing information, such as noun phrases that offer more context to the relation or to the arguments. For example, from the sentence "Thriller is an album by Michael Jackson", previous extractors would have captured ("Thriller", "is", "an album") as a triple, while ReVerb would capture ("Thriller", "is an album by", "Michael Jackson"). Incoherent extractions are those for which the extractor captured a relation that does not make sense, usually because the relation was created from words distant from each other in the sentence. For example, from the sentence "They recalled that Thriller was recorded in Los Angeles", previous extractors would label the relation between "Thriller" and "Los Angeles" as "recalled recorded" instead of "was recorded in". Additional syntactic rules that allow ReVerb to minimize this

kind of errors include verifying that a multi-word relation ends with a preposition, or that the tokens of the candidate relation are all contiguous.

Because of the syntactic rules, ReVerb could end up extracting over-specific relations, with many modifiers such as adverbs, that might not drastically change the meaning of the relation. To overcome this, ReVerb relies on a dictionary of relations constructed before the extraction phase. The dictionary is populated by processing hundreds of millions of sentences from the web using the syntactic rules described before. The arguments of those relations are identified, and an entry for each distinct relation is added to the dictionary if the relation has been observed for a minimum number of distinct arguments. Fader et al. report that the ReVerb dictionary holds more than 1.7 million different lexical relations. During the extraction phase, once a relation has passed the syntactic rule validation, ReVerb consults the dictionary to check if the candidate relation has been significantly observed before on the web. If that is not the case, the relation is discarded.

Once a relation has been identified correctly, ReVerb extracts its arguments by finding the nearest noun phrases to the left and right of the relation and applying additional syntactic constraints, such as that those noun phrases are not just pronouns. At the end of this process, ReVerb returns a triple.

ReVerb also trains a logistic regression classifier to estimate the confidence score of the triple extracted. The dataset required to train the classifier is relatively small, in the order of 1,000 sentences in total for positive and negative examples. Features used by the classifier include the length of the sentence, the type of the last preposition present in the relation, the argument's type of noun, among others. When taking confidence score into consideration, ReVerb achieves 30% more area under the curve than WOEparse and almost twice area than TextRunner and WOEpos. In terms of runtime, ReVerb is 20% faster in processing sentences than TextRunner and WOEpos, and almost 50 times faster than WOEparse.

2.3 Knowledge Base Construction

The knowledge base construction task uses methods of pattern matching, information extraction, open information extraction and data integration to automatically build knowledge bases. A related task called knowledge base population has the objective of extending and enhancing the coverage of an existing knowledge base. During the last couple of years, researchers have proposed systems that perform knowledge base construction from structured data such as database tables and semi-structured data such as Wikipedia infoboxes. More recently, in an effort to extract knowledge about events, systems that can process unstructured data, specifically news articles, have been proposed. This section presents six systems that are some of the most cited by papers on the knowledge graph construction literature. The systems described in detail below are: Yago, DBpedia, EVIN, XLike, ECKG and EventKG.

2.3.1 Yago

Instead of using an open information extraction approach to automatically build an encyclopedia-like knowledge base, in [37], Suchanek et al. present the idea of using structured data from Wikipedia and WordNet [38], a lexical database. The authors argue that for applications that require near-perfect knowledge, such as those that reason over assertions, systems based on open information extraction techniques, e.g., KnowItAll, generate many inaccurate extractions, and when only high confidence extractions are considered, recall is low. In order to avoid this problem, they propose Yago, a lightweight system that can identify a predefined set of 14 relations, producing more than 1 million entities and 5 million facts. Yago uses a set of heuristics to process Wikipedia pages and categories, and WordNet synsets and their hierarchical structure to produce extractions. A synset is a group of words that could be considered semantically equivalent or refer to the same concept, in addition that they belong to the same POS category, like the noun words “human” and “person”. In particular Yago leverages the hypernym/hyponym relation that exists between synsets. Their approach allows the Yago system to capture millions of assertions with over 95% accuracy.

Yago obtains information about entity instances, such as people, or cities, from

Wikipedia pages, most of which have been categorized by editors. For example, the Wikipedia page "The Godfather" has been tagged as part of the "1972 films" and "Films directed by Francis Ford Coppola" categories, among others. On Wikipedia there are pages explicitly created to list all articles that belong to a category. This category information allows Yago to identify entities, concepts, and relations. The Wikipedia category hierarchy and the link structure connecting different categories and pages is extracted by Yago from a relational database provided by Wikipedia. On the other hand, to create an ontology of concepts, Yago uses WordNet instead of Wikipedia, because the former is cleaner in its organizational structure than the latter. For example, on WordNet, a number of synsets are associated to the word "film", including the synset "film.n.01" and "movie.n.01". When one obtains the hypernyms of the synset "movie.n.01", the parent synsets include "product.n.02" and "show.n.03". In contrast, the parent categories listed on the Wikipedia page "Film" include "Art" and "French inventions", which are not useful for organizing concepts into a taxonomy.

The list of relations that Yago recognizes include the generic "type", "subClassOf" and "means" relations among others. Assertions for each relation are extracted using a particular set of rules. For example, for the "type" relation, Yago assumes that each Wikipedia page is a possibly a unique entity that needs to be assigned a class. Such assignment occurs through the "type" relation. To determine which is the right class for a candidate entity, Yago examines the categories it has been tagged with. Suchanek et al. observe that Wikipedia categories sometimes serve different purposes: some categories represent concepts, such as the category "Film", while others serve more like a way to group pages that have something in common, such as the category "1972 films". The concept categories associated to a candidate entity are identified by conducting shallow NLP analysis on the category name. For example, categories that have a plural word as head word are considered concept categories by Yago. By leveraging the database of categories and page links, Wikipedia can learn all the concept categories, and the candidate entities that are instances of such categories. Hence, at this point, Yago can explicitly establish the "type" relation with an assertion.

To establish the "subClassOf" relation, Yago first turns WordNet synsets into classes. Then, it reduces the set of Wikipedia categories by only considering those

that are leaf categories, i.e., those that do not have subcategories listed in Wikipedia. Subsequently, Yago attempts to establish a link between the leaf Wikipedia categories and the WordNet high level synsets. For example, Yago would try to create an assertion that indicates that the "1972 films" Wikipedia category is a subclass of the class "Film" derived from the WordNet synset "film.n.01". To do this matching, Yago stems the head word of the Wikipedia category and extracts its premodifier. Then it performs a lookup for WordNet derived classes that have the concatenation of the premodifier and the head word, and establishes a match if it finds one. If not, it performs a second lookup for WordNet derived classes that only match the head word and selects the top one in terms of its frequency as observed by WordNet. To extract other kind of relations, Yago applies other set of heuristics and rules to additional pieces of information such as redirect pages, links and backlinks, and category names starting with special phrases.

The authors also introduce a data model to express binary relations between entities, properties of relations, e.g., transitivity, and event relations between relations. Furthermore, their model requires that entities be grouped in classes and that each entity has to be an instance of a class. An assertion composed of two entities and a relation is called a fact, and is given a fact identifier within the model. Other assertions can provide additional information by introducing relations to such fact identifier, e.g., provenance information, extraction method used, or confidence level.

2.3.2 DBpedia

In [14], Auer et al. present DBpedia, a system that can create a knowledge graph by processing structured data from Wikipedia. In comparison to Yago, their method is not limited by the relations of interests one can think in advance, nor it requires the time-consuming and error-prone task of manually designing pattern-matching rules to extract such relations. Instead, DBpedia can relatively easily identify relations by inspecting Wikipedia templates, particularly infoboxes. For example, Wikipedia pages about cities in a specific country may use a city template that lists its attributes such as population or geographic coordinates. Wikipedia pages, including infoboxes, are

created by Wikipedia editors through MediaWiki[4], the software that allows them to write Wikipedia articles and runs the Wikipedia system. Templates are delimited with special characters and the MediaWiki software imposes a structure to them. It uses a set of keywords to express meta information that is utilized by the MediaWiki system when rendering them. According to the authors, between 25% and 33% of articles contain information-rich templates.

Wikipedia periodically publishes data dump files with relational database tables that contain articles and their texts. DBpedia uses this data as input and follows two different methods to extract information and build a knowledge base. The first method is to simply map relations already stated via the relational databases to an RDF triple form, possibly leveraging the column labels as predicates, table names as potential classes, keys as identifiers or additional relations, and table rows as entities or instances. The second method extracts information from templates, and hence it is much more sophisticated that the former. Auer and Lehmann describe the second method in detail in [39].

To extract information from templates, Auer and Lehmann proposed a five-step process. First, their algorithm issues a SQL query to the articles table to search for Wikipedia pages that have templates, which can be identified within articles' texts by the "{{" and "}}" delimiters generated by MediaWiki. Second, it extracts all templates used within the articles identified in the step before. Templates have types, names, attributes and values, with the last two sometimes linked to other Wikipedia pages. For example, the Wikipedia page "Los Angeles" uses an infobox template of type "settlement" and includes attributes such as "State" and "County". The author's algorithm filters out templates that have few attributes, and templates that are from a type that has been used rarely. Through these heuristics, the authors attempt to minimize incorporating errors made by editors and ingesting potentially contradicting information. Third, the algorithm parses each of the templates to create RDF triples. A URL based on the title of the page in which the template was found is created and is used as subject for all the triples derived from the template. The algorithm generates a different triple for each template or infobox attribute-value pair, with the attribute act-

[4] `https://www.mediawiki.org/wiki/MediaWiki`

ing as predicate and the value as object. Fourth, attribute values are further examined: if an attribute value links to a Wikipedia article, the triple object is replaced by the appropriate URL reference. This is easy to detect because within the template, tokens that represent entities described by Wikipedia articles are placed between the "[[" and "]]" delimiters. If the attribute value is a string, the algorithm tries to determine its type so that it can be appropriately represented by an RDF literal. Some attribute values contain lists instead of a single value. The algorithm can process such attributes and turn them into individual triples. The last step consists in assigning a class to the subject of the RDF triple, based on the article's category, and the template's type. DBpedia uses classes derived from ontologies defined by other systems such as Yago. The end result of this process is a knowledge graph in RDF representation containing over 100 million triples with over 2 million entities that have an article in Wikipedia.

Some of the limitations of DBpedia include that information not present in infoboxes is not uplifted at all, such as tables. Editors have guidelines on how to write articles but not any on when to use an infobox instead of a table to present information. Such decision is left at the discretion of contributors. Furthermore, editors are free to create additional templates and template types that might be redundant. Finally, some editors use infoboxes for administrative or formatting purposes instead of using them for summarizing key information. Such templates might be erroneously captured by DBpedia if its heuristics and extraction patterns are not revisited periodically.

2.3.3 EVIN

Previous general-purpose knowledge graphs like DBpedia or Yago hold limited information about events because of the curated source they use as input, e.g., Wikipedia, and because they focus on entities and facts. In [40] and further detailed in [41], Kuzey et al. propose EVIN, a system that takes a stream of news articles, extracts named events from it, and populates a knowledge base with such information, increasing coverage of events compared to general purpose knowledge bases. The authors point out that in order to build a high-quality knowledge base, the events need to be consolidated to avoid having assertions that describe the same event. Similarly, the authors emphasize

that named events need to be categorized with as much granularity as possible using a predefined hierarchy. For example, the "2016 U.S. Presidential Elections" event should be categorized as one of type "elections", while the "2018 World Cup Finals" event should be categorized as one of type "sports tournament". From the authors' perspective, an event is "something that happened at a certain point in time or during a certain time period".

EVIN takes news articles through a standard NLP pipeline to derive features from them, including the textual content, the article's publication date, named entities identified in the article, and the types of events mentioned in the article. Named entities are identified by a NER module in the pipeline but identifying the type of events mentioned in the article is more complicated. EVIN relies on Wikipedia categories to generate event types. EVIN takes all Wikipedia pages that belong to a Wikipedia category and constructs a language model based on the title, body content, and normalized dates that appear in the pages. Similarly, EVIN builds a language model for each news article. Then, it identifies the article's category by comparing its language model to the categories' language models using the Kullback-Leibler divergence metric. The Kullback-Leibler distance is a kind of measure that quantifies how similar two documents are. The categories that are the most similar to the article are accepted. EVIN further analyzes the accepted categories, by leveraging WordNet. EVIN searches corresponding WordNet event types using the head word of the accepted Wikipedia category as lookup term. From the WordNet event types recovered across all accepted Wikipedia categories associated to a single article, EVIN selects the event type deepest in the WordNet hierarchy. This process allows EVIN to classify articles into event categories of high granularity. For example, if EVIN ends up with the WordNet event types "contest" and "match" as results of a search to categorize an article about the "2018 FIFA World Cup Final Match", it would select "match" as the definitive event type, because "match" is a hyponym of "contest" in WordNet.

EVIN also calculates other measurements of similarity between news articles, such as content distance, temporal distance, entity distance, and type distance. The content distance is the cosine of the article's TF-IDF vectors based on a bag of words model. The temporal distance is the difference in the articles' creation date normalized over

the time horizon of the corpus, that is the difference between the oldest and newest article. The entity distance uses the Jaccard coefficient to quantity the overlap that exists between entities recognized in both articles taking the entities' TF-IDF values as input. The type distance also uses the Jaccard coefficient but quantifies the overlap on the articles' accepted categories by taking the IDF values of the set of accepted categories as input. All of these metrics allow EVIN to represent the corpus of articles in a graph. In this graph, each vertex represents an article's set of features, such as its creation date, the named entities recognized, and its accepted categories. If two vertices share at least one named entity and accepted category, they are connected by an undirected edge with a weight equivalent to the content distance. If two articles occur one after another according to their creation dates, the vertices that represents the articles are connected by a directed edge with a weight equal to the temporal distance. EVIN assumes that news articles will mostly describe a single event, given that the system represents articles in the graph and not fine-grained event expressions obtained via NLP from articles' sentences.

Once the graph is formed, EVIN goes through a process that reduces the graph complexity while preserving its properties. Specifically, EVIN follows an algorithm that allows it to estimate how much information would be lost if two vertices are combined into a single one. Through this process, EVIN is able to merge the representation of articles that are mostly about the same events. In theory, such vertices should share similar neighbors, have overlapping named entities, and categories. This process uses the distance metrics calculated before and through multiple iterations produces new versions of the graph with fewer vertices and with updated weights for the edges to their neighbors based on the vertices merged on the previous iteration. Once the graph becomes stable, and does not change anymore, EVIN applies a series of rules to each vertex and selects those that are high quality enough to be ingested into a knowledge base.

From observations of EVIN's UI, it appears EVIN is able to create a timeline of events, possibly leveraging the articles' creation-time based feature. However, it seems that the dates stated at the sentence level where only used for the generation of language models and did not play a role in the vertex representation of the article. Thus,

date literals are not stated in the timeline generated by EVIN, only order of events. This is one of the major deficiencies of the broad-event detection and article-level clustering approach followed by EVIN.

2.3.4 XLike

With the same objective of building a knowledge base from information extracted from news articles as Kuzey et al., but with an approach focused on extracting event information at a high level of granularity, Padró et al. propose the XLike system in [15]. A particularity of XLike is that it can process articles in many languages, including English and Chinese, and can consolidate event information into a semantic representation that is language independent. The obvious benefit of this approach is that events are reported in the news at different levels of detail depending on their relevance to the news publication and its country of origin. Thus, a system that can not only perform information extraction in many languages but also aggregates such knowledge, would significantly increase its coverage of events.

XLike implements an advanced NLP pipeline to process news articles. The NLP modules perform tasks such as NER, dependency parsing, word sense disambiguation (WSD), semantic role labeling (SRL) and frame extraction. The last three tasks leverage multilingual lexicons such as WordNet and support XLike's capability of representing knowledge in a language-agnostic fashion.

In natural language, words can have different meanings depending on the context in which they are used. For example, consider the sentences: "He saved the money he earned while working at the bank" and "One of the best ways to fish bass is to pound the bank". In both sentences the word "bank" is used but in the former, it refers to a financial institution, while in the latter, it refers to the terrain alongside the bed of a river. WSD is the task of determining which of multiple possible meanings of a word, i.e., sense, is being used in a particular sentence. WordNet and its extended versions that cover other languages besides English, group words in sets of synonyms called synsets. By using this and similar lexical resources such as the PredicateMatrix [42],

the XLike WSD module can map words that mean similar things in different languages and normalize them by referencing the same synset. For example, it can recognize that the word "bank" in English and the word "orilla" in Spanish, are mapped in the lexicons to the same synset and thus have the same meaning.

After a news article has been processed through the pipeline and syntactic information has been extracted, XLike performs SRL and frame extraction. A frame is a structure that defines elements that play specific semantic roles in particular frames and includes a list of predicates that use these roles. Lexical databases such as FrameNet [43] organize common frames, i.e., actions and events, in families and specify the typical roles involved in such frames, such as "actor" or "recipient". For example, when the action of buying a good is stated in natural language it usually involves concepts such as "buyer" and "seller", while the action of making or releasing a film would involve concepts such as "artist", "production company" or "studio". From the point of view of Padró et al., a frame corresponds to a predicate, and the participants in the frame correspond to the predicate arguments. For example, from the sentence "Google is based in Mountain View, California", XLike would detect the frame "base", which states that "someone or something is established somewhere". For this particular sentence and frame, "Google" is a participant that plays the "theme" role while "Mountain View, California" plays the "location" role. Through the use of WSD and frames, XLike is able to construct a semantic representation of a sentence in a graph form. XLike takes all graphs constructed from all sentences in an article and consolidates them into a single one for the entire document. To do this, XLike follows a simple coreference resolution approach by connecting individual graphs that have frame participants in common, such as repeated nouns and named entities labeled by the NER module. The semantic representation could go through an RDF transformation process that involves entity resolution, entity linking, and lastly persistence to a knowledge base.

2.3.5 ECKG

In [44] and more thoroughly described in [45], Rospocher et al. propose a system that can automatically construct an event centric knowledge graph, called ECKG, by

extracting information from a corpus of news articles. The authors define events as "things that happen" and that consist of four different pieces of information: an action, a time at which the action occurred, a location in which the action occurred, and the participants involved in the action. Within the research community [46], ECKG is currently considered the state-of-the-art in terms of systems that automatically build knowledge graphs from news articles.

The system Rospocher et al. propose has two main parts: one that processes documents and annotates their content with syntactic and semantic information, and another one that performs coreference resolution across documents. The first part is governed by a long and complex pipeline of modules that implement deep NLP techniques. The pipeline is capable of identifying events, even those that do not have a particular name, actors involved in the event, as well as the possible event location and time. The output of the pipeline is a representation of all the event mentions. It is important to point out that the system actually has four different pipelines, one for each of the language it supports, and that the NLP modules of each pipeline are language-specific. For example, the English pipeline consists of 15 modules while the Spanish one only has 11. The information that flows from module to module is represented using the NLP Annotation Framework (NAF) [47], which is a format to describe the results of different levels of linguistic analysis conducted over a text. The output of each module results in annotations added to the NAF representation of the news article.

Standard NLP operations are applied to the text by the first few modules of the pipeline. For example, the POS tags and lemmas are obtained, named entities identified, and constituency and dependency parse trees generated. This vital information is used by the pipeline's Nominal Coreference module to find out mentions of the same entities within an article. The WSD module takes lemmas and POS tags, and maps tokens to WordNet synsets. These synsets are eventually used to identify sentences that basically have the same predicate but in different languages.

