Knowledge Graph Construction from Unstructured Text with Applications to Fact Verification and Beyond

Ryan Clancy, Ihab F. Ilyas, and **Jimmy Lin**
David R. Cheriton School of Computer Science
University of Waterloo

Abstract

We present a scalable, open-source platform that "distills" a potentially large text collection into a knowledge graph. Our platform takes documents stored in Apache Solr and scales out the Stanford CoreNLP toolkit via Apache Spark integration to extract mentions and relations that are then ingested into the Neo4j graph database. The raw knowledge graph is then enriched with facts extracted from an external knowledge graph. The complete product can be manipulated by various applications using Neo4j's native Cypher query language: We present a subgraph-matching approach to align extracted relations with external facts and show that fact verification, locating textual support for asserted facts, detecting inconsistent and missing facts, and extracting distantly-supervised training data can all be performed within the same framework.

1 Introduction

Despite plenty of work on relation extraction, entity linking, and related technologies, there is surprisingly no scalable, open-source platform that performs end-to-end knowledge graph construction, taking as input a large text collection to "distill" a knowledge graph from it. Many enterprises today desire exactly such a system to integrate analytics over unstructured text with analytics over relational as well as semi-structured data. We are aware of a few commercial solutions for analyzing unstructured text, such as Amazon Comprehend and Refinitiv by Thomson Reuters, as well as many organizations with large internal efforts, most notably Bloomberg. However, there does not appear to be comparable open-source solutions.

This gap can be attributed, at least in part, to the fact that NLP researchers typically think about extraction in terms of sentences (or documents) and may not be interested in the engineering efforts required to scale out extraction to hundreds of thousands (or even millions) of documents. Furthermore, they are less likely equipped with the expertise (or interest) necessary to build distributed, scalable systems.

We share with the community an open-source platform for scalable, end-to-end knowledge graph construction from unstructured text called dstlr. By "end-to-end" we mean a solution that aspires to cover all aspect of the data management lifecycle: from document ingestion to relation extraction to graph management to knowledge curation to supporting downstream applications, plus integration with other systems in an enterprise's "data lake". Although other researchers have proposed solutions to knowledge graph construction (Augenstein et al., 2012; Kertkeidkachorn and Ichise, 2017), they do not appear to make open-source software artifacts available for download and evaluation.

At a high level, dstlr takes unstructured text and "distills" from it a usable knowledge graph. From a corpus stored in Apache Solr, a raw knowledge graph is populated using Stanford CoreNLP and ingested into the popular Neo4j graph database. This raw knowledge graph is further enriched with facts from an external knowledge graph, in our case, Wikidata (Vrandečić and Krötzsch, 2014). The final product can be manipulated via the declarative Cypher query language. All computations are orchestrated using Apache Spark for horizontal scaling.

On top of our platform, it is possible to build a number of applications, for example, to support business intelligence, knowledge discovery, and semantic search. In this paper, we describe an approach to align extracted relations from the corpus with external facts. We show that fact verification, locating textual support for asserted facts, detecting inconsistent and missing facts, and extracting distantly-supervised training data can all be formulated in terms of graph queries. As a case study, we extract and subsequently manipulate approximately 100 million triples from nearly 600K Washington Post articles on a modest cluster.

The contribution of this work is the creation of an end-to-end platform for constructing knowledge graphs from unstructured text "with minimal fuss" via the integration of four mature technologies: Apache Solr, Stanford CoreNLP, Apache Spark, and Neo4j. We demonstrate the potential of such a platform and are pleased to share our open-source project with the community.

2 Problem Formulation

We begin with a more precise formulation of the problem at hand. Given a (potentially large) collection of text documents, we wish to extract facts comprising what we call *mentions* (spans of natural language text such as named entities) and relations between them. We further assume the existence of an external knowledge graph which provides a "ground-truth" inventory of entities, and central to our task is linking *mentions* to these entities. The distinction between mentions and entities is crucial to our problem formulation, as mentions are simply spans of text that exhibit a wide range of linguistic phenomena (synonymy, polysemy, etc.), but entities are unique and have clear real-world referents. More formally:

- Documents contain zero or more mentions.
- In each document, there are zero or more relations between mentions. Mentions can participate in an arbitrary number of relations.
- Each mention has zero or exactly one link to an entity in the external knowledge graph.
- Entities participate in an arbitrary number of facts in the external knowledge graph.