In order to create a knowledge graph in which resources are linked to other knowledge bases, the Named Entity Disambiguation (NED) module takes named entities,

and queries a local service based on DBpedia Spotlight[5] REST server. DBpedia Spotlight is a service that provides an HTTP Application Programming Interface (API) through which clients can send the text of an entity and its NER type, and receive in response the full name of a resource with a similar name and type in DBpedia.

A critical module of this pipeline is the SRL one. The module takes tokens, lemmas, tags and the sentence's dependency parse tree, and associates specific semantic roles. The SRL task encompasses relation extraction including argument identification. One can think of SRL as the process of determining "who did what when and where". For example, for the sentence, "Nest sold to Google for 3 billion dollars", the action taking place is "selling", the role that "Google" plays is "buyer", and the role that "Nest" plays is both "goods" and "seller". In order to do this, the SRL module leverages many lexicons and corpus such as PropBank [48], VerbNet [49], and FrameNet. It is worth noting that the same action could be stated from different perspectives. For example, the sentence "Nest was acquired by Google for 3.4 billion dollars" has the exact same meaning as the one stated before. However, the SRL module would end up associating the "buying" frame to the latter instead of the "selling" frame. To reconcile different perspectives, additional processing is needed which is out of scope of SRL.

In order to avoid repeating the same event within the knowledge graph, ECKG incorporates an Event Coreference module. This module takes the SRL annotated predicates of an article and matches them semantically by examining predicate lemmas, WordNet similarity score, and potential event components in common. Hence, ECKG could reconcile the sentence "Google buys Nest for 3 billion dollars" to the sentence "Google's acquisition of Nest is expected to close in June next year", and merge the dispersed pieces of information, such as the transaction size and expected close date, into a single event representation. Unfortunately, the Event Coreference module cannot reconcile event mentions when they are expressed from different perspectives, as in the case of the sentence "Nest was acquired by Google for 3.4 billion dollars".

ECKG's pipeline also includes a Time and Date Recognizer module. This module can identify temporal expressions when presented with tokens annotated with their

[5] `https://www.dbpedia-spotlight.org/`

lemmas, POS tags, NER tags, and the associated constituency parse tree. The module categorizes the expressions in classes and follows a series of rules to normalize those expressions to a formal time representation. A subsequent module called the Temporal Relation Extractor takes all the events annotated by the Event Coreference module and all the time expressions identified by the Time and Date Recognizer module and identifies if there is a relation between an event and a time expression, or between two events, or between two time expressions. It also detects relations between the article's time and the events detected from the article's sentences.

Other interesting and unique modules used by ECKG in its pipeline include the Opinion Miner module, which can recognize what is the opinion that an entity is expressing about another one; the Causal Relation module, which can detect if an event was triggered by another one by detecting lexical clues such as "as a result"; and the Factuality module, which is able to assess from a sentence's lemmas and POS tags whether what was expressed was fact or not. The Factuality module is the last one in the pipeline, and after it finishes execution, the NAF representation collaborative generated by all modules is ready to be used by the second phase of the process.

The second phase of ECKG's process reconciles events described across different articles. This often occurs in the media, since as time progresses, publishers report new information about recent events. For example, an article that describes a crime initially provides information about the location and time of the crime, but subsequent articles could surface information about the perpetrator and victims. In order to consolidate the information about an event that is dispersed over many articles, the system performs cross-document coreference. To do so, ECKG first needs a representation of the aggregate components of events. This representation was created in the first phase by the Event Coreference module when it merged information from different sentences of the same article that were about the same event. For example, in a news article about a crime, the time and place could be stated in a sentence, but the victim could be stated in another. The Event Coreference module generates a single object, called Composite Event Object, that has all of the information of an event at a per article level. ECKG saves these objects to disk in different folders according to the event time, e.g., one folder per calendar day. To resolve coreference across documents,

Composite Event Objects in the same folder are merged in the second phase if they meet 3 conditions: the action of the objects has the same lemma or belong to the same WordNet reference relation hierarchy, the objects at least share one participant, and if both objects have a location component they match too. This process of merging objects keeps repeating until no more object could be merged within a folder. At this point, ECKG follows a series of rules to transform the resulting Composite Event Object into an RDF representation. ECKG gives a proper name to the relation and triples that explicitly state the category of the event using information obtained from the SRL module. References to the participants and location are added as triples that link the event to the formal resource names of these entities in DBpedia, if such information was obtained by the NED module during the first phase. Lastly, the time of the event is stored as a reference to a resource that represents the particular day in time in which the event occurred.

2.3.6 EventKG

In [46], Gottschalk and Demidova proposed EventKG, a system that automatically aggregates event information into a knowledge graph by using other knowledge bases such as DBpedia as input, and by processing semi-structured data from Wikipedia. Gottschalk's and Demidova's motivation arise from the fact that general purpose knowledge bases, such as Yago, have a large number of entities and facts, but little event information, or the event information is incomplete, e.g., missing date or location component. The authors point out that state-of-the-art systems that can extract event information from news articles, such as ECKG, are not capable of achieving accuracy higher than 55%. They hypothesize that for the purpose of creating a large high-quality repository of events, consolidating events captured in distinct knowledge bases already built from structured data as well as processing additional semi-structured data offers better results. In that respect, the authors are successful given that EventKG consolidates over 600,000 events, far more than any other knowledge based, and its extraction accuracy is approximately 90%.

EventKG's inputs consist of three public and massive knowledge bases, Wikidata

[50], DBpedia and Yago; and two semi-structured data sources, Wikipedia Current Events Portal (WCEP) and Wikipedia Event Lists. From Wikidata, EventKG retrieves resources that are instances of Wikidata's "event" or "occurrence" classes or subclasses. From each language-specific edition of DBpedia, EventKG recovers resources that are instances of dbo:Event or its subclasses. From Yago, it extracts resources that have been linked to via the owl:sameAs property from the events recovered from the first two knowledge bases, and it uses a set of rules to query for resources that match hand-made patterns based on Wikipedia categories. EventKG captures event temporal relations, i.e., event date, from Wikidata and Yago. The system also uses a series of mapping tables constructed manually to look for predicates that indicate complex events or series of events. This information is later modeled by EventKG in the resulting aggregate knowledge graph.

The WCEP presents daily events contributed by Wikipedia editors, and usually consist of a brief description of the event, event category, and event date. To incorporate data from WCEP, the authors rely on WikiTimes, a system proposed by Tran and Alrifai in [51]. WikiTimes extracts events listed in WCEP by parsing the structure of the page and indexes the events and their properties, e.g., date, into an Apache Lucene instance[6]. Wikipedia Event Lists are pages that cover a long period of time such as a year, e.g., `https://en.wikipedia.org/wiki/2018`, list events in chronological order and link to their individual event Wikipedia pages. Each entry of the list usually contains the location of the event and a description made of one or two sentences. To detect events from these pages and identify their date, location and description, in [52], Hienert1 et al. proposed a method that uses language-specific regular expressions. Gottschalk and Demidova leverage this method to incorporate additional event information into EventKG.

For each source, EventKG creates a named graph in which it ingests the extracted triples that describe events. Moreover, EventKG constructs a sixth graph that aggregates triples across all sources. To identify triples describing the same events coming from different sources, EventKG follows a heuristics-based approach that evaluates the similarity of the triples' descriptions, time components, and entities linked. Once triples

[6] `https://lucene.apache.org/`

expressing the same events have been identified, EventKG uses a set of rules to merge their properties. For example, it compiles a list with all the locations mentioned across triples of the same event and reduces the list by eliminating elements that contain others, from a geographical perspective. For instance, if the list of locations includes "Los Angeles", "California", and "United States", EventKG can determine that the highest granularity is provided by the resource describing "Los Angeles" and retain it as the definitive event location. For the time component, the system considers each source as casting a vote about what the right event time or date is, and then selects the value with the highest count. In case of draws, EventKG prefers temporal data from some sources over others, e.g., Wikidata over Yago. As a result of this process, ECKG can build a single representation of an event across sources, which is later modeled in RDF.

EventKG attempts to impose a uniform RDF structure to the aggregate events. It uses the Simple Event Model (SEM) [53] ontology to represent events and their details. SEM allows the authors to link events to their participants, to the event's location, and allows them to define instant and interval events via properties that define the beginning and end time of the event. To model series of events, the authors use the dbo:previousEvent and dbo:nextEvent to link events in a chain, and they use the Schema.org [7] ontology to specify that an event has subevents.

A particularity of EventKG is that it can measure how strong the relation of a participant with the event is, which could help determine the most important participant of an event. As a proxy for relation strength, EventKG counts the number of instances in which mentions of the subject and object entities appear in the same sentence across Wikipedia. To assess the popularity of events, the authors assume that popularity is directly proportional to how often the event subject and event object are connected via links; hence, EventKG counts the number of links from the Wikipedia page of the entity acting as the event's subject to the page of the entity acting as the event's object. These metrics become additional properties describing the event at the RDF level. Additionally, EventKG also links subject and object entities to their representations in other knowledge bases through the owl:sameAs property, and explicitly decorates the assertion with a property indicating from which source it was gathered, e.g., Wikidata.

[7] `https://schema.org/`

This chapter reviewed the relation extraction and open information extraction tasks and presented various methods that can solve such tasks. In addition, the chapter described some of the most important systems that build knowledge graphs automatically. The next chapter will introduce NewsTextAnalyzer, the system built as part of this project, and provide reasoning behind system design decisions.

Chapter 3

Design

This project presents NewsTextAnalyzer, a system that can automatically build a knowledge graph based on assertions and events extracted from political news articles. Moreover, the system can associate sentiment that qualifies people's reaction on social media to specific events. A high-level view of the entire NewsTextAnalyzer system is shown in Figure 3.1.

During the knowledge graph construction process, NewsTextAnalyzer utilizes a Lucene index called Person Resources Index for entity resolution. The Person Resource Index was created during an offline step by retrieving person entities from DBpedia. To associate sentiment information to specific events, the system uses a second index called Tweets Index. TweetsIndex is constructed by collecting tweets and replies from a publisher's Twitter account. The resulting knowledge base is stored in a triplestore, specifically OpenLink Virtuoso Universal Server[1]. The system also provides a simple querying interface that retrieves information, not only from the local triplestore but also from remote ones via SPARQL. Details about design decisions that drive the implementation of NewsTextAnalyzer are described in the remaining of this chapter while specifics about the implementation of NewsTextAnalyzer are provided in the next chapter.

[1] `https://virtuoso.openlinksw.com/`

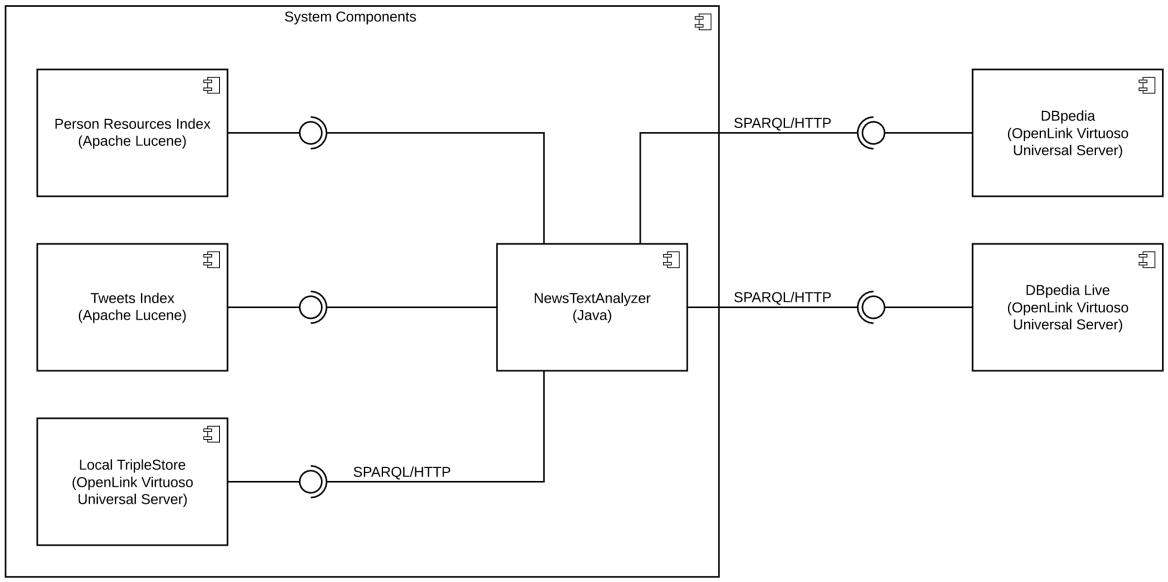


Figure 3.1: NewsTextAnalyzer - Component Diagram

3.1 Gathering Political News Articles

This project required a corpus of political articles from which to extract events. During the last few years, many online publishers that produce high quality journalism pieces have taken steps to limit access to their content, especially via programmatic channels. A significant number of well-renowned publishers, e.g., Washington Post[2], have joined an association called News API[3] that has developed an API that aggregates access to their content. Consequently, these publishers, e.g., BBC[4], shutdown their proprietary API endpoints or did not find a reason to invest in developing them in the first place. Unfortunately, News API lacks features that would allow filtering articles based on topic or editorial style like opinion or analysis pieces. Furthermore, with standard plans, News API only returns the first few hundred characters of an article, instead of the entire content, which would severely limit the fine-grained events that could be extracted from such a short text. Other publishers, such as The New

[2] `https://www.washingtonpost.com/`
[3] `https://newsapi.org/`
[4] `https://www.bbc.com/`

York Times[5], still offer first party APIs[6] but they are considerably rate limited, lack topic filtering capabilities or only provide access to articles that go back a few months. This project evaluated all of these options and concluded that the proprietary The Guardian's API[7] would enable gathering the most appropriate corpus. Reasons for selecting The Guardian's API include its breadth of filtering capabilities, the extent of its article coverage that allows it to surface pieces dating back 20 years or more ago, and the full access to articles' text it provides with standard subscriptions.

The possibility of crawling a publisher's website and parsing web pages containing news articles was considered too. However, because most online publishers have focused on embedding advertising and promotional text within articles' main body of content, it is difficult to determine simple patterns to distinguish which HTML pieces contain article information. In addition, in an effort to provide a better user experience, publishers added a considerable amount of styling, formatting and imagery to article pages, which results in more complex Document Object Model (DOM)[8] trees from which is challenging to only capture the article's text. Furthermore, most publishers do not provide pages that list and categorize articles by topic, so crawling might only surface a small subset of popular articles that are often linked to within the publishers' website. Lastly, additional pattern matching methods would be required to identify the article's topic, type and date of publication in order to determine if the piece is relevant for this project's purposes. For all of these reasons, this project abandoned the crawling option and decided to gather news articles via API.

3.2 Gathering Public Reaction to Political News

Politics is one of the content categories to which people tend to react emotionally and express their opinions online. The goal of this project includes incorporating the public's reaction to political events reported on the news, e.g., if the public's sentiment was

[5] `https://www.nytimes.com/`
[6] `https://developer.nytimes.com/`
[7] `https://open-platform.theguardian.com/`
[8] `https://www.w3.org/DOM/DOMTR`

positive or negative with respect to a particular politician winning a presidential election. The most popular platforms for English speakers to voice their opinion include social media sites such as Facebook[9] and Twitter, and content sites such as YouTube[10]. People also voice their opinion in other social media platforms such as Instagram[11], Pinterest[12], Reddit[13], or LinkedIn[14], but often the kind of topics discussed in those sites are not politics-centric. Content on Instagram and Pinterest revolves around lifestyle related news; Reddit users mostly discuss news about niche topics such as the introduction of new technology products; while content posted on LinkedIn is mostly composed of business or professional related articles.

Most news publications have a wider presence on Twitter and Facebook than on YouTube, although this has been changing recently. For example, the BBC has many more granular accounts on Twitter and Facebook than on YouTube. Although the BBC account that focuses on sports related news, @BBCSports, exists on all three sites, the account that focuses on politics, @BBCPolitics, only exists on the first two. Moreover, observations made on a sample of publishers indicate that most of them create more posts on Twitter and Facebook that are closely associated to the news articles they publish on their sites, than upload videos to YouTube.

When assessing social media APIs in terms of their level of access to content, this project discovered that YouTube's API[15] has granular endpoints that allow retrieval of comments, and that it provides great temporal coverage, possibly supporting the retrieval of comments that are years old. On the other hand, YouTube's API impose a complicated quota policy, and drastically reduced the API rate limits for non-paid accounts at the beginning of 2019. However, as mentioned before, the biggest hurdle to using YouTube as a source of public sentiment is the relatively low number of videos uploaded directly related to news published. If YouTube were to be used as source of sentiment, it would demand the use of complicated techniques to identify the relevant

[9] `https://www.facebook.com/`
[10] `https://www.youtube.com/`
[11] `https://www.instagram.com/`
[12] `https://www.pinterest.com/`
[13] `https://www.reddit.com/`
[14] `https://www.linkedin.com/`
[15] `https://developers.google.com/youtube/v3/`

YouTube videos related to news articles and events described in there. In comparison to YouTube's API, Facebook's API[16] is at the other extreme of the spectrum, being the most restrictive. As a result of privacy incidents, Facebook introduced a set of platform policies that require client applications to gain permissions before retrieving user data. In addition, Facebook changed the comments API endpoint to filter out comments made by other users than the one associated to the client application using the API. In other words, it is not possible to retrieve comments to Facebook posts anymore. Twitter's API[17] capabilities lie between YouTube's and Facebook's. It does not have granular endpoints to retrieve replies as YouTube's does, but there are workarounds to circumvent that obstacle. Client applications require explicit authorization from Twitter to extract posts and replies, but they do not need authorization from the users that created the posts or replies. Moreover, Twitter's API does not have extreme privacy policies that restrict retrieving user replies. The main disadvantage of Twitter's API is the low temporal coverage it provides with non-paid plans. Even with a paid account, the API quotas imposed by Twitter could become a problem: the lack of an endpoint designed to return replies forces developers to inefficiently use API calls to invoke other endpoints through which they can capture replies to accounts. However, the fact that matching Twitter posts to news articles published online is relatively straightforward compensates for these disadvantages and led this project to choose Twitter as the source for public sentiment on assertions expressed in political news.

Given that this project selected The Guardian as its source of political news articles, the @GdnPolitics account was selected as the channel from which to gather Twitter posts and replies. The @GdnPolitics account focuses on sharing information about worldwide politics, while the @guardian account shares information about a wide range of topics, from entertainment to sports. A particularity of the posting style of @GdnPolitics account is that their Twitter posts tend to contain the title of the news article it relates to, which facilitates matching tweets to assertions captured from news articles, see Figures 3.2 and 3.3. This project assumes that people wrote replies to @GdnPolitics posts to express their opinions primarily about the text used by the

[16] `https://developers.facebook.com/docs/apis-and-sdks/`
[17] `https://developer.twitter.com/en/docs.html`

Figure 3.2: Example of a news article from The Guardian's website

Figure 3.3: Example of a tweet posted by @GdnPolitics linking to a news article

tweet and not about details mentioned in the article. This assumption justifies the matching assertions extracted from news articles to tweet posts in order to associate the sentiment of the replies to the assertion.