The usage scenario we have in mind is the construction of enterprise-centric knowledge graphs. Most large organizations today already have internal knowledge graphs (or ongoing efforts to build them); the simplest example might be a machine-readable product catalog with product specifications. Our goal is to provide a "360-degree view" of unstructured text in an organization's "data lake"; although similar capabilities already exist for relational and semi-structured (e.g., log) data, unstructured free text remains vastly underexplored. As we show, it is exactly this interplay between relations extracted from unstructured text and facts in the external knowledge graph that give rise to interesting applications in fact verification and related tasks. Currently, we use Wikidata as a stand-in, as our platform is designed to be enterprise and domain agnostic.

Of course, relation extraction and complementary tasks such as entity linking, coreference resolution, predicate mapping, etc. have been studied for decades. Notable efforts include the Knowledge Base Population (KBP) and Knowledge Base Acceleration (KBA) tasks in the Text Analysis Conference (TAC) series (Ji and Grishman, 2011; Getman et al., 2018), NELL (Never-Ending Language Learning) (Mitchell et al., 2015), open information extractors such as Ollie (Mausam et al., 2012), and approaches based on weak supervision such as Snorkel (Ratner et al., 2017). Our focus, however, is very different as we wish to build an end-to-end platform that not only supports extraction, but the entire data management lifecycle, as discussed in the introduction. Part of this effort is the development of various applications that exploit knowledge graphs (for example, fact verification). In this sense, our work is complementary to all these abovementioned systems and techniques, as the dstlr platform is sufficiently general to incorporate their extraction results.

3 System Overview

The overall architecture of our open-source dstlr[1] platform is shown in Figure 1. We assume the existence of a document store that holds the document collection we wish to "distill". Currently, we use Apache Solr for this role, although a good alternative would be Elasticsearch (which we are currently implementing support for).

The rationale for depending on a document store, as opposed to simply reading documents from a file system (e.g., one designed for distributed storage such as the Hadoop Distributed File System) are many: First, it is likely that users and applications of dstlr desire full-text and metadata search capabilities. A system like Solr readily provides an "industrial strength" solution. Second, a document store provides more refined mechanisms for managing incremental data ingestion, e.g., the periodic arrival of a new batch of docu-

[1]`http://dstlr.ai/`

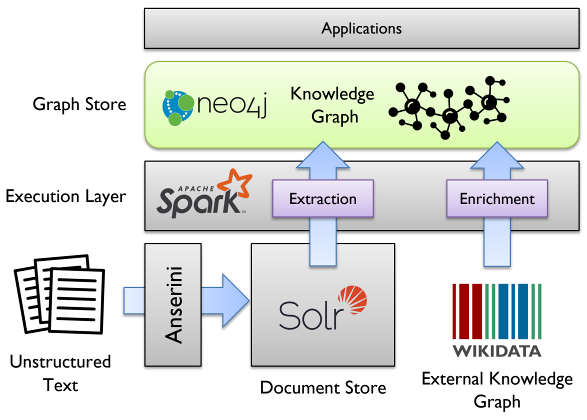

Figure 1: The overall architecture of the dstlr platform. Documents are "distilled" into a raw knowledge graph in the extraction phase, which is then enriched with facts from an external knowledge graph. Spark orchestrates execution in a horizontally scalable manner. Neo4j holds the knowledge graph, which supports applications via its query interface.

ments. Third, the integration of search capabilities with a document store allows dstlr to focus analyses on subsets of documents, as demonstrated in Clancy et al. (2019b). For convenience, our open-source search toolkit Anserini (Yang et al., 2018) provides a number of connectors for ingesting document collections into Solr (Clancy et al., 2019a), under different index architectures.

The execution layer, which relies on Apache Spark, coordinates the two major phases of knowledge graph construction: *extraction* and *enrichment*. The knowledge graph is held in the popular graph database Neo4j. Applications built on top of the dstlr platform take advantage of a declarative query language called Cypher to manipulate the contents of the knowledge graph.