Sadly, the @GdnPolitics Twitter account is not as popular as other publishers' accounts, e.g., @BBCPolitics, so the average number of replies per post is in the single-digits range. The possibility of matching assertions extracted from The Guardian news articles to sentiment extracted from replies to @BBCPolitics tweets was considered but eventually discarded because of the additional methods that would be required to assess if a tweet and an assertion are describing the same fact.

3.3 Triple Extraction

To obtain assertions, including events, from news articles, an open information extraction approach that uses a lightweight NLP pipeline was selected. As explained in Chapter 2, open information extraction is a task that can capture relations without providing training data or specifying relations of interests in advance. It is important to note that extraction accuracy achieved by this family of methods is low compared to traditional supervised extraction methods. Semi-supervised relation extraction methods were also considered and short experiments were conducted to test if they could work for the purposes of this project. Experiments demonstrated that a corpus far larger than the one collected for this project would be required in order for such approach to be partially effective.

This project selected ReVerb as the open information extraction method to extract triples from text. A number of reasons support such decision. First, compared to other methods, ReVerb only relies on low level syntactic features, such as chunk and POS tags, that can be easily generated from text by publicly available NLP tools. Second, the features ReVerb relies on are at least an order of magnitude less computationally expensive to derive than more sophisticated and deep NLP features such as dependency parse or constituency parse trees, as explained by Etzioni et al. and Banko et al. in [28] and [31], respectively. Methods that rely on the latter, e.g., those based on SRL

such as [44], usually require additional significant investment in software engineering tasks to make the document processing pipeline work in a distributed manner, so that it can process documents in a reasonable time interval. Lastly, ReVerb's algorithm is based on short and relatively simple syntactic patterns that work effectively for over 85% of English binary verbal relations, which is sufficient for the scope of this project. Consequently, a pipeline based on ReVerb can process hundreds to thousands of sentences per minute on a single standard personal computer, while one based on more sophisticated features, such as constituency parse trees and SRL, might only process sentences in the range of tens per minute on the same kind of hardware.

3.4 Entity Recognition and NLP Tools

To represent person entities appropriately within a knowledge graph, mentions of peoples' names need to be identified in the article's text. Moreover, because this project considers an event as an action that occurred at a precise moment in time, NewsTextAnalyzer also needs to capture temporal entities from the article's sentences. Both of these requirements can be achieved by performing NER tagging, a standard and common task supported by many NLP toolkits, such as Stanford CoreNLP, spaCy[18] or Apache Open NLP. NewsTextAnalyzer prefers Stanford CoreNLP because it is generally considered to be up to par with the state-of-the-art, and often achieves marginally better F-measure scores than other toolsets in low level NLP tasks across different kinds of corpus, as Rizzo et al. demonstrated in [54].

3.5 Entity Resolution and Linking

Linked Data [11] best practices demand interconnecting entities to broader knowledge bases whenever possible. This project selected DBpedia as the main external knowledge graph to use for the purpose of linking person entities. DBpedia was selected because it offers great entity coverage about people, with over a million resources of

[18] `https://spacy.io/`

type dbyago:Person100007846, the Yago class most often associated to people by this knowledge graph. In addition, DBpedia's relation coverage is far broader than Yago's given that it does not take a set of relations of interests as input. To resolve the external resource that best fit an entity extracted from a news article, this project built the Person Resources Index from DBpedia data. The Person Resources Index allows lookups of entities of type "person" by their name, emulating part of the functionality provided by DBpedia Spotlight but running such service in a local environment. Like EventKG, NewsTextAnalyzer uses the owl:sameAs property to link out an entity with its corresponding resource in DBpedia.

As explained in Chapter 2, in theory, the DBpedia knowledge graph is completely constructed from scratch using Wikipedia data dumps every 6 months to 18 months. It is possible to query the resulting graph through a public SPARQL endpoint[19]. However, from experiments conducted with the endpoint, it appears the last execution of the full DBpedia knowledge graph construction process occurred more than three years ago. Thus, the information presented by this endpoint is out-of-date and does not reflect the current state of the world. For example, on DBpedia, Donald Trump is only described as a 2016 U.S. Presidential Candidate, and not yet as U.S. President, see Figure 3.4. Fortunately, the DBpedia Live system described by Morsey et al. in [55] attempts to solve this problem by running a continuous process that uses the Wikipedia change logs to update a second DBpedia knowledge graph. Sadly, the DBpedia Live knowledge graph does not incorporate all the information that exists in the DBpedia knowledge graph, which forces NewsTextAnalyzer to use federated SPARQL queries directed to both, DBpedia and DBpedia Live[20] endpoints, to get the full picture of an entity and merge properties of interest.

3.6 Assertion and Event Representation

A triple returned by ReVerb as a result of processing a sentence is considered an assertion. An assertion for which NewsTextAnalyzer was able to associate temporal infor-

[19] `http://dbpedia.org/sparql`
[20] `http://dbpedia-live.openlinksw.com/sparql`

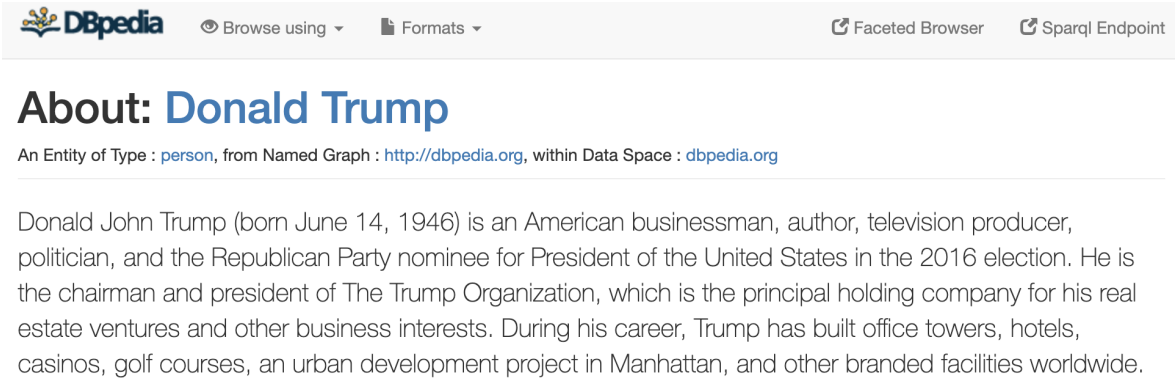


Figure 3.4: Example of an out-of-date DBpedia page

mation is considered an event. In order to represent assertions and attach metadata information to them, NewsTextAnalyzer employs the Singleton Property, as described in [56]. In simple terms, the Singleton Property approach considers the predicate as the main component of the primary triple that represents the assertion and considers occurrences of predicates as instances of a more generic predicate. By making the predicate of the primary triple an instance, one can attach metadata to it. A comparable approach called reification can also handle metadata, but the Singleton Property can do it by using fewer triples per main assertion, resulting in less disk space being consumed by the knowledge graph. The Singleton Property allows NewsTextAnalyzer to associate provenance information to the main triple, in a similar fashion that EventKG does. For each main triple, NewsTextAnalyzer attaches the following properties as metadata:

- hasSource, provides the URL of the news article from which the assertion was extracted
- hasSourceSentence, provides the entire sentence from which the assertion was extracted
- rawSubject, the text fragment that ReVerb returned as the triple's subject
- rawPredicate, the text fragment that ReVerb returned as the triple's predicate
- rawObject, the text fragment that ReVerb returned as the triple's object

- confidenceScore, the confidence score that ReVerb assigned to the extracted triple
- tokenList, the entire phrase composed of the triple
- xsdSourceDate, the publication date of the source news article as an xsd:Date literal

In the case of events, NewsTextAnalyzer associates the type dbo:Event to the main assertion's predicate and describes the date of event in two ways. In the first way, the system represents the date as an xsd:Date literal associated to the assertion's predicate through a property called "xsdDate". In the second way, NewsTextAnalyzer defines a resource of the class DateTimeDescription from the Time Ontology, as described in [57], and associates it via a property from such ontology called "time:inDateTime". NewsTextAnalyzer uses the following Time Ontology properties to further describe the DateTimeDescription instance:

- time:unitType, set to time:unitDay for all events
- time:day, set to the day component of the literal value of the "xsdDate" property
- time:dayOfWeek, set to the resource that represents the appropriate day of the week, e.g., Monday, in the time ontology for the event date. NewsTextAnalyzer uses a Gregorian calendar to determine the right value for this and the following two properties
- time:dayOfYear, set to the day number of the year for the event date, considering that January 1st is numbered as 1, February 1st is numbered as 32, and so on.
- time:week, set to the week number of the year for the event date
- time:month, set to the month component of the literal value of the "xsdDate" property
- time:year, set to the year component of the literal value of the "xsdDate" property

The temporal information saved following the Time Ontology is not currently used by the Querying module of NewsTextAnalyzer, but because the information has been stored in granular detail, it could be leveraged by complex queries issued directly to the knowledge graph via a SPARQL endpoint. Some of such queries could ask for events that occurred in a specific week of the year or events that occurred during weekends.

Lastly, when an assertion is matched to a tweet post through the Tweets Index, NewsTextAnalyzer adds the following properties to the assertion's predicate:

- hasTweetSource, provides the URL of tweet post to which the assertion has been matched
- sentimentScore, provides the aggregate sentiment score calculated on the replies to the tweet post

Figures 4.17 and 4.18 present examples of the complete RDF representation of an event.

This chapter elaborated on design decisions that influenced the way in which NewsTextAnalyzer was constructed. The next chapter continues by describing the implementation of NewsTextAnalyzer in detail.

Chapter 4

Implementation

This chapter presents the details of the implementation of all the components that are part of the NewsTextAnalyzer system, including tools built to collect data, the NLP pipeline designed to process news articles, and the querying interface created to demonstrate the use of the resulting knowledge graph.

The name of the components referred to in this section match the name of their classes and scripts in the source code of the projects included in the accompanying USB flash drive.

4.1 Data Gathering

For data gathering purposes, this project opted to build utilities in Python[1] because of the language's simplicity and minimal verbosity, its practical data structures, and its modules that facilitate integration with web services. Those modules include: the Requests[2] module, which facilitates constructing HTTP API requests and the OAuthLib for Requests[3] module, which makes it easy for applications to authenticate themselves against web services.

[1] `https://www.python.org/`
[2] `https://2.python-requests.org/en/master/`
[3] `https://github.com/requests/requests-oauthlib`

4.1.1 NewsCollector and NewsPreprocessor

The NewsCollector component retrieves news articles from The Guardian via API calls to their content endpoint[4]. This endpoint provides sufficient flexibility in terms of the search criteria to only retrieve political news articles written in English between specific dates. Moreover, the endpoint allows filtering out opinion pieces which in turn allows to reduce the number of non-factual triples generated by the knowledge graph construction process. Compared to other publishers' APIs, The Guardian's content endpoint allows retrieval of the entire article body instead of just a summary or abstract, which contributed to the extraction of more triples.

The news articles included pieces about U.K., U.S. and Europe politics. The NewsCollector component explored news articles since January 1st, 2000 on-wards, gathering more than 216,500 documents that were saved to disk as files in JavaScript Object Notation (JSON) format. The component organized the documents per date, saving news articles published on the same day in the same folder.

The NewsPreprocessor component examined each JSON document and reduced the documents to the most important information: news article identifier, title, URL, date of publication, and the main body of content. Furthermore, because the main content returned by the endpoint contained HTML, the component uses the HTMLParser[5] Python module to strip the content from tags such as <p>. In addition, the component replaces some tags, such as <h1>, <figcaption>, or <figure>, with punctuation marks to appropriately convert headings and subtitles to valid sentences from an NLP perspective.

NewsCollector and NewsPreprocessor are included as a Python projects in the accompanying USB flash drive.

[4] `https://content.guardianapis.com/search`
[5] `https://docs.python.org/3/library/html.parser.html`

4.1.2 TweetsCollector and TweetIndexer

The TweetsCollector component performs two tasks. First, it retrieves tweets authored by The Guardian Politics team, @GdnPolitics, and saves the tweets to files on disk. Second, it retrieves replies from the public to such tweets. Then, it estimates the sentiment score of the replies, calculates the average sentiment, and updates the appropriate files on disk.

For the first task, the TweetsCollector component uses the timelines endpoint, [6], to get content from the @GdnPolitics timeline. The component saves the tweet posts it recovers to files in JSON format. Each file contains the tweet's id, URL, entire tweet text, and its creation date. The component organizes the tweet posts files created on the same day in the same folder. The component is designed to run periodically. For that reason, it saves configuration information that indicates what was the last date and tweet it collected, so that it can use this information within the parameters of subsequent API calls and only retrieve new tweet posts it has not observed yet.

Unfortunately, Twitter caps access to users' timelines to the 3,200 most recent tweets, limiting the content available for retrieval to only a few months for accounts with high engagement [58]. Considering that the component filters out retweets, the component only retrieved approximately 2,700 original tweet posts, the oldest one having March 8th, 2019 as creation date.

Regarding the second task, it is important to reiterate that Twitter's API does not offer a dedicated endpoint to retrieve replies to specific tweets. Thus, the component is forced to use the search tweets[7] endpoint, which provides some filtering capability by username. Effectively, the endpoint allows selecting only tweets that are replies to a specific user, in this case @GdnPolitics. When TweetsCollector finishes exploring tweets authored by @GdnPolitics, it loads a mapping of the tweet posts saved on disk from previous runs, including their identifiers. As the component retrieves tweet replies from the search tweets endpoint, it cross-checks if those replies belong to any tweet post in the mapping. If so, the component processes the reply text through a sentiment

[6] `https://api.twitter.com/1.1/statuses/user_timeline.json`
[7] `https://api.twitter.com/1.1/search/tweets.json`

analyzer and computes a sentiment score for the reply. Sentiment analysis is conducted via a wordlist-based method. The component uses the afinn[8] Python module, which implements a sentiment analyzer that uses a lexicon of more than 3,0000 words, each with a polarity score; details of this method are described in [59]. Lastly, the reply and its sentiment score are added to the respective tweet post file, and the running average sentiment score and reply count is recalculated and updated on the same file. The component retrieves a sample of a maximum of 100 replies per tweet post; once that limit is reached, the component does not consider that tweet post in the mapping anymore. For an example of a tweet post file, see Figure 4.1.


```
{
  "screen_name": "GdnPolitics",
  "id": 1148144709019131900,
  "id_str": "1148144709019131904",
  "url": "https://twitter.com/GdnPolitics/status/1148144709019131904",
  "full_text": "Liam Fox defends UK ambassador to US in row over leaked memos https://t.co/klQqgUj9Vm",
  "status": 0,
  "created_at": "Mon Jul 08 08:20:04 +0000 2019",
  "reply_count": 2,
  "replies": [
    {
      "reply": "@GdnPolitics Whoever has been hanging on to these has leaked them at an opportune time,
and whatever their intent, it is another blow to UK's credibility on the world stage. I take confidence
that what diplomats report is only with our security and interests in mind. Others are doing the same",
      "sentiment_score": 2
    },
    {
      "reply": "@GdnPolitics I was convinced the Ambassador was talking about the UK Government, had to
read it several time to realize he was talking about the USA",
      "sentiment_score": 1
    }
  ],
  "avg_sentiment_score": 1.5
}
```


Figure 4.1: Example of a tweet post file generated by TweetsCollector

Sadly, with a Twitter API standard account, it is only possible to retrieve replies that are at most seven days old [60]. With paid Twitter API access, the entire tweet archive could be searched, but the quota of maximum API calls per month is so low that it is unlikely the system could retrieve replies to @GdnPolitics posts from years ago.

TweetIndexer, a second component written in Java[9], creates a Lucene index by

[8] `https://github.com/fnielsen/afinn`

[9] `https://www.java.com/en/`

going through the list of tweet post files and adding each as a document. All tweet post file fields are retrievable, the creation date and reply count fields are searchable, but only the main tweet text is processed, i.e., tokenized, stemmed, etc. During the enrichment phase of the knowledge graph creation process, NewsTextAnalyzer utilizes this Lucene index to match triples extracted from news articles to tweet posts and assigns the average sentiment score calculated from the replies to such triples.

TweetsCollector is included as a Python project in the accompanying USB flash drive. TweetIndexer is a Java class within the NewsTextAnalyzer Java project also included in the USB flash drive.

4.1.3 DBpediaScraper and DBpediaRecordRetriever

A crucial benefit of creating knowledge graph is the ability to link them to existing ones. Through this process, it is possible for a system to use federated queries, i.e., solving complicated queries by issuing requests to multiple triplestores that have parts of the answer. The focus of this project is around politicians and their actions. Thus, the natural interlinkage occurs among people identified in the news articles from which the knowledge graph is built and resources that represent those people in other knowledge graphs, such as DBpedia and DBpedia Live.

Unfortunately, during experiments with DBpedia, the project found that the response time of SPARQL queries that look for resources with similar names to one given as parameter was in the order of tens of seconds, and in more than 30% of the cases the queries timed out. To overcome this problem, and speed up the knowledge graph construction process, this project created the DBpediaScraper and DBpediaRecordRetriever components. These components query the DBpedia and DBpedia Live SPARQL endpoints for resource records that match three criteria:

- Resources of type dbyago:Person100007846; dbyago:Person100007846 is the most common type associated to resources that represent people on DBpedia
- Resources with the rdfs:label property

- Resources whose rdfs:label property has a variant in English

Both endpoints restrict the maximum number of records returned within a response to a SPARQL query, with the current limit set to 10,000 [61]. For that reason, DBpediaScrapper first issues a query to count the number of resources that match the criteria described above. Then, it creates a queue of work items, each representing one request to obtain 10,000 different records, with as many work items as necessary to cover all the records, according to the count retrieved by the first query. DBpediaScrapper also creates a pool of DBpediaRecordRetriever threads, passes the queue of work items, passes a queue of sets of result records, initiates the execution of the threads, and waits for their completion.

Each thread executing the DBpediaRecordRetriever component inspects the queue of work items for one that is pending retrieval. If one is found, it pulls the work item from the queue, prepares a SPARQL query with pagination clauses, i.e., LIMIT and OFFSET, to retrieve the appropriate subset of records, and sends the request to the endpoint. The query obtains the resource id, the resource label, the value of the dbo:birthDate property, in case the resource had such property associated, and the count of the number of types associated to the resource that are either dbyago:Politician110450303, umbelrc:Politician, dbo:Politician or dbyago:Person100007846. The first three are the types most commonly associated to politicians within the DBpedia-Yago ontology based on observations when testing the endpoint. The thread running the component extracts the records, adds them to the queue of sets of result records, and repeats the process. If there are no more work items pending processing, the thread terminates.

The DBpediaScrapper component performs this logic for DBpedia and DBpedia Live because as stated before, the complete picture of the person resource is often divided among both knowledge graphs. Thus, once the threads running the DBpediaRecordRetriever component complete, the DBpediaScrapper component merges the queue of result records based on the person resource id. For example, on DBpedia, the resource that represents Theresa May, dbr:Theresa_May has the dbo:birthDate property, and on DBpedia Live, the same resource is linked to the type umbelrc:Politician.

Thus, the complete representation of the resource will have both properties. Whenever a property is repeated in both, DBpedia and DBpedia Live, DBpediaScrapper prefers the information provided by DBpedia Live.