The extraction phase is responsible for populating the raw knowledge graph with mentions, entities, and relations identified from unstructured text. Currently, we use Stanford's CoreNLP toolkit (Manning et al., 2014) for the JVM due to its support for many common language analysis tasks (i.e., tokenization, part-of-speech tagging, named entity recognition, etc.) that can be chained together in a pipeline. While dstlr is extractor agnostic and we have explored a number of different systems, we have found CoreNLP to be the most straightforward package to deploy from an engineering perspective. One of the contributions of our platform is the infrastructure to scale out the CoreNLP toolkit using Spark to process large document collections in an scalable manner. At a high level, annotator output is converted into tuples (as part of Spark DataFrames) that are then ingested into the knowledge graph.

In the enrichment phase, we extract entities from the external knowledge graph (Wikidata) that are found in the unstructured text, thereby enriching the raw knowledge graph with high-quality facts from an external source. We perform this filtering step because, typically, only a portion of entities in an external source like Wikidata are referenced in a corpus; thus, for query and storage efficiency, it makes sense to only enrich entities that are mentioned in the source documents.

In what follows, we provide more details about the extraction and enrichment phases.

3.1 Extraction

For each document in the collection, we extract mentions of named entities, the relations between them, and links to entities in an external knowledge graph. Through Solr/Spark integration, extraction can be performed on all documents in the document store, or a subset that a user or an application may wish to focus on, for example, containing a particular metadata facet or the results of a keyword query (Clancy et al., 2019b).

Named Entity Extraction. We use CoreNLP's `NERClassifierCombiner` annotator to extract entity mentions of 20 different types, such as persons, organizations, locations, etc. (Finkel et al., 2005). Each mention corresponds to a row in a Spark DataFrame that contains the document id, a mention id, and a list of key–value pairs containing the mention class, mention text, and character offsets of the mention in the source document. Subsequent occurrences of the same mention in the same document are mapped to an existing mention so that character offsets can later be consolidated into an array. In this manner, we retain accurate provenance that allows us to trace back an extracted mention to its source.

Relation Extraction. CoreNLP provides two different annotators for relation extraction: the Open Information Extraction (OpenIE) annotator (Angeli et al., 2015) and the Knowledge Base Population (KBP) annotator (Surdeanu et al., 2012). The OpenIE annotator provides open-class relations based on the provided text while the KBP annotator fills slots for a fixed set of 45 relations, such as `org:city_of_headquarters`. We use the latter, as it is more appropriate for our task. As with the entities, each extracted relation corresponds to a row in a Spark DataFrame with the document id, the subject mention, the relation, the object mention, and a confidence score.

Entity Linking. In the final extraction step, we use CoreNLP's `WikidictAnnotator` to link mentions to their corresponding Wikipedia entity, which in turn allows us to map to Wikidata entities. For each entity mention, we produce a row in a Spark DataFrame that contains the document id, the mention id, and the URI of the most likely entity link. If a linked entity for a mention cannot be found, the row will contain a `null`; we made the design decision to explicitly record these cases. Note that this important step establishes correspondences between information extracted from a document and the external knowledge graph, and is critical to enabling the host of applications that we discuss later.

Pulling everything together, consider the sample sentence "Facebook is an American social media and social networking company based in Menlo Park" from document `b2c9a`. The dstlr extraction pipeline might discover the mentions "Facebook" and "Menlo Park" with a `has_hq` relation between them. Furthermore, "Facebook" is linked to the entity Q355 in Wikidata. This translates into the following knowledge graph fragment, in terms of (subject, relation, object) triples, simplified for illustrative purposes:

(`b2c9a`, `mentions`, "Facebook")
(`b2c9a`, `mentions`, "Menlo Park")
("Facebook", `subject_of`, `has_hq`)
(`has_hq`, `object_of`, "Menlo Park")
("Facebook", `links_to`, Q355)

The relation itself is reified into a node to facilitate efficient querying by consumers of the knowledge graph (more details later). Following the extraction of all entity mentions, relations, and entity links as described above, the resulting rows from the Spark DataFrames are bulk-loaded into Neo4j according to the "schema" above. While it is entirely possible to construct the raw knowledge graph incrementally, we have found it to be far more efficient to perform ingestion in bulk.

3.2 Enrichment

In the enrichment phase, we augment the raw knowledge graph with facts from the external knowledge graph (Wikidata). This is accomplished by first manually defining a mapping from CoreNLP relations to Wikidata properties. For example, the "headquarters" relation from CoreNLP most closely corresponds to P159 in Wikidata.[2] Since there are only 45 relations, this did not require much effort. Then, for each distinct entity that was discovered in the document collection, we extracted the corresponding facts from Wikidata. This process, in essence, extracts subgraph fragments around referenced entities to enrich the raw knowledge graph. Currently, we only perform the augmentation with relations that are covered by CoreNLP and referenced entities in the unstructured text, but we could easily scale up (or down) the enrichment effort by "pulling in" more (or less) of Wikidata, depending on the needs of various applications. For fact verification and the related tasks that we explore in this paper, the parts of Wikidata that do not overlap with the raw knowledge graph are not needed.

As with extraction, execution of the enrichment process is coordinated by Spark via the manipulation of DataFrames. With Spark, it is easy to identify the distinct entities referenced in the corpus, which are then fed as input into the enrichment process to "pull out" the appropriate parts of Wikidata. For each entity in the corpus, we produce a row in a Spark DataFrame containing the entity URI, the relation type, and its value. These are then bulk-loaded into Neo4j; once again, this is done primarily for efficiency, just as with the extraction output.

In our running example about Facebook, this would lead to the insertion of the following triple in the graph (once again, slightly simplified):

(Q355, `links_to`, FACT_{34a8d})

where FACT_{34a8d} is a node that has type `has_hq` and value "Menlo Park". Facts from Wikidata are factored according to this "schema" to facilitate efficient querying (more details below).

4 Performance Evaluation

To demonstrate the scalability of our dstlr platform, we describe an evaluation comprising both extraction and enrichment performed on a cluster comprising nine nodes. Each node has two Intel E5-2670 @ 2.60GHz (16 cores, 32 threads)

[2] https://www.wikidata.org/wiki/Property:P159

CPUs, 256GB RAM, 6×600GB 10K RPM HDDs, 10GbE networking, and runs Ubuntu 14.04 with Java 9.0.4. We utilize one node for the master services (YARN ResourceManager and HDFS NameNode); the remaining eight nodes each host a HDFS DataNode, a Solr shard, and are available for Spark worker allocation via YARN.

We ran dstlr on the TREC Washington Post Corpus,[3] which contains 595K news articles and blog posts from January 2012 to August 2017. This corpus has been used in several recent TREC evaluations and is representative of a modern newswire collection. We performed some light data cleaning before running extraction, discarding documents longer than 10,000 tokens and sentences longer than 256 tokens. These outliers were typically HTML tables that our document parser processes into very long "sentences" (concatenating all table cells together). After filtering, we arrive at a collection with 580K documents, comprising roughly 23M sentences and approximately 500M tokens.

Extraction was performed via a Spark job configured with 32 executors (four per machine), each allocated 8 CPU cores and 48GB of memory for task processing; the configuration attempts to maximize resource usage across the cluster. We extracted 97M triples from the approximately 580K documents in 10.4 hours, for a processing rate of 13K token per second. Mentions of entities and mention-to-entity links account for 92.2M triples (46.1M each) while the remaining 4.8M represent relations between mentions. Of the 46.1M mention-to-entity links, 30.7M correspond to an actual Wikidata entity (recall that we explicitly store `null` links); this represents 324K distinct entities.

Currently, dstlr uses Neo4j Community Edition, a popular open-source graph database, as the graph store. Running on a single node, we are able to insert 97M triples in 7.8 hours using a single Spark worker, co-located on the same machine as Neo4j. This translates into an ingestion rate of nearly 2.9K triples/sec, which we find acceptable for a corpus of this size. Further scale out is possible via a distributed version of Neo4j, which is available as part of the Enterprise Edition, but requires a commercial license.