DBpediaScrapper leverages the PersonResourceLookup component to create a Lucene index based on the person resource records recovered. All person resource record fields are retrievable, but only the resource label is processed, i.e., tokenized, stemmed, etc. During the knowledge graph creation process, NewsTextAnalyzer uses this Lucene index to lookup people by name and interlink resources from the local knowledge graph to DBpedia. In total, more than 1 million resources were recovered and indexed.

DBpediaScraper and DBpediaRecordRetriever are both classes of the NewsTextAnalyzer Java project.

4.2 Knowledge Graph Construction

The components of the NewsTextAnalyzer project perform the knowledge graph construction task. Such components are written in Java. All components mentioned in this section are either classes within the NewsTextAnalyzer Java project or Lucene indexes. The main reasons behind the decision to use Java are:

- There are NLP tools implemented in Java that support tasks, such as NER, that are essential for the entity resolution. NewsTextAnalyzer uses Stanford CoreNLP and Apache OpenNLP libraries when required.
- There are Java libraries that facilitate the integration to triplestores. NewsTextAnalyzer uses Apache Jena[10] to insert triples to and query Virtuoso OpenLink Server, a popular triplestore.
- Sophisticated development tools exist in the Java ecosystem that allow the inspection, debugging, and profiling of code. During development, the Eclipse[11]

[10] `https://jena.apache.org/`
[11] `https://www.eclipse.org/ide/`

and NetBeans[12] IDEs were used and their Debugger and Profiler[13] tools, respectively, were helpful in identifying bugs and detecting bottlenecks caused by the sub-optimal use of NLP tools. Without them, the runtime of the knowledge graph construction process would be an order of magnitude greater.

- Robust and well-tested indexing and search technology software is available, such as Apache Lucene, that facilitated operations such as lookups and matching.

The knowledge graph construction process consists of two distinct phases, extraction, and enrichment. The extraction phase is governed by the NewsCorpusProcessor and PipelineManager components, which take each news article and its sentences through six different pipeline steps. During the enrichment phase, the SentimentEnricher component associates sentiment score to specific predicates by inserting additional triples. A detailed component diagram of the NewsTextAnalyzer pipeline is presented in Figure 4.2.

4.2.1 NewsCorpusProcessor and PipelineManager

The NewsCorpusProcessor retrieves the news articles generated by the NewsPreprocessor component. If necessary, the NewsCorpusProcessor could filter which news article to process by date. The component passes each news article to the PipelineManager component. The PipelineManager component uses the Stanford CoreNLP to tokenize the text and split the news article content into sentences. From this point, the PipelineManager passes the news article and the sentences down to underlying pipeline steps and forwards the relevant output of each step to the next, in this order: Referencer, ReVerbOIE, EntityValidator, IntraLinker, InterLinker, and VirtuosoPersistor.

4.2.2 Referencer

The first pipeline step is the Referencer component. It utilizes the Stanford CoreNLP library to perform NER on each sentence. By configuring a basic Stanford CoreNLP annotator that detects basic entities such as “person” and “organization”, but disabling

[12] `https://netbeans.org/`
[13] `https://profiler.netbeans.org/`

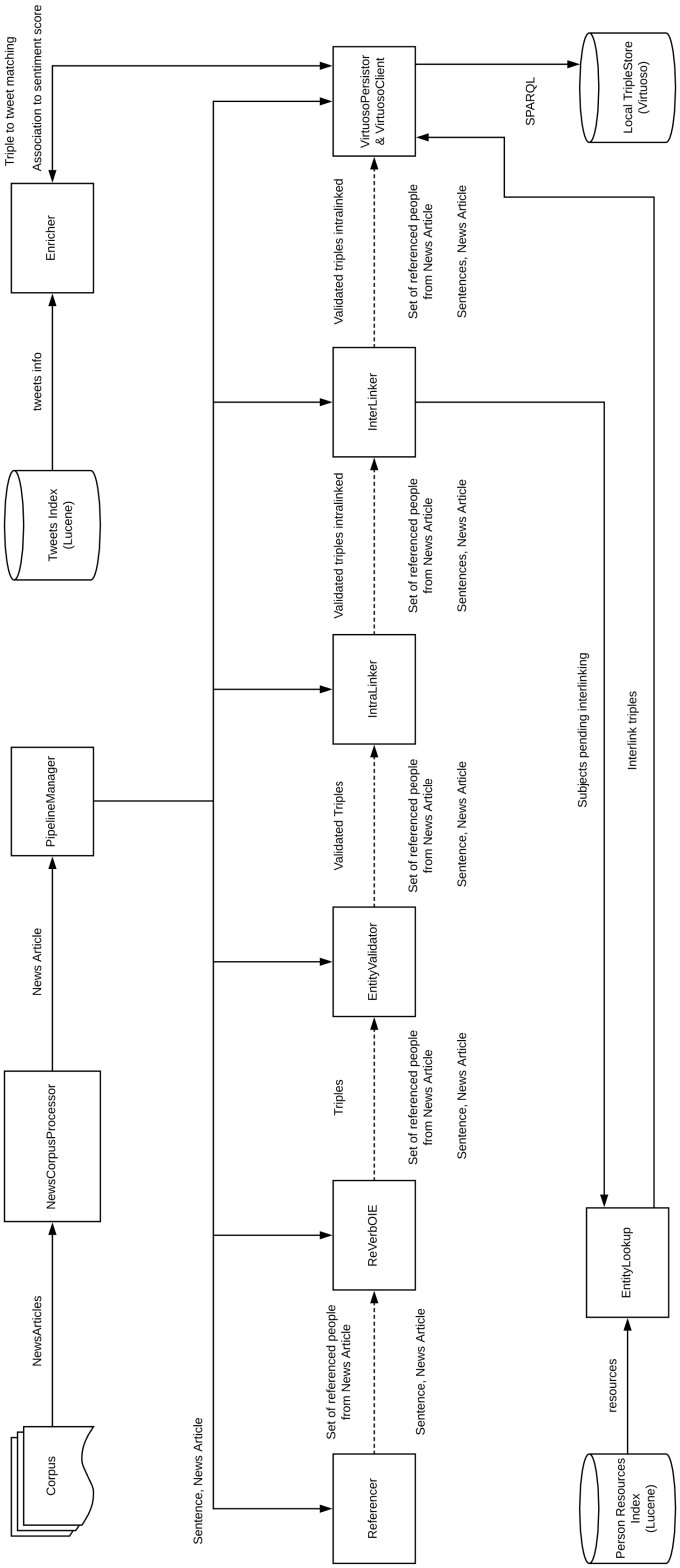


Figure 4.2: NewsTextAnalyzer - Detailed Component Diagram

detection of complex entity types, such as “date”, or “number”, the Referencer can process sentences twice as fast as if it were to use the default annotator. Once each sentence has been annotated with NER tags, the Referencer searches for patterns that indicate a sequence of names. Each sequence of matches is added to an ordered set created by the Referencer when it receives the first sentence of a news article. This ordered set of matches is passed to the following pipeline steps and used by the IntraLinker to associate different variations of the name that probably refer to the same person. Figure 4.3 depicts a sentence and its annotated NER tags. The ordered set managed by the Referencer after processing such sentence would contain two sequences: ”Robert Mueller” and ”Donald Trump”.

Tokens	Robert	Mueller	did	not	exonerate	Donald	Trump	.
NER Tags	PERSON	PERSON	0	0	0	PERSON	PERSON	0

Figure 4.3: NER tags on a sample sentence

4.2.3 ReVerbOIE

The ReVerbOIE component is a wrapper that uses ReVerb classes to extract triples from the sentences coming from the previous pipeline step. To do so, the component first chunks a sentence using a ReVerbChunker based on Apache OpenNLP’s tokenizer, POS tagger, and chunker. Once the sentence is chunked, the ReVerb extractor recovers triples from it. In addition, the component uses ReVerb utility classes to calculate the confidence score on the triple extracted, and discard those that are below a threshold, currently set at 0.75. The component also examines the subject’s chunk and POS tags and discards triples whose subjects are not proper nouns. Through this method, ReVerbOIE can filter out generic triples that do not contribute information about specific politicians. Performing NER is orders of magnitude slower than chunking or POS tagging. Hence, this step helps reduce the domain of sentences and triples over which subsequent pipeline steps would perform NER on, speeding up the entire process. The surviving triples are added to a list of extracted triples and passed down the pipeline. See Figure 4.4 for an example of a sentence annotated with chunk and POS tags. See Figure 4.5 for an example of the output of the ReVerb extractor.

Tokens	Robert	Mueller	did	not	exonerate	Donald	Trump	.
Chunk Tags	B-NP	I-NP	B-VP	I-VP	I-VP	B-NP	I-NP	O
POS Tags	NNP	NNP	VBD	RB	VB	NNP	NNP	.

Figure 4.4: POS and chunk tags on a sample sentence

Subject	Robert Mueller
Predicate	did non exonerate
Object	Donald Trump

Figure 4.5: Example of a triple produced by ReVerb

4.2.4 EntityValidator

This component receives the extracted triples and processed sentence and conducts additional analysis on them. To extract a higher level of meaning, the EntityValidator component relies on Stanford CoreNLP classes to perform NER on the triple's subject. The component analyzes the NER tags of the triple's subject and discards triples whose subjects do not contain a "person" or "organization" entity type. For examples of these cases, see Figures 4.6 and 4.7.

For the surviving triples, the component extracts the reference to people's names from the subject, determines if the subject's NER tags are all of "person" type, and if so, marks the triple's subject as a candidate for interlinking, which becomes useful for the intralinking and interlinking steps further down the pipeline.

Tokens	President	Donald	Trump	visited	France	.
NER Tags	TITLE	PERSON	PERSON	O	LOCATION	O

Figure 4.6: NER tags for a sample sentence whose subject involve different entity types

Tokens	Trump	ramps	up	campaign	to	distort	Mueller	report	finding	.
NER Tags	PERSON	0	0	0	0	0	PERSON	0	0	0

Figure 4.7: NER tags for a sample sentence whose subject only involve "person" entity type

If after such processing there are triples remaining for the giving sentence, the component runs an additional NER task over the entire sentence to look for temporal expressions. For that purpose, it configures a Stanford CoreNLP NER annotator to detect entities of type "date" and "time". The component assumes that the temporal expression is connected to the triples extracted from the sentence. For an example of a temporal expression within a sentence, see Figure 4.8.

Tokens	Trump	entered	the	Republican	field	in	June	2015	.
NER Tags	PERSON	0	0	ORGANIZATION	0	0	DATE	DATE	0

Figure 4.8: NER tags for a sample sentence with a temporal expression

By analyzing a random sample of temporal expressions from the corpus, the following patterns were identified as being able to detect the most common expressions:

- ([ner: TIME])+ [ner: DATE]
- ([ner: DATE] [ner: TIME])
- ([ner: DATE]){2,}

Once the component extracts the temporal expression, it converts it from a string representation into a formal date representation. For that purpose, the EntityValidator relies on Natty[14], a Java library that provides a parser that takes natural language and applies language recognition and translation techniques to generate candidate dates. Some temporal expressions found within the corpus use relative terms such as "last week" or "last month". To be able to resolve such expressions, in addition to feeding

[14] `http://natty.joestelmach.com/`

the temporal expression string to the Natty parser, the EntityValidator also passes the news article publication date as a reference point, see Figure 4.9. The component takes the Natty output and associates it to the triples. Lastly, the component passes the list of surviving validated triples down the pipeline.

Input, string	Input, reference	Output
"June 2015"	2019/07/09	2015/06/01
"last week"	2019/07/09	2019/07/02

Figure 4.9: Example of transformations performed by Natty

4.2.5 IntraLinker

This component performs two tasks. First, it resolves person names to their formal form. To do so, it takes the validated triples whose subjects are composed of a single person entity, and examines the ordered set of names produced by the Referencer so far. If within such set, the component finds a person name that contains the single-word name present in the triple's subject, it proceeds to replace it for the more complete name from the set. To speed up the search, the IntraLinker maintains a cache of the matches of single-word names to formal names. The cache is managed at a per article level; it is created when the first validated triple of a news article reaches the IntraLinker, and disposed when it is turn to process the next news article. Figure 4.10 showcases an example in which the subject of the triple, "Mueller", would be matched to the "Robert Mueller" entry in the set that was produced when the Referencer evaluated the sentence in Figure 4.3.

	Subject	Predicate		Object				
Tokens	Mueller	broke	his	two-year	public	silence	on	Wednesday
NER Tags	PERSON	0	0	0	0	0	0	DATE

Figure 4.10: Sample sentence with a subject that matches an entry within the set produced by the Referencer

The IntraLinker also attempts to connect objects to previously observed subjects. The component creates a global map during initialization, and every time it inspects a validated triple's subject, it adds the subject to the global map. As subsequent triples are explored, the component checks if it has seen such expression as a valid subject before. Figure 4.11 presents an example of this scenario; if the IntraLinker observed the sentence from Figure 4.3 before, then when it processes the sentence from Figure 4.11, it could connect the "Robert Mueller" mention in the triple's object to the subject instance that exists in the global map.

As a result of the IntraLinker's work, most of the name mentions would be normalized to complete person names, which in turn facilitates the interlinkage of resources performed by the next pipeline step. It is important to note that the IntraLinker does not discard triples, it only decorated some with additional information.

	Subject			**Predicate**		**Object**	
Tokens	The	White	House	continue	attacking	Robert	Mueller

Figure 4.11: Sample sentence with an object that matches a previous observed subject

4.2.6 InterLinker and EntitySearcher

The goal of the InterLinker is to connect people mentioned in the news articles to their resource representations in DBpedia and DBpedia Live. It takes the triples from the IntraLinker, and searches for the ones whose subjects are made entirely of at least two tokens and in which all tokens are of "person" entity type. For each triple that meets such criteria, the component examines if the subject has already been interlinked in a previous run of the system. To do so, the component checks a map of linked subjects that was recovered from disk at the beginning of the program execution. See Figure 4.12 for an example of such a map. If the subject has not been interlinked before, then the component adds the subject to a set of subjects pending interlinking.

```
{
  "Annie Lööf": "http://dbpedia.org/resource/Annie_Lööf",
  "Ann Widdecombe": "http://dbpedia.org/resource/Ann_Widdecombe",
  "Lesley Ferrando": null,
  "Jonathan B Jarvis": null,
  "Yvette Cooper": "http://dbpedia.org/resource/Yvette_Cooper",
  "Willie Gallacher": "http://dbpedia.org/resource/Willie_Gallacher",
  "Ashley Taylor": "http://dbpedia.org/resource/Ashley_Taylor",
  "Victoria Woodhull": "http://dbpedia.org/resource/Victoria_Woodhull",
  "Mitch McConnell": "http://dbpedia.org/resource/Mitch_McConnell",
  "Gore Vidal": "http://dbpedia.org/resource/Gore_Vidal",
  "Shaun Bailey": "http://dbpedia.org/resource/Shaun_Bailey",
  "John Redwood": "http://dbpedia.org/resource/John_Redwood",
  "Patrick Shanahan": "http://dbpedia.org/resource/Patrick_Shanahan",
  "Adam Schiff": "http://dbpedia.org/resource/Adam_Schiff",
  "Megan Corton Scott": null,
  "Amber Rudd": "http://dbpedia.org/resource/Amber_Rudd",
  "Ronald Reagan": "http://dbpedia.org/resource/Ronald_Reagan",
  "Hillary Clinton": "http://dbpedia.org/resource/Hillary_Clinton",
  "Theresa May": "http://dbpedia.org/resource/Theresa_May",
  "David Bernhardt": null,
  "John McCain": "http://dbpedia.org/resource/John_McCain",
  "Simon Ellin": null,
  "Jim Fitzpatrick": "http://dbpedia.org/resource/Jim_Fitzpatrick_(actor)",
  "Franziska Ohnsorge": null,
  "Elizabeth Warren": "http://dbpedia.org/resource/Elizabeth_Warren",
  "David Kirkwood": "http://dbpedia.org/resource/David_Kirkwood",
  "Heiko Maas": "http://dbpedia.org/resource/Heiko_Maas",
  "Digby Jones": "http://dbpedia.org/resource/Digby_Jones_(musician)"
}
```


Figure 4.12: Example of a map file managed by the InterLinker

The actual interlinkage occurs as a batch process after the knowledge graph is created. When all news articles are processed by the pipeline and triples have been added to the triplestore, the NewsCorpusProcessor will inform the PipelineManager of this event, and the latter will notify each pipeline step, so they can run finalization tasks. At that point, the InterLinker will leverage the EntitySearcher component to find DBpedia person resources by label, passing the set of subjects pending interlinking and the map of linked subjects. The EntitySearcher, in turn, will go through each entry in the set of subjects pending interlinking and pass each entry to PersonResourceLookup component, which will prepare a phrase query over the indexed label field using the entry passed, i.e., person names, and send the query to the Lucene Person Resources Index. The EntitySearcher component will examine the results returned by the PersonResourceLookup component, and determine which one is the best match.

From experiments conducted, it was observed that the best matching criteria is the Lucene score match. Other criteria that were considered included selecting the person record with the birth date that was the closest to the news article publication

date, and the person record with the highest count of politician types. See Figure 4.13 and 4.14 for an example of interlinking the subject "Donald Trump" to the resource `http://dbpedia.org/resource/Donald_Trump` by conducting a search by name on the index managed by the PersonResourceLookup component. Once the EntitySearcher finishes evaluating all subjects pending interlinking, the InterLinker proceeds to save the map of linked subjects to disk.

	Subject		**Predicate**						**Object**	
Tokens	Donald	Trump	abruptly	ended	a	critical	meeting	with	Democratic	leaders
NER Tags	PERSON	PERSON	0	0	0	0	0	0	0	0

Figure 4.13: NER tags of a sample sentence whose subject is pending interlinking

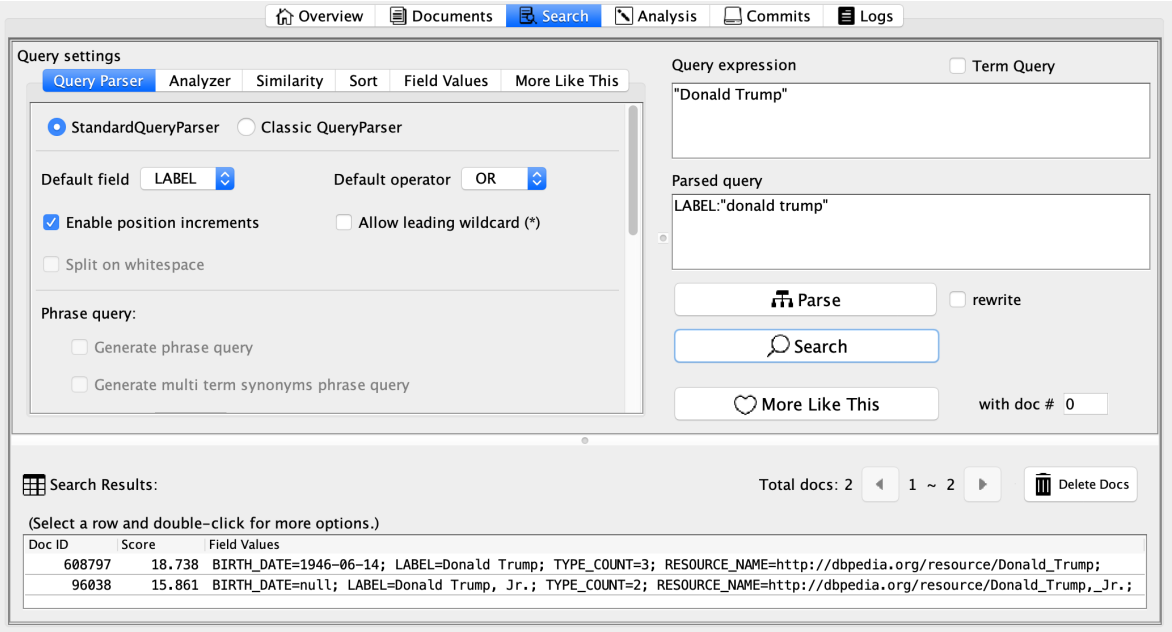


Figure 4.14: Result of a search over the Person Resource Index by label

Once the pipeline steps completed their finalization steps, the NewsCorpusProcessor works with the VirtuosoClient component to insert triples to the triplestore to formalize the links between local person resources and their counterparts in DBpedia. For this purpose, NewsTextAnalyzer leverages the map of interlinked subjects created

by the InterLinker and the EntitySearcher.