Enrichment is performed by querying a local instance of Wikidata that has been ingested into Apache Jena Fuseki, which is an open-source RDF store that provides a REST-based SPARQL endpoint. We import the "truthy" dump, consisting of only facts, in addition to a mapping from Wikipedia to Wikidata URIs, as CoreNLP links entities to Wikipedia URIs.

For each of the 324K distinct entities found in the corpus, we fetch the corresponding facts from Wikidata using our Jena Fuseki endpoint. As with the extraction phase, the enrichment process is orchestrated by Spark. For example, for the "headquarters" relation, we are able to retrieve 11.7K corresponding facts from Wikidata in around 14 minutes. These extracted facts are then inserted into Neo4j, as described in Section 3.2. This ingestion takes only a few seconds.

5 Graph Alignment and Applications

In our case study, the "product" of dstlr is a knowledge graph constructed from a corpus of unstructured text (Washington Post articles) that has been enriched with high-quality facts extracted from an external knowledge graph (Wikidata). The knowledge graph, stored in Neo4j, can then be manipulated by different applications using Neo4j's declarative query language called Cypher.

We describe a query-driven approach to align extracted relations from CoreNLP to external facts from Wikidata. This, in essence, performs fact verification (Thorne and Vlachos, 2018) against an external knowledge source that is presumed to have high-quality facts. While fact verification using external knowledge sources is not novel (Vlachos and Riedel, 2015), the contribution of our particular case study is to illustrate how it can be recast into a query-driven subgraph alignment problem. Within this framework, fact verification, locating textual support for asserted facts, updating incorrectly-asserted facts, asserting newly-discovered facts, and data augmentation via distant supervision can all be viewed as different aspects of the same underlying task.

As an illustration of these ideas, we consider the `city_of_headquarters` relation identified by CoreNLP. Figure 2 shows a Cypher query that performs one possible subgraph alignment. A typical Cypher query has three main clauses: the `MATCH` clause describes, in a pseudo-graphical notation, the graph pattern being searched for; the `WHERE` clause specifies additional constraints on the patterns, akin to the `WHERE` clause in SQL; finally,

[3]https://trec.nist.gov/data/wapost/

```
MATCH (d:Document)-->(s:Mention)
MATCH (s)-->(r:Relation
{ type:  "CITY_OF_HEADQUARTERS" })
MATCH (r)-->(o:Mention)
MATCH (s)-->(e:Entity)
MATCH (e)-->(f:Fact {relation:  r.type})
RETURN d, s, r, o, e, f
```

Figure 2: Cypher query to match extracted relations with external facts.

the `RETURN` clause specifies the result of the query. This particular query looks for two mention nodes connected by a `city_of_headquarters` relation for which there exists a linked entity for the subject mention with the same fact type in Wikidata as the extracted relation, returning tuples of all matched instances. In other words, we look for all instances of extracted headquarters, and then match the headquarters with what's stored in the external knowledge graph.

We query Neo4j with its Spark connector so that the results can be integrated with subsequent distributed processing on our cluster. This particular query returns around 21K results in less than twenty seconds, where the majority of the time is spent on Spark overhead such as copying dependencies to worker nodes.

What can we do with the results? Logically, there are three possibilities: an extracted relation matches what's in the external knowledge graph, an extracted relation doesn't match what's in the external knowledge graph, and the extract relation isn't found in the external knowledge graph.[4] These correspond to identification of supporting evidence, inconsistent facts, and missing facts; we consider each of these cases in detail below.

5.1 Supporting Facts

If an extracted relation matches a fact asserted in a high-quality external source such as Wikidata, we can conclude with reasonable certainty that the information contained in the source document is indeed factual. As a specific example, our Cypher query identified Washington Post article `eb3b8f` as discussing the company "Good Technology", and from that article, CoreNLP was able to identify its headquarters as Sunnyvale. Figure 3 shows this subgraph, illustrating agreement on Sunnyvale between the external knowledge graph (leftmost node) and the document (rightmost node).

[4]This is actually the result of another query, but same idea.

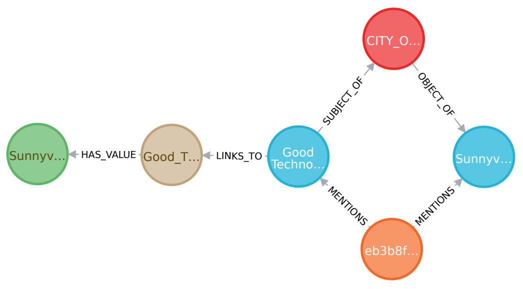

Figure 3: Example where an extracted relation matches an external fact, providing textual support for the fact.

This forms the basis of fact verification, although there are nuanced cases that require human judgment (and thus are difficult to automate). For example, it could be the case that *both* the knowledge graph and the document are wrong, such as when a fact is outdated. Since we retain the provenance of all extracted relations, it is possible for a human to trace back evidence to its source in order to consider the broader context.

This feature allows applications to locate supporting evidence for facts in Wikidata. Existing knowledge graphs are typically constructed through a combination of different processes, ranging from manual entry to semi-automated techniques. It is frequently the case that the provenance is lost, and thus the knowledge graph cannot answer the question: how do we know this fact to be true? Our application can provide such support, and from multiple sources to boot.

5.2 Inconsistent Facts

Relations extracted from unstructured text may be inconsistent with facts in the external knowledge graph for a variety of reasons, but can be grouped into two categories, either imperfect extractors or factual errors in the documents. Distinguishing these two cases requires manual inspection, but once again, subgraph alignment provides the basis of fact verification.

Based on our own manual inspection, the overwhelming majority of inconsistencies stems from extractor errors. For example, Washington Post article `b02562` contains the sentence "The company said that it will have watches to demo at department stores around the world: the Galeries Lafayette in Paris, Isetan in Tokyo, . . . ", from which CoreNLP asserts that Isetan has headquarter in Paris, which is obviously incorrect to a human reader. The corresponding subgraph is presented in Figure 4, which shows that the Wiki-

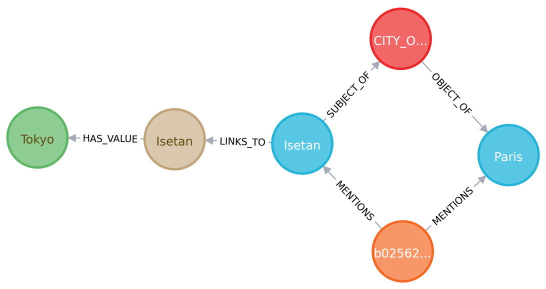

Figure 4: Example where an extracted relation is inconsistent with an external fact. In this case, the inconsistency arises from an extractor error.

data fact (leftmost node) differs from the extracted value (rightmost node).

These inconsistencies can serve as negative training data that can then be used to train better extraction models. While this is a standard technique in the literature on distant supervision for relation extraction (Smirnova and Cudré-Mauroux, 2018), we show how the process can be formulated in terms of graph queries in our dstlr platform, in essence, as a side effect of fact verification. In principle, supporting texts, such as those from the previous section, can be used as positive examples as well, although their benefits are likely to be limited as the extractor was able to correctly identify the relation to begin with.

5.3 Missing Facts

In trying to align subgraphs from extracted relations with external facts, the third possibility is that we find no corresponding fact. For example, Washington Post article `498e15` discusses a climatologist at the International Arctic Research Center in Fairbanks, Alaska. Our platform extracts Fairbanks as the headquarters of the International Arctic Research Center (see Figure 5). During the enrichment process, no value from Wikidata was present for the property P159 (based on our CoreNLP to Wikidata mapping). This can also be confirmed by noticing the lack of an infobox in the upper right-hand corner of its Wikipedia page.

In other words, we have discovered a missing fact in Wikidata, thus providing an opportunity to populate Wikidata with new knowledge. Of course, human vetting is likely needed before any facts are added to an external knowledge graph, but once again, subgraph alignment via Cypher graph queries provides the starting point for knowledge acquisition.

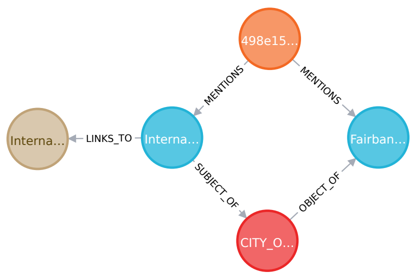

Figure 5: Example where an extracted relation does not correspond to any fact in the external knowledge graph, providing an opportunity acquire new knowledge.