4.2.7 VirtuosoPersistor and VirtuosoClient

The last pipeline step is VirtuosoPersistor component which utilizes the VirtuosoClient component to conduct operations over the Virtuoso triplestore. The VirtuosoClient component generates URLs for triples' subjects and for objects marked as linked by the IntraLinker. Because the system models information using the Singleton Property approach, VirtuosoClient needs to maintain a map of predicate roots and their counts to generate the appropriate predicate URLs.

Figure 4.15 presents an example with two triples extracted by ReVerbOIE. When VirtuosoClient prepares the SPARQL query to insert the triples into the triplestore, it transforms the predicates into two different instances of the generic predicate "ran for", as shown in Figure 4.16a. Then, VirtuosoClient inserts additional triples to establish the relation between the instance predicates and the generic predicate, as shown in Figure 4.16b.

	Subject	Predicate		Object	
Tokens1	Clinton	ran	for	the	presidency

(a) Sample triple with predicate "ran for"

	Subject		Predicate		Object			
Tokens2	Victoria	Woodhull	ran	for	the	Equals	Rights	party

(b) Another sample triple with predicate "ran for"

Figure 4.15: Sample triples with the same predicate

It is important to note that the number of triples inserted is much greater than the ones identified by ReVerbOIE and validated through the pipeline. This is due to the fact that metadata information, such as the news article source, needs to be linked to the primary triple identified by ReVerbOIE following the Singleton Property approach. For an example of a complete decorated triple, see Figure 4.17. Notice the additional

	Subject	Predicate	Object
Formal Triple 1	http://myuri/HillaryClinton	http://myuri/ranFor#1	"the presidency"
Formal Triple 2	http://myuri/VictoriaWoodhull	http://myuri/ranFor#2	"the Equal Rights party"

(a) Triples with URLified subjects and predicates

	Subject	Predicate	Object
Triples	http://myuri/ranFor#1	http://myuri/singletonPropertyOf	http://myuri/ranFor
	http://myuri/ranFor#2	http://myuri/singletonPropertyOf	http://myuri/ranFor

(b) Triples using the Singleton Property approach to express assertions with the same predicate

Figure 4.16: RDFication of triples with the same predicate via Singleton Property

metadata properties such as confidenceScore and tokenList.

As explained in Chapter 3, when an event is detected, VirtuosoClient saves the temporal information directly via a xsdDate property. In addition, to support complex temporal queries, VirtuosoClient creates an additional triple to represent the date using the OWL-Time ontology [57]. The component defines this additional triple as one of type time:DateTimeDescription. The component relies on utility classes to get more details on the date such as the day of the week, day of the year, and week of the year. All of these would become properties of the new triple.

Figure 4.17 represents a main triple, while Figure 4.18 presents an example of its associated OWL-Time ontology triple that describes the event's time component. Notice that the date "2019-01-09" has been correctly expanded into a DateTimeDescription instance, and that the year, month, day, dayOfWeek, and dayOfYear are congruent.

Subject	Predicate	Object
http://myuri/DonaldTrump	http://myuri/respondedTo#1	"Romney's criticism"
http://myuri/respondedTo#1	http://myuri/rawSubject	"Trump"
http://myuri/respondedTo#1	http://myuri/confidenceScore	0.823358344545645
http://myuri/respondedTo#1	http://myuri/singletonPropertyOf	http://myuri/respondedTo
http://myuri/respondedTo#1	http://myuri/xsdSourceDate	2019-01-02
http://myuri/respondedTo#1	http://myuri/rawObject	"Romney's criticism"
http://myuri/respondedTo#1	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2006/time#Instant
http://myuri/respondedTo#1	http://myuri/tokenList	"Trump responded to Romney's criticism"
http://myuri/respondedTo#1	http://myuri/rawPredicate	"responded to"
http://myuri/respondedTo#1	http://www.w3.org/2006/time#inDateTime	http://myuri/temporalDescription9
http://myuri/respondedTo#1	http://myuri/hasSource	"https://www.theguardian.com/us-news/2019/jan/02/mitt..."
http://myuri/respondedTo#1	http://myuri/xsdDate	2019-01-09

Figure 4.17: RDF representation of an event using the Singleton Property

Subject	Predicate	Object
http://myuri/temporalDescription9	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2006/time#DateTimeDescription
http://myuri/temporalDescription9	http://www.w3.org/2006/time#timeZone	http://www.w3.org/2006/timezone-worldZTZ
http://myuri/temporalDescription9	http://www.w3.org/2006/time#unitType	http://www.w3.org/2006/time#unitDay
http://myuri/temporalDescription9	http://www.w3.org/2006/time#day	9
http://myuri/temporalDescription9	http://www.w3.org/2006/time#dayOfWeek	http://www.w3.org/2006/time#Wednesday
http://myuri/temporalDescription9	http://www.w3.org/2006/time#dayOfYear	9
http://myuri/temporalDescription9	http://www.w3.org/2006/time#week	2
http://myuri/temporalDescription9	http://www.w3.org/2006/time#month	1
http://myuri/temporalDescription9	http://www.w3.org/2006/time#year	2019

Figure 4.18: RDF representation of a temporal expression via a time:DateTimeDescription instance

4.2.8 SentimentEnricher and TweetSearcher

Once all news articles have been processed, and all pipeline steps have finalized, the SentimentEnricher component runs to associate sentiment score to primary triples. This component relies on the VirtuosoClient component to query all existing triples in the triplestore. Then, it searches for a tweet with similar content to the assertion by using the TweetSearcher component. The TweetSearcher component receives the triple's text and the news article publication date. With this information, it calculates a time window to restrict the search criteria, so that only tweets posted around the publication date of the triple's source news article are considered. Currently, such time window goes from one day before the news article publication date to two days past such date. TweetSearcher, in turn, invokes the TweetLookup component, which manages the Lucene Tweets Index created by the TweetIndexer component during the

data gathering process. Because the tweets' creation date fields have been indexed to support searching, the TweetLookup component can return a set of matches limited by the specified time window. TweetLookup creates a complex query: a phrase query for the triple's subject content, a phrase query for the triple's predicate, a regular query for the triple's object, and a filter query to account for the time window.

Figure 4.19 and 4.20 present an example of a match attempt between a triple and tweets. Figure 4.19 describes a main triple with date "2019-06-17". Figure 4.20 showcases a search query on the Tweets Index using the main triple's subject, predicate, object, and a time window based on the triple's date information.

Subject	Predicate	Object
http://myuri/MattHancock	http://myuri/backs#7	http://myuri/BorisJohnson
http://myuri/backs#7	http://myuri/sentimentScore	1.4
http://myuri/backs#7	http://myuri/hasTweetSource	"https://twitter.com/GdnPolitics/status/1140519963343507456"
http://myuri/backs#7	http://myuri/singletonPropertyOf	http://myuri/backs
http://myuri/backs#7	http://myuri/hasSource	"https://www.theguardian.com/politics/2019/jun/17/matt...
http://myuri/backs#7	http://myuri/rawSubject	"Matt Hancock"
http://myuri/backs#7	http://myuri/rawPredicate	"backs"
http://myuri/backs#7	http://myuri/rawObject	"Boris Johnson"
http://myuri/backs#7	http://myuri/confidenceScore	0.856785425120043
http://myuri/backs#7	http://myuri/tokenList	"Matt Hancock backs Boris Johnson"
http://myuri/backs#7	http://myuri/xsdSourceDate	2019-06-17

Figure 4.19: RDF representation of an event with sentiment data associated to it

TweetSearcher is responsible for selecting a match from the results returned by TweetLookup. SentimentEnricher takes the sentiment score and the URL of the tweet recovered from the Lucene result set and invokes VirtuosoClient to ingest extra triples to decorate the primary triple predicate. VirtuosoClient adds hasTweetSource and sentimentScore properties to the appropriate predicate. Notice that such properties are listed in Figure 4.19.

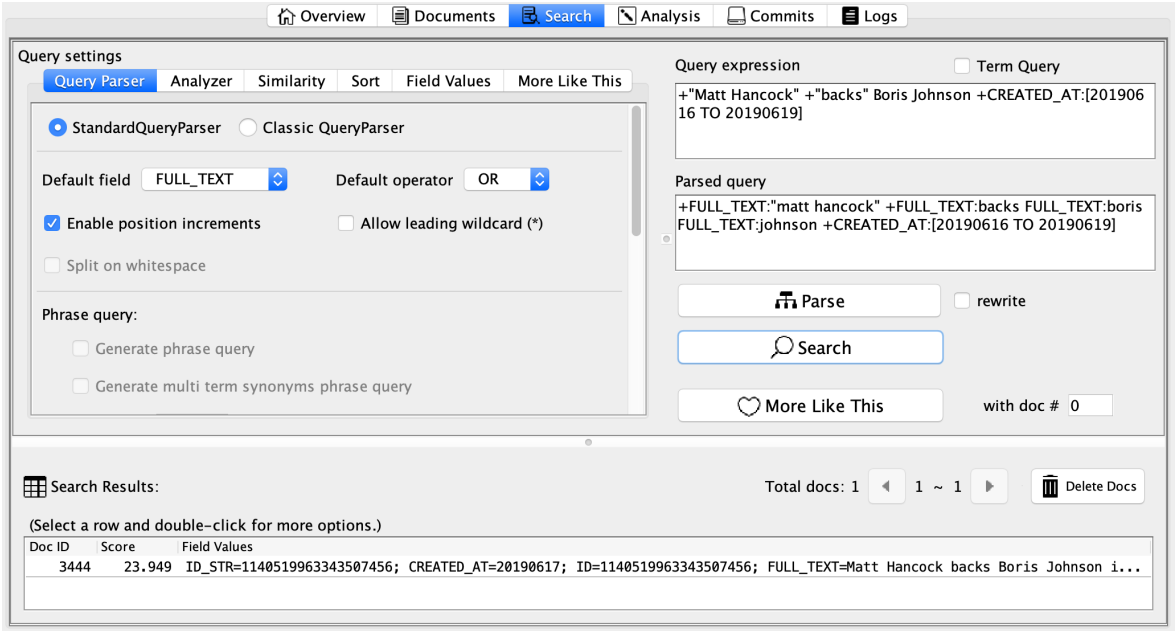


Figure 4.20: Result of a search over the Tweets Index by text and time window

4.3 Querying

NewsTextAnalyzer includes a lightweight web application to facilitate issuing a set queries to the knowledge graph. The web application was written is Servlet based and part of the NewsTextAnalyzer Java project. It runs on an Apache Tomcat[15] 9.0 HTTP web server. The main components of the Querying module include: LookupServlet, VirtuosoLookupClient, QueryServlet, and VirtuosoQueryClient.

The QueryServlet takes HTTP requests to the web application URL and serves the web content to render the HTML interface to the user. Once the user has completed the necessary form fields and submits, QueryServlet is responsible for extracting the query parameters and invoking the VirtuosoQueryClient to perform the appropriate query against the knowledge graph. VirtuosoQueryClient takes the query parameters and constructs a SPARQL query that is sent to the SPARQL endpoint of the local triple-store. Once QueryServlet obtains the results of the query from VirtuosoQueryClient, QueryServlet prepares the response page and sends it back to the user's browser.

[15] `http://tomcat.apache.org/`

The interface presented by QueryServlet allows the user to select a query type. Depending on the query type, the interface loads different form fields the user should fill before submitting a query. Currently, the querying interface supports five different query types: by person name, by predicate pattern, by resource type and country of birth, by associated subject, and by sentiment score range.

To help users prepare queries more easily, the interface supports lookups on some of the form fields. Such lookups result in Ajax requests sent to the LookupServlet. The LookupServlet inspects the requests, identifies the lookup type and the lookup parameters, and conducts the lookup operation through the VirtuosoLookupClient component. The VirtuosoLookupClient prepares SPARQL queries that are sent to the SPARQL endpoints of the DBpedia and DBpedia Live triplestores. This component takes care of merging the results from both sources to get a single set of results. Once VirtuosoLookupClient returns the results of the query, LookupServlet prepares an HTTP response with the query results in JSON format. The interface receives the response and updates the form appropriately.

The interface also allows the user to apply two constraints over queries. Users can restrict the query to only return events, that is, assertions that have a temporal expression associated to them as a result of the processing performed by the pipeline. In addition, users can also specify a date range, and in response, the system will only return events whose associated date lies within such range. Both of these filters are available for every query type.

4.3.1 Query by Person Name

This query type enables lookup by a person's name to obtain a specific person's resource name. When LookupServlet receives this type of lookup, it instructs the VirtuosoLookupClient to query DBpedia and recover resources of type dbyago:Person100007846 whose label in English matches a regular expression based on the person's name. The matches of the lookup are presented to the user, so it can select a specific person re-

source name.

When the user submits the form, the resource name of the person is passed as a parameter to the QueryServlet component, which in turn relies on VirtuosoQueryClient to consult the local triplestore for triples whose subject has been interlinked to the selected DBpedia person resource. In other words, it looks for triples whose subject has an owl:sameAs property and matches the selected resource. Figure 4.21 presents an example of the interface for this query type.

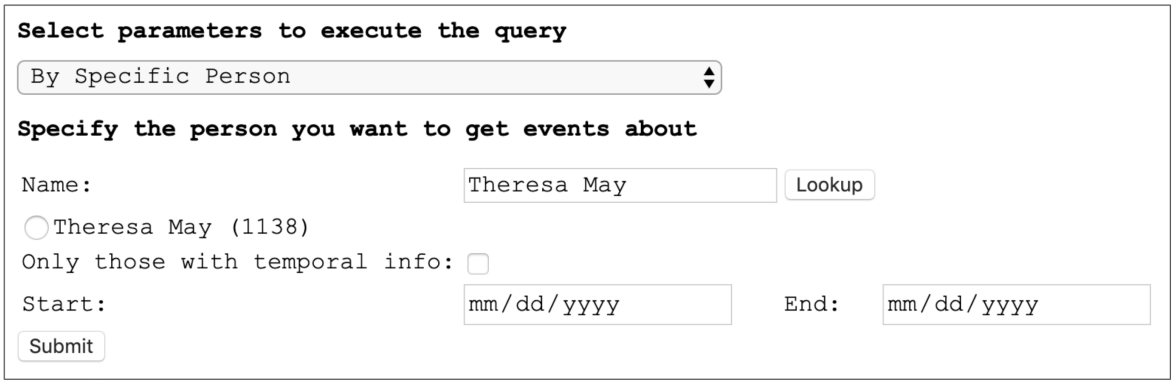


Figure 4.21: Example of the Query by Person Name UI

4.3.2 Query by Predicate Pattern

This is a simple query executed completely over the local triplestore. The user types a keyword of interest, and the QueryServlet and VirtuosoQueryClient components work together to look for triples whose predicate matches a regular expression based on the keyword and return the response. See Figure 4.22 for an example of this query type.

Figure 4.22: Example of the Query by Predicate Pattern UI

4.3.3 Query by Resource Type and Country Property

This query type presents two lookup options: resource type and country of birth. For the country of birth lookup, LookupServlet leverages VirtuosoLookupClient to query DBpedia for all resources of type dbo:Country, whose English labels match a regular expression based on the country name passed as parameter by the interface.

For the resource type lookup, the components look for Dbpedia-Yago types that have been used within the DBpedia knowledge graph, that are direct or indirect subclasses of the dbyago:Person100007846 type, and whose resource name matches a regular expression based on the type description, i.e., role, passed as parameter. See Figure 4.23 for an example.

After the user selects the country of birth and resource type, and submits the form, the QueryServlet receives the type name, and country resource name. Then the QueryServlet instructs VirtuosoQueryClient to recover triples whose subjects are resources of the type selected and have a dbo:birthPlace property that meets at least one of the following conditions:

- The object of the dbo:birthPlace property is a resource that in turn has a dbo:country property whose object is the country of birth resource name passed as parameter
- The object of the dbo:birthPlace property is the country of birth resource name passed as parameter

Figure 4.23: Example of the Query by Resource Type and Country Property UI

- The object of the dbo:birthPlace property has direct or indirect (transitive) dbo:isPartOf relationship to a resource that has a dbo:country property whose object is the country of birth resource name passed as parameter

4.3.4 Query by Subject Property

DBpedia does not completely map category information to resource types. Instead, DBpedia models much of that information through the dcterms:subject property. For example, there is not a resource type for the 2020 U.S. Presidential Election Candidates, but there is a resource named "`http://dbpedia.org/resource/Category:Candidates_in_the_2020_United_States_presidential_election`" that is linked to person resources through the aforementioned property.

To allow queries by category, the interface allows lookups by the dcterms:subject property. Upon receiving a keyword for the subject property, the LookupServlet and VirtuosoLookupClient query DBpedia for all the distinct values that have been used as objects of the dcterms:subject property by subjects that are resources of the dbyago:Person100007846 type. In addition, the categories have to match a regular expression based on the keyword passed as parameter. These lookup results are presented by the interface so the user can select one, see Figure 4.24.

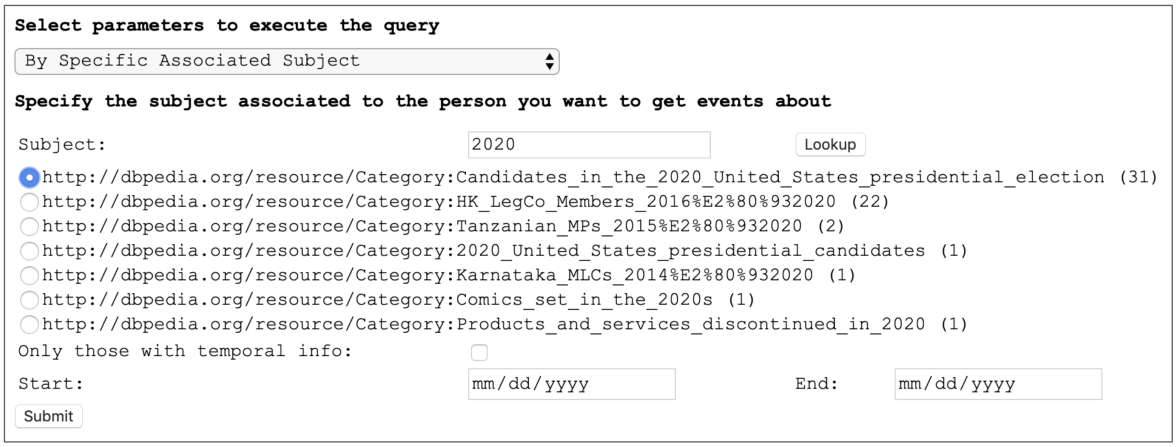


Figure 4.24: Example of the Query by Subject Property UI

When the user submits the form, QueryServlet and VirtuosoQueryClient obtain from DBpedia a list of resource records whose dcterms:subject property matches the value selected by the user. Then, VirtuosoQueryClient inspects the local knowledge graph for triples whose subjects are interlinked through the owl:sameAs property to

the ones in the list mentioned before.

4.3.5 Query by Sentiment Range

The last query is the simplest as it does not involve a lookup and the entire query is executed over the local triplestore.

The interface presents the user with an option to select an operator and a sentiment score threshold. When QueryServlet receives the query, it tasks VirtuosoQueryClient with finding triples that had an associated sentiment score, and whose score complies with the criteria selected by the user. See Figure 4.25 for an example of the interface for this query type.

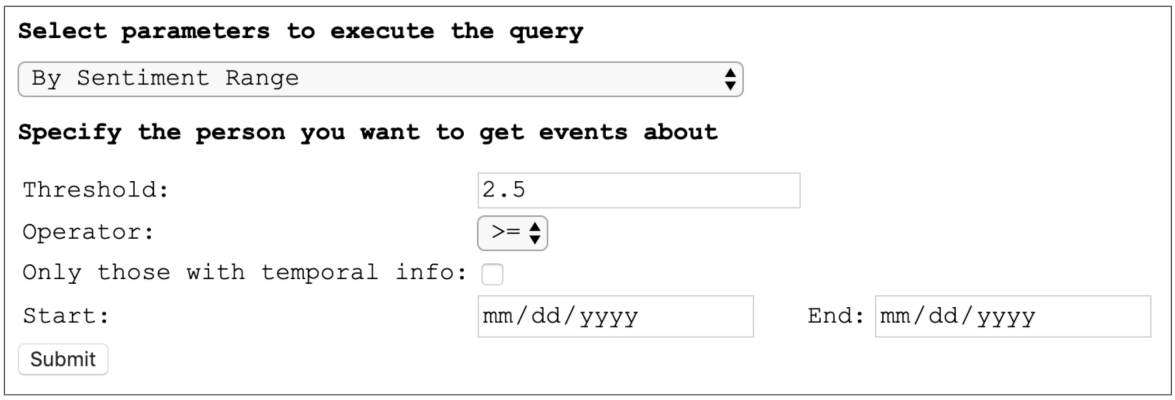
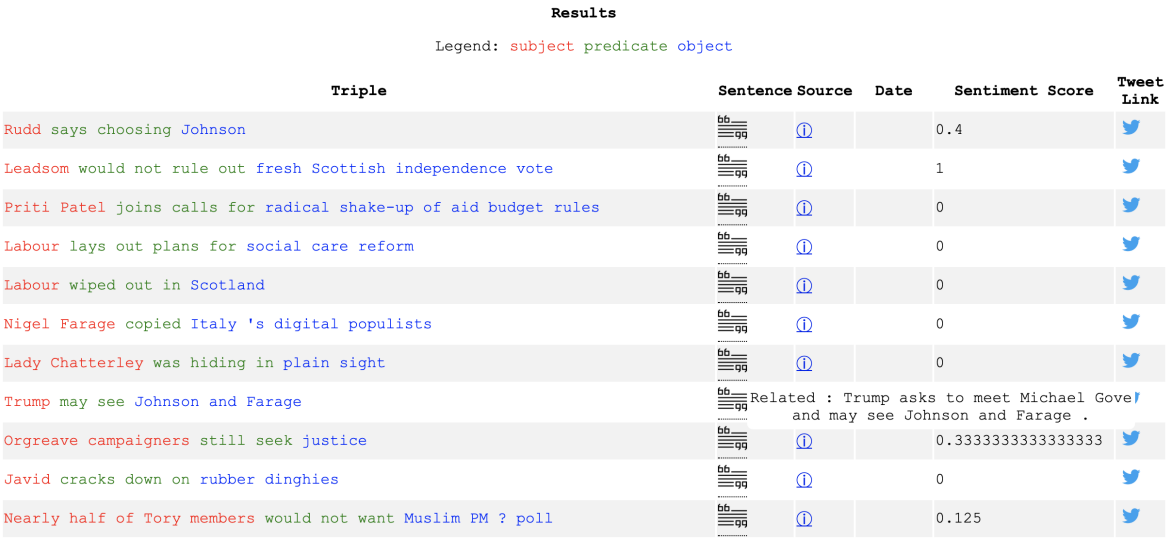


Figure 4.25: Example of the Query by Sentiment Range UI

4.3.6 Query Results

The results for all queries are lists of triples. The interface presents such lists using a table structure with the following columns: triple in natural language, with subject highlighted in red, predicate in green, and object in blue; an icon that when moused over shows the entire sentence from which the triple was extracted; an icon that when clicked takes the user to the source news article on The Guardian website; the date the pipeline attached to the event, in case one was detected; the sentiment score from

the associated tweet post, in case one was matched; and an icon that when clicked takes the user to the source tweet post by @GdnPolitics on Twitter. The output format is the same for all query types. See Figure 4.26 for an example of results to a query.



Results

Legend: subject predicate object

Triple	Sentence	Source	Date	Sentiment Score	Tweet Link
Rudd says choosing Johnson				0.4	
Leadsom would not rule out fresh Scottish independence vote				1	
Priti Patel joins calls for radical shake-up of aid budget rules				0	
Labour lays out plans for social care reform				0	
Labour wiped out in Scotland				0	
Nigel Farage copied Italy 's digital populists				0	
Lady Chatterley was hiding in plain sight				0	
Trump may see Johnson and Farage	Related : Trump asks to meet Michael Gove and may see Johnson and Farage .				
Orgreave campaigners still seek justice				0.3333333333333333	
Javid cracks down on rubber dinghies				0	
Nearly half of Tory members would not want Muslim PM ? poll				0.125	

Figure 4.26: Example of the Query Results UI

This chapter provided detailed information about NewsTextAnalyzer functionality and its components. The next chapter will assess the similarities and differences of NewsTextAnalyzer compared to similar systems.

Chapter 5

Evaluation

In this section, NewsTextAnalyzer is qualitatively compared to other systems that automatically build knowledge graphs such as EVIN [40], XLike [15], ECKG [44] and EventKG [46]. The criteria selected for comparison are: Data Source, Event Granularity and Extraction Methods, Entity Recognition, Coreference Resolution, Entity Linking, Relation Extraction, Relation Normalization, Event Classification, Complex Event Support and Knowledge Extracted. These criteria constitute the most common properties observed across different systems that automatically build knowledge graphs. In general, ECKG is the most advanced system due to its advanced NLP pipeline, while EventKG is relatively the simplest. As mentioned before, NewsTextAnalyzer is the only system that incorporates public sentiment reaction as an event property. For a summary of the comparison, see Figure 5.1.

	EVIN	XLike	ECKG	EventKG	NewsTextAnalyzer
Data Source	News articles	News articles	News articles	Other KGs and Wikipedia pages	News articles
Languages Supported	English	English, Spanish, Catalan, German, Slovene, Croatian, and Chinese	English, Spanish, Italian and Dutch	English, German, French, Russian and Portuguese	English
Event Granularity	Article level events	Fine-grain events	Fine-grain events	Fine-grain events from KGs but article level events from Wikipedia	Fine-grain events
Extraction Methods	Based on language models	Sophisticated NLP pipeline	State-of-the-art NLP pipeline	Hand crafted mapping rules, pattern matching, and link analysis	Lighweight NLP pipeline
Entity Recognition	NER via Stanford CoreNLP but doesn't use entity types	NER via FreeLing for all entity type classes	NER via 2 ad-hoc tools, one for general entity types, and one specialized in date entity type	Through link analysis	NER via Stanford CoreNLP for all entity types
Coreference Resolution	N/A	Simple, over named entities	Advanced, over named entities and events	N/A	Simple, over specific named entities
Entity Linking	N/A	N/A	Ad-hoc NED module based on DBpedia Spotlight	N/A	Simple NED module similar to DBpedia Spotlight
Relation Extraction	N/A	Advanced SRL via Treeler	Advanced SRL via MATE-tools	Via mapping and rules from other KGs; no extraction from descriptions on Wikipedia pages	Simple open information extraction via ReVerb
Relation Normalization	N/A	Advanced frame mapping via FrameNet and PredicateMatrix	Advanced frame mapping via FrameNet and PredicateMatrix	N/A	Minimal, just for the purpose of RDF conversion
Event Classification	Granular, via Wikipedia categories and WordNet event types	N/A	Granular, via ESO ontology	High level, topic based, via types of the entities involved	N/A
Complex Event Support	Only finds and surfaces related articles in terms of content	N/A	Can find and represent events that directly caused others	Uplifts subevents and chain of events defined in other KGs Finds and surfaces events that have similar participants	N/A
Knowledge Extracted	Minimal, at topic level	Participants, provenance, location, and time	Participants, provenance, location, time, and opinion	Participants, provenance, location, time, relation strength, and event popularity	Participants, provenance, and time
Public Sentiment Captured	N/A	N/A	N/A	N/A	Yes, from replies to posts made on Twitter

Figure 5.1: Summary of capabilities of EVIN, XLike, ECKG, EventKG, and NewsTextAnalyzer

5.1 Data Source

EVIN, XLike, ECKG and NewsTextAnalyzer process unstructured data, specifically news articles from publishers that cover a vast array of topics such as The New York Times, Bloomberg[1] and WikiNews[2]. EventKG's data sources are a combination of structured data, e.g., DBpedia, and semi-structured data, e.g., WCEP and Wikipedia Event Lists. Consequently, EVIN, XLike, ECKG and NewsTextAnalyzer can be applied to wide variety of corpus while EventKG is restricted to work with sources with very specific properties.

5.2 Event Granularity and Extraction Methods

EVIN does not employ NLP tools to extract information about events but instead generates a representation of an event theme from an entire news article through a language model. Thus, EVIN is incapable of capturing fine grain events at the sentence level.

Given that EventKG mostly works over structured or semi-structured data, it does not use an NLP pipeline; at most, it uses a mapping to normalize information from different knowledge bases, and patterns and rules to identify pieces of information from WCEP and Wikipedia Event Lists.

On the other hand, XLike, ECKG and NewsTextAnalyzer use NLP pipelines to extract information from text at the sentence level, so they are able to capture many more events than EVIN. In terms of the sophistication of the NLP pipeline of these systems, ECKG uses the largest number of NLP modules and their modules perform the deepest syntactic analysis, followed by XLike, and NewsTextAnalyzer. This means that ECKG is more likely to achieve a higher accuracy in event extraction than any other system at the cost of higher processing time cost per sentence.

[1] `https://www.bloomberg.com/`
[2] `https://en.wikinews.org/wiki/Main_Page`

5.3 Entity Recognition

EVIN utilizes Stanford CoreNLP to identify entity mentions in the news articles that it takes as input. Although Stanford CoreNLP can determine the entity classes, e.g., location, organization, it does not seem EVIN uses class information when is time to build the knowledge base.

For articles in English, XLike conducts NER by utilizing the FreeLing NLP[3] toolset. FreeLing is able to determine the entities' classes and XLike's advanced NLP modules use this information later when it extracts semantic frames to model the events mentioned in the articles' sentences.

ECKG uses ad-hoc NLP tools and classify entities in at least three types: "person", "location", and "organization". It is important to note that ECKG's architecture includes a module based on a tool called TimePro that was specifically designed as part of the TextPro[4] toolset to detect and extract temporal expressions. For normalization, ECKG uses a library called timenorm[5] in order to formalize the temporal expression to a well-formatted date [45].

EventKG does not perform any NER on text, thus it entirely depends on the labels or hints provided within its input data to identify entities. EventKG assimilates entities from other knowledge graphs by extracting resources acting as subjects and objects of instances of class dbo:Event [46]. Regarding WCEP and Wikipedia Event Lists, EventKG detects entities from the text containing the event description using predefined patterns. It examines the underlying HTML and considers tokens that link out to other Wikipedia pages and external websites as entity mentions [51, 52]. Links to the entity mentions are associated by Wikipedia editors when they create articles, or by regular users during the process of improving and existing article.

NewsTextAnalyzer's approach is similar to ECKG, but it uses a NER tool, Stanford CoreNLP, to perform both typical entity recognition and temporal expression

[3] `http://nlp.lsi.upc.edu/freeling/`
[4] `http://textpro.fbk.eu/`
[5] `https://github.com/clulab/timenorm`

identification, and uses Natty for temporal expression normalization. Thus, NewsTextAnalyzer provides greater entity identification capabilities than EventKG and EVIN, comparable capabilities to XLike, and less advanced capabilities than ECKG.

5.4 Coreference Resolution

As explained in Chapter 2, coreference resolution consists in finding and resolving references to entities mentioned within a document. Such references could use variations of the entity's name, or pronouns and pronoun phrases to refer to entities. By incorporating coreference resolution, systems can connect more assertions among each other, which in turn surfaces more relations between entities and could surface valuable information for complex queries.

EVIN represents an article as a set of features derived from the language model of the article, from the entities the article mentions, and from the article's date of publication. The article's features are considered a single node connected to others in a graph that represents the entire corpus. EVIN simplifies the graph by merging nodes that have very similar features but do not alter the structure of the graph significantly [40]. Because of this clustering-like approach that EVIN follows, no coreference resolution is performed.

On the other hand, XLike performs coreference resolution by detecting named entity aliases and repetitions of common nouns that allows it to aggregate semantic frames extracted from each sentence into a single graph that represents the entire document. It is interesting that XLike performs coreference resolution over semantic level representations, instead than over entity mentions.

In the case of ECKG, it actually conducts coreference resolution not only over entities, e.g., people, but also over events within the same article. It does the latter by matching lemmas of predicates or by estimating the WordNet similarity score of predicates, over all the sentences of the document. It is worth noting that the entity coreference resolution module developed by ECKG uses dependency parsers as input,

and thus, are more advanced than those used in XLike.

EventKG does not perform any coreference resolution within the short descriptions of the event included in WCEP and Wikipedia Event Lists.

NewsTextAnalyzer's coreference resolution follows a similar but simpler approach than XLike's with the Intralinker module. Hence, NewsTextAnalyzer's coreference resolution capabilities over text are better than EVIN and EventKG, slightly less advanced than XLike's and much less advanced than ECKG's.

5.5 Entity Linking

Linking entities to external representations also contributes to have denser knowledge graphs and resolve queries that might require information that is spread over many triplestores. Thus, the more entities a knowledge graph can correctly link, the better.

EVIN does not perform entity linking because it does not analyze sentences at a granular level. Instead, it uses a graph coarsening approach in which articles are represented by nodes with features based on statistics derived from the article's text and from the corpus, e.g., TF-IDF.

Padró et al. do not mention any entity resolution task as part of the XLike pipeline described in [15], so it is assumed that none is performed.

In comparison, ECKG developed an ad-hoc named entity disambiguation module based on DBpedia Spotlight that takes an entity mention and returns a candidate resource from DBpedia to link to. Their module has two modes of operation "disambiguate" and "candidates". In the former, it only returns the best resource to link to, while in the latter, it returns a ranked list of candidates with a measure of their similarity scores to the input text. An additional capability of ECKG entity linking module is that it can use the entity type, e.g., "location", to return only relevant entities of the right type, improving the accuracy of the linking process [45].

EventKG does not perform entity resolution. The source knowledge graphs it uses already have entities linked. Regarding the event description included in WCEP and Wikipedia Event Lists, EventKG only considers mentions with links as entities, so the entities are already resolved.

Given that NewsTextAnalyzer created the Person Resources Index to perform entity resolution, its entity linking capabilities are more extensive than EVIN, XLike and EventKG, but not as advanced as ECKG because it only resolves entities identified as people, while ECKG can resolve many more entity types.

5.6 Relation Extraction

EVIN does not perform granular relation extraction due to the same reason it does not conduct entity resolution.

XLike does not perform traditional relation extraction but instead uses SRL. It leverages an ad-hoc library called Treeler[6] developed for the XLike project. ECKG also uses SRL but the tool used in that case is called MATE-tools[7]. As explained before, SRL allows one to identify the semantic roles that some words play in a sentence. By identifying the roles that exist in the sentence, it is possible to better assess the kind of relation that is being described. For example, if the roles "buyer" and "goods" were identified in a sentence, the relation taking place is probably a purchase. Moreover, with additional processing on the semantic roles, it would be possible to reconcile sentences that are different at a lexical level but similar at the semantic level, e.g., "Google bought Nest" and "Nest was acquired by Google". Notice, that for those two example sentences, relation extraction would provide 2 different predicates, i.e., "bought" and "was acquired by", and some kind of lexicon would be required to try to perform a similar reconciliation. In addition, because entities would likely be given semantic roles, it is relatively straightforward to assign the right predicate to link entities to the assertion, e.g., if "Google" is labeled as "buyer" in the previous example,

[6] `http://devel.cpl.upc.edu/treeler/`
[7] `https://code.google.com/archive/p/mate-tools/`

then that is an appropriate predicate to link the entity to the resource representation of the transaction.

EventKG derives relations between entities from what it observes in the knowledge graphs used as sources. Using a set of rules, EventKG also maps specific properties from other knowledge graphs to their own property naming convention in order to consolidate properties used by different knowledge graphs but that mean the same, e.g., location. However, EventKG does not perform relation extraction nor semantic role labeling on the event description text of WCEP or Wikipedia Event Lists.

NewsTextAnalyzer does not use semantic role labeling because it requires a dependency parse and constituency parse trees which are expensive to derive, especially for long sentences. Instead, NewsTextAnalyzer uses ReVerb, an open information extractor that returns triples with text fragments as subject, predicate and object. Thus, the relation extraction capabilities of NewsTextAnalyzer are above EVIN and EventKG, but below XLike and ECKG because SRL provides much more information about entities in a sentence that can be used to better model events.

5.7 Relation Normalization

Normalizing relations allow the knowledge graph to introduce more structure and use the same denomination for lexically different predicates that have almost the same meaning. This in turn simplifies creating queries that look for specific kind of facts.

In the case of EVIN, since it does not perform relation extraction, there is no relation to normalize.

XLike and ECKG are very similar in terms of relation normalization. They use frame extraction to verify if the semantic roles and other syntactic features, such as the dependency parse, match a well-established frame from a lexical database, such as FrameNet or PredicateMatrix. Through the use of these lexicons, XLike and ECKG are able to normalize predicates that are lexically different but semantically equivalent.

EventKG does not need to perform relation normalization because of the way it obtains entity relations.

NewsTextAnalyzer does not normalize predicates, nor reorganize the triple structure to detect if there are others that express the same meaning. However, it does track and count predicates that have the same tokens in order to assign the right instance identifier according to the Singleton Property method. Hence, NewsTextAnalyzer's relation normalization capabilities are at the par of EVIN and EventKG, well below XLike and ECKG.