6 Conclusion

The contribution of dstlr is a scalable, open-source, end-to-end platform that distills a potentially large text collection into a knowledge graph. While each of the components in our architecture already exist, they have not been previously integrated in this manner to support knowledge graph construction and applications that exploit knowledge graphs.

The other interesting aspect of our work is the use of subgraph alignment to support a number of related tasks: we show that fact verification, locating textual support for asserted facts, detecting inconsistent and missing facts, and extracting distantly-supervised training data can all be performed within the same graph querying framework. The dstlr platform is under active development, with plans to integrate more extractors, particular ones based on neural networks, in support of applications in business intelligence, knowledge discovery, and semantic search.

Acknowledgments

This research was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada, with additional funding from the Waterloo–Huawei Joint Innovation Lab.

References

Gabor Angeli, Melvin Johnson Premkumar, and Christopher D. Manning. 2015. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 344–354, Beijing, China.

Isabelle Augenstein, Sebastian Padó, and Sebastian Rudolph. 2012. LODifier: Generating linked data from unstructured text. In *Proceedings of the 9th Extended Semantic Web Conference (ESWC 2012)*, pages 210–224, Heraklion, Crete.

Ryan Clancy, Toke Eskildsen, Nick Ruest, and Jimmy Lin. 2019a. Solr integration in the Anserini information retrieval toolkit. In *Proceedings of the 42nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2019)*, pages 1285–1288, Paris, France.

Ryan Clancy, Jaejun Lee, Zeynep Akkalyoncu Yilmaz, and Jimmy Lin. 2019b. Information retrieval meets scalable text analytics: Solr integration with Spark. In *Proceedings of the 42nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2019)*, pages 1313–1316, Paris, France.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 363–370, Ann Arbor, Michigan.

Jeremy Getman, Joe Ellis, Stephanie Strassel, Zhiyi Song, and Jennifer Tracey. 2018. Laying the groundwork for knowledge base population: Nine years of linguistic resources for TAC KBP. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*, Miyazaki, Japan.

Heng Ji and Ralph Grishman. 2011. Knowledge base population: Successful approaches and challenges. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1148–1158, Portland, Oregon.

Natthawut Kertkeidkachorn and Ryutaro Ichise. 2017. T2KG: An end-to-end system for creating knowledge graph from unstructured text. In *Proceedings of the AAAI-17 Workshop on Knowledge-Based Techniques for Problem Solving and Reasoning*, pages 743–749, San Francisco, California.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Baltimore, Maryland.

Mausam, Michael Schmitz, Stephen Soderland, Robert Bart, and Oren Etzioni. 2012. Open language learning for information extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 523–534, Jeju Island, Korea.

Tom M. Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, Justin Betteridge, Andrew Carlson, Bhavana Dalvi Mishra, Matthew Gardner, Bryan Kisiel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Thahir Mohamed, Ndapa Nakashole, Emmanouil Antonios Platanios, Alan Ritter, Mehdi Samadi, Burr Settles, Richard Wang, Derry Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves, and Joel Welling. 2015. Never-ending learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*, pages 2302–2310, Austin, Texas.

Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, 11(3):269–282.

Alisa Smirnova and Philippe Cudré-Mauroux. 2018. Relation extraction using distant supervision: A survey. *ACM Computing Surveys*, 51(5):106:1–106:35.

Mihai Surdeanu, Julie Tibshirani, Ramesh Nallapati, and Christopher D. Manning. 2012. Multi-instance multi-label learning for relation extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 455–465, Jeju Island, Korea.

James Thorne and Andreas Vlachos. 2018. Automated fact checking: Task formulations, methods and future directions. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3346–3359, Santa Fe, New Mexico.

Andreas Vlachos and Sebastian Riedel. 2015. Identification and verification of simple claims about statistical properties. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2596–2601, Lisbon, Portugal.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: A free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85.

Peilin Yang, Hui Fang, and Jimmy Lin. 2018. Anserini: Reproducible ranking baselines using Lucene. *Journal of Data and Information Quality*, 10(4):Article 16.