5.8 Event Classification

Like relation normalization, event classification organizes facts making them easier to retrieve in the case of queries that ask for types or subtypes of events.

EVIN performs event classification, although at a high level. EVIN constructs a language model for each news article and for each Wikipedia category based on all Wikipedia pages under such category. Then, EVIN selects the most appropriate category for the news article, based on the similarities of the language models. EVIN follows an additional step to map the Wikipedia category to the lowest possible event type in WordNet based on similarity of category and event type head words.

It does not seem XLike performs any event classification, as no mention of this task is described by Padró et al. in [15].

ECKG uses the Event and Implied Situation Ontology (ESO)[8] to classify events. ESO is a hierarchy composed of 63 different event classes. ECKG can assign the right class to the assertion by examining its frame and by following a set of rules. Some of the classes include "BeingAtAPlace", "Selling", and "Investing"; the last two being

[8] `https://github.com/newsreader/eso-and-ceo`

children classes of "FinancialTransaction".

EventKG also classifies events but instead of classifying them in terms of the action taking place, EventKG classifies them by topic or category. Some of the categories that EventKG uses include "Culture", "Sports" and "Disasters" [52]. EventKG performs the classification by deriving features from the resource types of the resolved entities mentioned in the event. For example, if an entity mentioned in an event is of type dbo:Politician, then it is likely the right category for the event is "Politics". EventKG trains an SVM classifier to use these features as inputs and predict the right class.

NewsTextAnalyzer does not classify events. Thus, it is at par of XLike, and well below EVIN, ECKG, and EventKG.

5.9 Complex Event Support

Complex event support refers to the knowledge graph capability of linking events in a chain, e.g., one event follows another, or identifying subevents of an event, e.g., specific football matches within the FIFA World Cup Finals.

Because EVIN works at the article level and not at the sentence level, it assumes that each article describes a core event. Following such assumption, EVIN links related articles in a chain in chronological order according to the article's publication date. Having that in mind, EVIN is able to surface the most relevant subsequent article to one given as input.

It does not seem XLike can detect chain of events from text, as there is no mention of a task that decorates the knowledge graph with properties for such purpose in [15].

ECKG has a special Causal Relation Extractor module that can detect lexical cues such as "as a result" or "due to" connecting two events within the same sentence. This module allows ECKG to derive causality assertions and model them in its knowledge graph.

EventKG does not interlink events but it can find related ones by using a set of heuristics. Given an event as a reference point, it returns a list with the events that occurred in a ten-year window around the reference event date, and that have at least a predefined minimum number of entities in common. It then sorts the list by number of common entities in descending order [52]. In addition, EventKG can represent a chain of events that it captures from other knowledge graphs through properties such as dbo:previousEvent and dbo:nextEvent. However, it does not seem that EventKG can infer such a chain of events from text [46].

Currently, NewsTextAnalyzer cannot detect chain of events from texts. Thus, it is at par of XLike, and well below EVIN, ECKG, and EventKG.

5.10 Knowledge Extracted

The knowledge extracted category refers to the information that the system can uplift from the data source and persists in a triplestore for future querying.

It does not appear that EVIN performs any work in modeling the events it captures in a triple format. At most, EVIN saves the entities related to an article in its knowledge base. Because for EVIN, the entire article is the unit of knowledge, provenance information is captured by default.

XLike captures event participants, time information, location and provenance information from the news articles it takes as input.

Similarly, in addition to capturing the same information as XLike, ECKG is able to detect if an assertion is a fact or not. Moreover, ECKG also detects the opinion expressed by a text, identifying who expresses the opinion, the recipient of the opinion, and whether the opinion is positive or negative. It is important to note that this is different from sentiment gathered from the public, because the opinion captured from sentences refers to the point of view reported by the news that a particular individual,

possibly someone well-known, expressed as a statement, while public sentiment is the reaction of a group of regular people.

EventKG can capture participants, time information, location, provenance information, and estimate the relationship strength to participants and event popularity.

NewsTextAnalyzer can also capture participant, temporal and provenance information. At the moment, NewsTextAnalyzer does not capture location information. Thus, in terms of the breadth of the knowledge extracted, NewsTextAnalyzer's coverage is greater than EVIN's and relatively at par of XLike's, ECKG's and EventKG's.

5.11 Public Sentiment Captured

Public sentiment captured refers to the ability of the system to enrich the knowledge graph by incorporating sentiment as another component of the event representation.

Neither EVIN, XLike, ECKG or EventKG capture public sentiment. In contrast, NewsTextAnalyzer captures public sentiment from Twitter, and associates such sentiment to the right events via the Tweet Index component.

In this chapter, NewsTextAnalyzer capabilities were compared to similar systems. What sets NewsTextAnalyzer apart is its capabilities to quickly process news articles with a lightweight NLP pipeline that performs open information extraction, and does not require high level syntactic analysis, such as constituency parse. In addition, NewsTextAnalyzer is the only system that retrieves public reaction from social media and associates such sentiment information to events captured from the news. The next chapter will list areas in which NewsTextAnalyzer can be improved as a result of the evaluation conducted.

Chapter 6

Future Work

This chapter examines the most important pieces of work that could be conducted in the future with the goal of improving NewsTextAnalyzer in terms of its coverage on assertions and events, its accuracy and its performance. Although all of the following improvements would make a significant difference in the quality of the information captured by the system, extending entity resolution capabilities, adding a module to distinguish facts from opinions, and incorporating more social media data sources have higher priority among the rest of ideas.

6.1 Extended Entity Resolution

At the moment, NewsTextAnalyzer recognizes entities of type "organization" and "location", but in both cases, they are not disambiguated against DBpedia. Creating additional Lucene DBpedia Resource indexes for this purpose should be straightforward, and the resulting triples that express the linkage of these entities to their DBpedia representation would support more complex queries. In addition, once NewsTextAnalyzer can disambiguate event locations, it should persist the locations that were correctly resolved by adding triples to associate the locations to the event at the RDF level. That would allow to query events by their location, or by a property derived from their location, such as their country.

6.2 Fact and Opinion Checking

The ReVerb extractor does not make a distinction between sentences that state a fact, sentences that include a quote or sentences that express opinion. For example, a news article[1] from The Guardian included a sentence that said, "But that hasn't stopped Trump, who memorably fueled the conspiracy theory that Barack Obama was born in Kenya". Such sentence was processed by the pipeline, and the triple ("Barack Obama", "was born in", "Kenya") was produced by ReVerb and ingested into the triplestore. This assertion is false and even contradicts other triples extracted by ReVerb such as ("Barack Obama", "was born in", "the United States of America"). In addition, the confidence score provided by ReVerb for opinion base triples can be high. In the example mentioned before, the confidence score calculated by ReVerb was 0.87; thus, confidence score cannot be used to filter out these misleading extractions. Although one could envision resolving these assertion conflicts by following what the majority of them indicate, this method would not work in cases in which a false assertion has caused a scandal and has been picked up and analyzed widely by the media. The pipeline extracted fourteen assertions indicating Barack Obama was born in Kenya, five assertions indicating he was not born in the U.S., and only three assertions indicating he was born in the U.S.

Given the limitations of the ReVerb extractor in this regard, the NewsTextAnalyzer system would need to incorporate additional steps to address this problem. In order to have a knowledge graph that can provide useful and truthful information to users, it would be necessary to gather signals to quantify the level of confidence in the assertion. One simple but extreme method would be to create a blacklist of verbs such as "say" or "believe", and discard triples that come from sentences that have tokens whose lemmas overlap with the blacklist. Another approach would be to instead of immediately filtering out triples whose sentences look suspicious, use search engines to check their validity. For example, for the false triple ("Barack Obama", "was born in", "Kenya"), Google search results include words such as "conspiracy", "discredited claim", and "outrageous claims", which could give NewsTextAnalyzer sufficient evi-

[1] https://www.theguardian.com/us-news/2016/jan/12/where-was-ted-cruz-born-citizenship-presidential-debate

dence to not persist the primary triple. However, this might not work for assertions that are just misleading and not completely false. Perhaps, the best course of action is to emulate ECKG, and construct a classifier to detect if the way the sentence is phrased indicates a fact or not, and only persist facts. A benefit of this approach is that according to Agerri et al. [45] the classifier only requires tokens and POS tags, so it would not impose a large additional runtime penalty to the pipeline.

6.3 Enhanced Intralinking

NewsTextAnalyzer intralinks name mentions of people to their full names whenever the latter appeared before the former in the article. However, NewsTextAnalyzer only does this when the subject of the extraction is composed entirely of entities of type "person". In politics, it is common to refer to people by their title followed by their last name, e.g., "President Trump". In such cases, intralinking does not occur. Additional work could be carried out to evaluate the head words and POS tags of a subject in order to decide if such subject should be intralinked or not. For example, "President Trump" should be intralinked to "Donald Trump" but "President Trump's son-in-law" or "The proposal of President Trump" should not be. The last two phrases contain possessives and prepositional phrases, but there are many other cases in which a subject includes NER tags of type "person" but the core of the subject is not such person. Finding a rule-based approach that can work for the majority of those cases and does not rely on dependency parse trees could be challenging.

6.4 Predicate Normalization

Currently NewsTextAnalyzer detects and persists predicates literally without attempting to normalize them in any way. Thus, the knowledge graph includes primary triples with predicates such as "resigned as", "resigned abruptly as", and "resigned his position as". These predicates basically describe the same event but because they are different lexically, NewsTextAnalyzer would not link them to the same Singleton Prop-

erty parent predicate. The evaluation of predicate head words and POS tags could help reconcile predicates as the ones presented before. It is unclear if the use of lemmas would be enough to reconcile predicates that are lexically different but have the same meaning, e.g., reconciling "quit his position as" with "left his job as". Normalizing predicates would allow categorization of assertions and events, as well as more easily constructing SPARQL queries that depend less on filters based on regular expressions, e.g., finding all the "resignation" events even if they do not use the word "resignation" in the predicate.

6.5 Parallel Computing Support

The current implementation of the pipeline is single-threaded. The corpus collected by this project consists of more than 216,500 politics news articles that in total represent over 10,880,000 sentences. Processing the entire corpus takes approximately 12 hours on a 2.6 GHz Intel Core i7, 16GB RAM computer, with the task of obtaining NER tags through Stanford CoreNLP taking approximately 50% of the runtime. Thus, NewsTextAnalyzer could benefit significantly from parallel computing. This would demand verifying that the libraries used by the project are thread safe and developing infrastructure to allow multiple instances of the pipeline operate over different news articles. Moreover, the multiple instances of the pipeline would need to share information among themselves for the purpose of intralinking. For example, a politician might have been observed as the subject of a triple in an article, but as the object of another triple in a different article; hence, in order to link objects to subjects correctly, the system needs a single view of all subjects observed by all pipeline instances. Incorporating other programming languages and frameworks, e.g., Scala[2] and Akka[3], could reduce the amount of work required to make the pipeline work in a parallel and collaborative fashion but would also require research in integrating modules written in different languages, porting libraries, etc.

[2] `https://www.scala-lang.org/`
[3] `https://akka.io/`

6.6 Additional Social Media Data

The limited access to Twitter data puts a constraint on the system, which currently only linked sentiment to approximately 200 extractions captured from news articles. Given that is not possible to obtain tweet replies that go beyond seven days old, the TweetsCollector module should be enhanced to not only collect social media data from @GdnPolitics, but also from many other news sources. In this way, the chance of matching a tweet post and an extraction from news articles increases. As explained in Chapter 3, this approach would require a much more sophisticated matching process that the one NewsTextAnalyzer uses today. One idea is to use head words and lemmas extracted from the triple's subject, predicate and objects to perform the matching. Another idea, inspired by EVIN, would be to use language models based on the head words of the triple and calculate their similarity to language models generated from the tweet posts within a reasonable time window. A more ambitious approach that involves a higher risk of not achieving good results, would be to use the YouTube APIs to pull captions of videos uploaded by political publications, take the captions through a similar NLP pipeline to extract assertions, check with the knowledge graph if one event from news article matches an assertion from the captions, and if so, parse the video comments to extract sentiment. A slight variation of these ideas would be to send search queries to Twitter and YouTube with the main content of the triple, take the most popular results in terms of tweet posts or video uploads, and extract sentiment from their comments. In any case, as mentioned before, determining if the post or video is relevant and encompasses the assertion is the most challenging obstacle.

6.7 Complex Event Representation

NewsTextAnalyzer does not capture subevents or chain of events. However, such structures already exist in some news articles. For example, there are news articles from The Guardian that present a timeline of events surrounding the Snowden scandal [4]. Many of this type of articles exist but the pipeline cannot detect that they provide a summary of events. Taking the enhanced version of KnowItAll presented in [29] as

[4] `https://www.theguardian.com/world/2013/jun/23/edward-snowden-nsa-files-timeline`

well as EventKG, it might be possible to create a module that uses patterns to identify articles with timelines, uses a secondary NLP pipeline to ingest the events in case they have not been captured before, and links them in a chain.

Another enhancement that could support surfacing chain of events consists in distinguishing instant from interval events during the knowledge construction process. At the moment, NewsTextAnalyzer does not search for cues that would hint the presence of an internal event, such as the "from" and "to" tokens. If the pipeline could recognize these cases, the events' beginning point in time and duration could be described using other Time Ontology classes such as ProperInterval and DurationDescription [57]. By correctly representing interval events, it would be possible to create queries that ask for events that are consecutive, i.e., one ends when the other starts, events that overlap, or events that completely encompass others. By having interval information correctly modeled, it might be possible to infer that some events are subevents of others.

This chapter covered the areas in which NewsTextAnalyzer could be improved. The next chapter will present the conclusions of the project, including implementation findings.

Chapter 7

Conclusions

This chapter reviews challenges and findings that arose throughout the project and presents the conclusions.

7.1 Challenges and Findings

Different challenges surfaced during the design, implementation and evaluation of the results of NewsTextAnalyzer. This subsection presents the most important ones.

7.1.1 Relevant Sentence Classification Effects

To decrease the number of sentences that go through the entire pipeline and speed up the process, a simple Classifier component was initially included as the first pipeline step. This simple Classifier used a list of interesting lemmas to filter out sentences received from the PipelineManager. The Classifier component analyzed the sentence's lemmas with the Stanford CoreNLP library and discarded sentences that did not have any lemma in common with the list of interesting lemmas. This prevented those sentences from going through the remaining pipeline steps, reducing corpus processing run time. The list of lemmas was generated with the help of WordNet and consisted of 349 lemmas derived from 32 handpicked verbs that are commonly used within the

politics context, e.g., resign.

Although the Classifier component helped speed up the process by skipping approximately 30% of sentences that were unlikely to be interesting for the purpose of this research project, it also produced unintended consequences. For instance, the Classifier filtered out some sentences that contained politicians' full names. Consequently, the Referencer component did not captured those names, which prevented the IntraLinker component from reconciling last name mentions to the missing full names. Although the Classifier was removed from the final pipeline, it might be necessary to adapt it in a different configuration if more advanced NLP techniques are introduced. This reflects a constant situation encountered throughout the project: the tradeoff of more accurate information extractions versus its impact on runtime.

7.1.2 Relation Extraction Challenges

To identify relationships and extract triples from sentences, the project first attempted to follow a simplified version of the approach depicted in [25] but without the automatic generation of patterns described in [62] and implemented in [25] and [24]. The patterns were manually created and were based on tokens and NER tags. A subset of the patterns created allowed to identify resignation events, i.e., when a person resigns to a position or office. Some of those patterns are shown below (tokens in lowercase, NER tags in uppercase):

- ".*PERSON, who resigned as TITLE.*"
- ".*resignation of TITLE PERSON.*"
- ".*PERSON resigned as TITLE.*"
- ".*TITLE PERSON resigns.*"
- ".*PERSON 's resignation as TITLE.*"
- ".*resignations, including .* TITLE, PERSON.*"

Few occurrences of this relations were found within the corpus, but the quality and precision of those relations were high; in other words, the pattern extracted resignation events that were accurate but found few of them. If this approach were to be followed by NewsTextAnalyzer, every relation of interest would demand creating patterns to specifically detect them, which would take considerable amount of time. Moreover, it is difficult to anticipate all the different ways in which a specific event could be expressed; for complex relationships in which the arguments of the relationship are many words apart, this becomes even more difficult. Thus, it is likely that this approach would find just a small subset of events within the corpus because the list of patterns is not comprehensive. Lastly, this approach assumes that the arguments of the relationship of interest can be recognized as entities by a NER tagger, such as Stanford CoreNLP. However, a significant number of events would not meet this criterion. For example, if we were interested in finding events in which a person gives a speech, the object of the relationship would not be marked by the tagger. In this case, more patterns that rely on words would be necessary in order to describe all the different ways in which "speech" could be expressed in a sentence.

The project also considered following the bootstrapping approach described and implemented in [62] and [24]. Such approach would allow circumventing the task of creating patterns by hand. The approach proposes providing the system with a starting set of seed tuples that are known to have the relationship of interest, and then finding sentences where these tuples are present in a large corpus. Then, through the use of heuristics, the system can extract new patterns about the relationship of interest from the matching sentences. The new patterns would become seed tuples for the next iteration of the process. To test this approach, the following list of seed tuples was used to find patterns for the "government position" relationship:

- ("Boris Johnson", "Secretary of State")
- ("David Davis", "Secretary of State")
- ("Al Franken", "Senator")
- ("Jeremy Heywood", "Secretary")

- ("Tracey Crouch", "Minister")

Unfortunately, the corpus did not have many matches for these tuples, so few patterns were identified. Other tuples were tested too but returned similar results. For this approach to work effectively, the relation of interest for the seed tuples need to be expressed in different ways over many sentences of the corpus. That allows the system to find a wide variety of distinct patterns and increase coverage. Given that the corpus used in this project is made of news articles from a single source, The Guardian, it is possible that most relations between the seed tuples were expressed just a few times and using very similar syntactic constructs. In addition, the size of the corpus is more than 100 times smaller than the one used originally in [24], and in that project the system only looked for one relation of interest. To capture many relations of interest, one would need a much bigger corpus. These factors explain the challenges of getting results with this approach.

Because of these difficulties, the idea of using an ad-hoc extractor was abandoned, and the project incorporated a pre-built open information extractor, ReVerb. As mentioned before, open information extraction does not require handmade patterns and can scale and retrieve a large number of relations. As with the previous point, the tradeoff made with this decision involved sacrificing accuracy for reach and scalability.

7.1.3 DBpedia and DBpedia Live Inconsistencies

Information in DBpedia and DBpedia Live is not modeled consistently among both knowledge bases, or even among resources within them. Common inconsistencies in DBpedia and DBpedia Live include resources of similar characteristics not being associated the same type. For example, Elizabeth Warren and Kamala Harris are both current U.S. Senators but the latter is not associated to any DBpedia-Yago Senator class while the former is associated with three of them. Moreover, both DBpedia and DBpedia Live also include errors that could end up affecting the resolution on queries over resources linked to them. For example, if one searches for resources of type dbo:Country and whose resource name matches a regular expression based on "United_States", one would find resources such as `http://dbpedia.org/resource/1990s_United_States_`

`boom` or `http://dbpedia.org/resource/Ambassador_of_Iceland_to_the_United_States`, which are clearly not countries, see Figure 4.23. These errors could in turn affect queries that look to match country by name, for example, or interfere in more complex queries that use the result of a subquery about countries to perform additional processing. The project opted to provide lookups on the UI to mitigate this type of issues, since users would have no problem in selecting the right value over the spurious ones from the result set returned by the lookup.

7.1.4 ReVerb Extractor Limitations

Another limitation of the ReVerb extractor is its inability to capture non-verbal relations, such as those expressed by noun phrases. For example, from the noun phrase "Senator Elizabeth Warren", ReVerb will not generate the triple ("Elizabeth Warren", "is", "Senator"). Consequently, a number of simple but important triples will not be captured by the extractor. Granted, this kind of expressions are more likely to produce assertions than events, but they are still important and could help with entity disambiguation.

Lastly, occasionally ReVerb fails to completely extract the core of a triple's object. For example, when given the sentence "Trump also authorized his attorney general, William Barr, to declassify any intelligence surrounding what prompted the Russia inquiry", ReVerb generates the triple ("Donald Trump", "also authorized", "his attorney general") which is not that informative. This again reemphasizes the tradeoff between extraction accuracy and runtime described before.

7.2 Conclusions

This project introduced NewsTextAnalyzer, a system that automatically builds a knowledge graph by extracting assertions and events from unstructured data, i.e., politics news articles, and enriches the graph with public sentiment recovered from Twitter. The resulting knowledge graph consists of more than 460,200 assertions, more than 53,400 events, more than 15,600 entities linked to DBpedia, and more than 200 asser-

tions enriched with sentiment data. The system also provides a querying interface that can be used to answer event related questions and leverage the information the system extracted from the news. NewsTextAnalyzer's capability to integrate sentiment data makes it unique among this class of knowledge extraction systems.

Through NewsTextAnalyzer, the project designed and implemented a lightweight NLP pipeline to uplift information from a large corpus composed of more than 216,500 articles and 10,000,000 sentences in under 12 hours running on a single thread on a standard personal computer. Such pipeline solved the first research question presented in Chapter 1. Moreover, through the use of the Singleton Property, and the Time Ontology, NewsTextAnalyzer can represent events in a way that facilitates querying as demonstrated by its Querying UI. This capability answered the second research question of this project. Lastly, by leveraging particularities of the posting style of publishers and through the use of a Lucene index, NewsTextAnalyzer is able to match assertions to post made on Twitter, and correctly associate sentiment data to particular events, successfully resolving the last research question of the project.

NewsTextAnalyzer could serve as a foundation for a more elaborate system that can track complex events, e.g., subevents, chains of events, by implementing ideas proposed in Chapter 6. In any case, NewsTextAnalyzer, if provided with more data, could be used to support political science researchers interested in understanding political events and the reaction that people had to them, which would be useful in scenarios such as political campaign planning, evaluation public support behind proposed legislation, among many others.

Bibliography

[1] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[2] C. Zhai and S. Massung, *Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining*. New York, NY, USA: Association for Computing Machinery and Morgan & Claypool, 2016.

[3] W. A. Woods, L. A. Bookman, A. Houston, R. J. Kuhns, P. Martin, and S. Green, "Linguistic knowledge can improve information retrieval," in *Sixth Applied Natural Language Processing Conference*, (Seattle, Washington, USA), pp. 262–267, Association for Computational Linguistics, Apr. 2000.

[4] J. Beall, "The weaknesses of full-text searching," *The Journal of Academic Librarianship*, vol. 34, no. 5, pp. 438–444, 2008.

[5] J. Callan, "Using knowledge resources to improve information retrieval." `http://www.cs.cmu.edu/~callan/Projects/IIS-1422676/`.

[6] L. Ehrlinger and W. Wöß, "Towards a definition of knowledge graphs," 09 2016.

[7] P. Hitzler, M. Krtzsch, and S. Rudolph, *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC, 1st ed., 2009.

[8] R. W. Group, "Resource description framework (rdf)." `https://www.w3.org/RDF/`.

[9] W3C, "Rdf 1.1 turtle, terse rdf triple language." `https://www.w3.org/TR/turtle/`.

[10] W3C, "Sparql 1.1 query language." `https://www.w3.org/TR/sparql11-query/`.

[11] W3C, "Linked data." `https://www.w3.org/standards/semanticweb/data`.

[12] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: A collaboratively created graph database for structuring human knowledge," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, (New York, NY, USA), pp. 1247–1250, ACM, 2008.

[13] S. S. Sahoo, W. Halb, S. Hellmann, K. Idehen, T. Thibodeau Jr, S. Auer, J. Sequeda, and A. Ezzat, "A survey of current approaches for mapping of relational databases to rdf," 2009.

[14] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, (Berlin, Heidelberg), pp. 722–735, Springer-Verlag, 2007.

[15] L. Padró, Ž. Agić, X. Carreras, B. Fortuna, E. García-Cuesta, Z. Li, T. Štajner, and M. Tadić, "Language processing infrastructure in the XLike project," in *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*, (Reykjavik, Iceland), pp. 3811–3816, European Languages Resources Association (ELRA), May 2014.

[16] D. Jurafsky and J. H. Martin, *Speech and Language Processing (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2009.

[17] V. Yadav and S. Bethard, "A survey on recent advances in named entity recognition from deep learning models," in *Proceedings of the 27th International Conference on Computational Linguistics*, (Santa Fe, New Mexico, USA), pp. 2145–2158, Association for Computational Linguistics, Aug. 2018.

[18] N. Kambhatla, "Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations," in *Proceedings of the ACL 2004 on Interactive Poster and Demonstration Sessions*, ACLdemo '04, (Stroudsburg, PA, USA), Association for Computational Linguistics, 2004.

[19] S. Strassel, A. Mitchell, and S. Huang, "Multilingual resources for entity extraction," in *Proceedings of the ACL 2003 Workshop on Multilingual and Mixed-language Named Entity Recognition - Volume 15*, MultiNER '03, (Stroudsburg, PA, USA), pp. 49–56, Association for Computational Linguistics, 2003.

[20] Z. GuoDong, S. Jian, Z. Jie, and Z. Min, "Exploring various knowledge in relation extraction," in *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, (Stroudsburg, PA, USA), pp. 427–434, Association for Computational Linguistics, 2005.

[21] D. P. T. Nguyen, Y. Matsuo, and M. Ishizuka, "Relation extraction from wikipedia using subtree mining," in *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 2*, AAAI'07, pp. 1414–1420, AAAI Press, 2007.

[22] Y. S. Chan and D. Roth, "Exploiting syntactico-semantic structures for relation extraction," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, (Stroudsburg, PA, USA), pp. 551–560, Association for Computational Linguistics, 2011.

[23] S. Pawar, G. K. Palshikar, and P. Bhattacharyya, "Relation extraction : A survey," *ArXiv*, vol. abs/1712.05191, 2017.

[24] S. Brin, "Extracting patterns and relations from the world wide web," in *Selected Papers from the International Workshop on The World Wide Web and Databases*, WebDB '98, (London, UK, UK), pp. 172–183, Springer-Verlag, 1999.

[25] E. Agichtein and L. Gravano, "Snowball: Extracting relations from large plain-text collections," in *Proceedings of the Fifth ACM Conference on Digital Libraries*, DL '00, (New York, NY, USA), pp. 85–94, ACM, 2000.

[26] R. Gabbard, M. Freedman, and R. Weischedel, "Coreference for learning to extract relations: Yes virginia, coreference matters," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, (Portland, Oregon, USA), pp. 288–293, Association for Computational Linguistics, June 2011.

[27] T. Hasegawa, S. Sekine, and R. Grishman, "Discovering relations among named entities from large corpora," in *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, (Barcelona, Spain), pp. 415–422, July 2004.

[28] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates, "Web-scale information extraction in knowitall: (preliminary results)," in *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, (New York, NY, USA), pp. 100–110, ACM, 2004.

[29] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates, "Unsupervised named-entity extraction from the web: An experimental study," *Artif. Intell.*, vol. 165, pp. 91–134, June 2005.

[30] Y. Yan, N. Okazaki, Y. Matsuo, Z. Yang, and M. Ishizuka, "Unsupervised relation extraction by mining Wikipedia texts using information from the web," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, (Suntec, Singapore), pp. 1021–1029, Association for Computational Linguistics, Aug. 2009.

[31] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni, "Open information extraction from the web," in *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, (San Francisco, CA, USA), pp. 2670–2676, Morgan Kaufmann Publishers Inc., 2007.

[32] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Comput. Linguist.*, vol. 19, pp. 313–330, June 1993.

[33] M. Banko and O. Etzioni, "The tradeoffs between open and traditional relation extraction," in *Proceedings of ACL-08: HLT*, (Columbus, Ohio), pp. 28–36, Association for Computational Linguistics, June 2008.

[34] A. Yates and O. Etzioni, "Unsupervised resolution of objects and relations on the web," in *Human Language Technologies 2007: The Conference of the North*

American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference, (Rochester, New York), pp. 121–130, Association for Computational Linguistics, Apr. 2007.

[35] F. Wu and D. S. Weld, "Open information extraction using wikipedia," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, (Stroudsburg, PA, USA), pp. 118–127, Association for Computational Linguistics, 2010.

[36] A. Fader, S. Soderland, and O. Etzioni, "Identifying relations for open information extraction," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, (Stroudsburg, PA, USA), pp. 1535–1545, Association for Computational Linguistics, 2011.

[37] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A core of semantic knowledge," in *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, (New York, NY, USA), pp. 697–706, ACM, 2007.

[38] G. A. Miller, "Wordnet: A lexical database for english," *Commun. ACM*, vol. 38, pp. 39–41, Nov. 1995.

[39] S. Auer and J. Lehmann, "What have innsbruck and leipzig in common? extracting semantics from wiki content," in *Proceedings of the 4th European Conference on The Semantic Web: Research and Applications*, ESWC '07, (Berlin, Heidelberg), pp. 503–517, Springer-Verlag, 2007.

[40] E. Kuzey and G. Weikum, "Evin: Building a knowledge base of events," in *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, (New York, NY, USA), pp. 103–106, ACM, 2014.

[41] E. Kuzey, J. Vreeken, and G. Weikum, "A fresh look on knowledge bases: Distilling named events from news," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, (New York, NY, USA), pp. 1689–1698, ACM, 2014.

[42] M. L. de Lacalle, E. Laparra, and G. Rigau, "First steps towards a predicate matrix," in *Proceedings of the Seventh Global Wordnet Conference*, (Tartu, Estonia), pp. 363–371, University of Tartu Press, Jan. 2014.

[43] C. F. Baker, C. J. Fillmore, and J. B. Lowe, "The berkeley framenet project," in *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1*, ACL '98/COLING '98, (Stroudsburg, PA, USA), pp. 86–90, Association for Computational Linguistics, 1998.

[44] M. Rospocher, M. van Erp, P. Vossen, A. Fokkens, I. Aldabe, G. Rigau, A. Soroa, T. Ploeger, and T. Bogaard, "Building event-centric knowledge graphs from news," *Web Semant.*, vol. 37, pp. 132–151, Mar. 2016.

[45] R. Agerri, I. Aldabe, Z. Beloki, E. Laparra, M. L. de Lacalle, G. Rigau, A. Soroa, A. Fokkens, R. Izquierdo, M. van Erp, P. Vossen, C. Girardi, and A. Minard, "Event detection, version 2, deliverable d4.2.2." `http://www.newsreader-project.eu/files/2012/12/NWR-D4-2-2.pdf`.

[46] S. Gottschalk and E. Demidova, "Eventkg: A multilingual event-centric temporal knowledge graph," *ArXiv*, vol. abs/1804.04526, 2018.

[47] A. Fokkens, A. Soroa, Z. Beloki, N. Ockeloen, G. Rigau, W. R. Van Hage, and P. Vossen, "Naf and gaf: Linking linguistic annotations," in *Proceedings 10th Joint ISO-ACL SIGSEM Workshop on Interoperable Semantic Annotation*, pp. 9–16, 2014.

[48] P. Kingsbury and M. Palmer, "From TreeBank to PropBank," in *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02)*, (Las Palmas, Canary Islands - Spain), European Language Resources Association (ELRA), May 2002.

[49] K. K. Schuler, *Verbnet: A Broad-coverage, Comprehensive Verb Lexicon*. PhD thesis, Philadelphia, PA, USA, 2005. AAI3179808.

[50] F. Erxleben, M. Günther, M. Krötzsch, J. Mendez, and D. Vrandeăić, "Introducing wikidata to the linked data web," in *Proceedings of the 13th International Semantic*

Web Conference - Part I, ISWC '14, (New York, NY, USA), pp. 50–65, Springer-Verlag New York, Inc., 2014.

[51] G. B. Tran and M. Alrifai, "Indexing and analyzing wikipedia's current events portal, the daily news summaries by the crowd," in *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, (New York, NY, USA), pp. 511–516, ACM, 2014.

[52] D. Hienert, D. Wegener, and H. Paulheim, "Automatic classification and relationship extraction for multi-lingual and multi-granular events from wikipedia," in *DeRiVE@ISWC*, 2012.

[53] W. R. Van Hage, V. Malaisé, R. Segers, L. Hollink, and G. Schreiber, "Design and use of the simple event model (sem)," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 9, no. 2, pp. 128–136, 2011.

[54] G. Rizzo, M. van Erp, and R. Troncy, "Benchmarking the extraction and disambiguation of named entities on the semantic web," in *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*, (Reykjavik, Iceland), pp. 4593–4600, European Languages Resources Association (ELRA), May 2014.

[55] M. Morsey, J. Lehmann, S. Auer, C. Stadler, and S. Hellmann, "Dbpedia and the live extraction of structured data from wikipedia," *Program*, vol. 46, pp. 157–181, 2012.

[56] V. Nguyen, O. Bodenreider, and A. Sheth, "Don't like rdf reification?: Making statements about statements using singleton property," in *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, (New York, NY, USA), pp. 759–770, ACM, 2014.

[57] W3C, "Time ontology in owl." `https://www.w3.org/TR/owl-time/`.

[58] Twitter, "Get tweet timelines - documentation pages." `https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user_timeline.html`.

[59] F. Å. Nielsen, "A new anew: Evaluation of a word list for sentiment analysis in microblogs," *arXiv preprint arXiv:1103.2903*, 2011.

[60] Twitter, "Search tweets - documentation pages." `https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets.html`.

[61] DBpedia, "Public sparql endpoint - documentation pages." `https://wiki.dbpedia.org/public-sparql-endpoint`.

[62] M. A. Hearst, "Automatic acquisition of hyponyms from large text corpora," in *Proceedings of the 14th Conference on Computational Linguistics - Volume 2*, COLING '92, (Stroudsburg, PA, USA), pp. 539–545, Association for Computational Linguistics, 1992.

Appendix A

README

Four different projects are included in the accompanying USB flash drive to this document. Those projects are listed below:

- NewsCollector, a Python 3 project
- NewsPreprocessor, a Python 3 project
- TweetsCollectorNTA, a Python 3 project
- NewsTextAnalyzer, a Java 8 project

A.1 NewsCollector and NewsPreprocessor

NewsCollector as well as the rest of Python projects were created in PyCharm[1] 2018.3.5.

NewsCollector and NewsPreprocessor take care of retrieving content from The Guardian and extracting the important article information.

A.1.1 NewsCollector Configuration

1. Specify a directory where to save the news articles in the NewsAPICollector.py script.

[1] `https://www.jetbrains.com/pycharm/`

2. Specify an API key to retrieve content from The Guardian's API in the same script.

A.1.2 NewsPreprocessor Configuration

1. In the TheGuardianNewsPreprocessor.py script, specify the directory where the NewsCollector retrieved the news articles from The Guardian .

2. In the same script, specify a directory where NewsPreprocessor should save the preprocessed news articles.

A.2 TweetsCollectorNTA

TweetsCollectorNTA takes care of retrieving tweet posts from @GdnPolitics and replies made by the public to such posts.

A.2.1 Configuration

1. In the collector.py script, specify the directory where TweetsCollectorNTA should save the data recovered from Twitter.

2. In the same script, specify an API key, APP secret, user OAuth token, and user OAuth token secret to retrieve content from Twitter's API.

A.3 NewsTextAnalyzer

NewsTextAnalyzer is the main project that contains the NLP pipeline to extract assertions and events from news articles and creates the knowledge graph. The project was created with Eclipse 2018-09 (4.9.0).

A.3.1 Requirements

- A triplestore needs to be set up and accessible through a connection string like `jdbc:virtuoso://localhost:1111`, with username "dba" and password "dba". The triplestore needs to contain a named graph called `http://test1/`.
- The path to The Guardian preprocessed corpus needs to be correctly specified in the Const.java class.
- The path to the @GdnPolitics tweets needs to be correctly specified in the TweetIndexer.java class.
- Two Lucene indexes need to be created: the Person Resource Index and the Tweets Index. Paths to such indexes need to be correctly specified in the PersonResourceLookup.java and TweetIndexer.java classes, respectively.

A.3.2 Installation and Configuration

1. Import the project as a Maven[2] project on Eclipse.
2. Install Apache Tomcat/9.0.21 and configure it to run on port 8080.
3. In the tomcat-users.xml file within Tomcat's conf folder, create a "manager-gui" role and a user with roles "manager-gui,manager-script". Consult links[3] [4] for additional information on this task.
4. Install the Maven Apache Tomcat 7 plugin[5] to support easy deployment of WARs.
5. In the global Maven settings.xml file located within the Maven installation conf folder, add a server entry called "TomcatServer" to match the name of the server defined in the project's pom.xml file. Next, specify the username and password to operate such server within the server entry. Such username and password must match the ones defined on the Tomcat's tomcat-users.xml.

[2] `https://maven.apache.org/`
[3] `https://www.baeldung.com/tomcat-deploy-war`
[4] `https://howtodoinjava.com/maven/tomcat-maven-plugin-example/`
[5] `https://mvnrepository.com/artifact/org.apache.tomcat.maven/tomcat7-maven-plugin`

A.4 Execution

See below for the list of steps to run the system.

1. Run NewsCollector to gather news articles. Depending on start and end date the process could take a couple of hours.
2. Run NewsPreprocessor on the retrieved articles using the same time window.
3. Run TweetsCollectorNTA to recover tweets. Because of the seven-day old limit imposed by Twitter, it is necessary to run this script periodically, daily if possible, to eventually build a small corpus.
4. Run NewsTextAnalyzer with the following parameter configured in Eclipse: "scrap". This parameter will indicate the program to build the Tweet Index and build the Person Resource Index by retrieve resources from DBpedia. This process could take up to 12 hours depending on how 'busy' DBpedia is, but it is a one-time process. Subsequent runs of the system should not execute this step.
5. If the system will rerun from scratch, make sure the entity linking cache file in NewsTextAnalyzer/data/cross_entity_linking.txt is blank.
6. Run NewsTextAnalyzer with the following two parameters configured in Eclipse: "process" "enrich". This will take the preprocessed news articles through the pipeline, build the knowledge graph and enrich it with sentiment from the Tweets Index. Depending on the start and end date specified in the App.java class, this process could take about 12 hours to process 250,000 articles.
7. Using the Maven Tomcat 7 plugin, deploy the WAR generated by the project. Use the command "mvn tomcat7:deploy" on terminal to do so.
8. Visit `http://localhost:8080/myapp` to access the application query UI.