Building Knowledge Graphs

Lukas Blübaum
Nick Düsterhus
Monika Werner

University of Paderborn

https://github.com/LukasBluebaum/BKG

June 21, 2018

Overview

Extraction and cleaning of text from Wikipedia Dump

Foundation

Wikipedia Dump consisting of plain text without markup, infoboxes or chapter subdivision

- Using regular expressions to remove unnecessary URLs, parentheses, whitespace, nullchar, symbols
- WikiCleaner class using Writer and Reader Threads for faster cleaning

Extraction and cleaning of text from Wikipedia Dump

```java
package wikicleaner;

import java.util.concurrent.BlockingQueue;

public class Worker implements Runnable{

    private static final Pattern URLS = Pattern.compile("http.*?\\s");

    private static final Pattern PARENTHESES = Pattern.compile("\\(.*?\\)");

    private static final Pattern SYMBOLS = Pattern.compile("[^a-zA-Z0-9. ]");

    private static final Pattern NULLCHAR = Pattern.compile("\0");

    private static final Pattern WHITESPACE = Pattern.compile("\\s+");

    private BlockingQueue<String> readQueue = null;

    private BlockingQueue<String> writeQueue = null;

    public Worker(BlockingQueue<String> readQueue, BlockingQueue<String> writeQueue){
        this.readQueue = readQueue;
        this.writeQueue = writeQueue;
    }

    @Override
    public void run() {
        try {
            while(true) {
                String article = readQueue.take();
                if(article.equals(WikiCleaner.END)){
                    readQueue.put(WikiCleaner.END);
                    writeQueue.put(WikiCleaner.END);
                    break;
                }
                writeQueue.put(cleanArticle(article));
            }
        } catch(InterruptedException e) {
            e.printStackTrace();
        }
    }

    private String cleanArticle(String article) {
        article = NULLCHAR.matcher(article).replaceAll("");
        article = URLS.matcher(article).replaceAll("");
        article = PARENTHESES.matcher(article).replaceAll("");
        article = SYMBOLS.matcher(article).replaceAll("");
        article = WHITESPACE.matcher(article).replaceAll(" ");
        return article;
    }
}
```

```java
public class WikiCleaner {

    protected final static int WORKERAMOUNT = 4;

    protected final static String END = "END";

    private final static int QUEUESIZE = 10000;

    public void cleanWikiDump(File input, File output) {
        BlockingQueue<String> readQueue = new ArrayBlockingQueue<String>(QUEUESIZE);
        BlockingQueue<String> writeQueue = new ArrayBlockingQueue<String>(QUEUESIZE)

        Reader reader = new Reader(readQueue, input);
        Writer writer = new Writer(writeQueue,output);

        Worker[] workers = new Worker[WORKERAMOUNT];
        for(int i = 0; i<workers.length; i++) {
            workers[i] = new Worker(readQueue,writeQueue);
        }

        new Thread(reader).start();
        for(int i = 0; i<workers.length; i++) {
             new Thread(workers[i]).start();
        }
        new Thread(writer).start();
    }

    public static void main(String[] args) {
        File input = new File("resources/enwiki-20171103-pages.tsv");
        File output = new File("resources/out.txt");

        WikiCleaner cleaner = new WikiCleaner();
        cleaner.cleanWikiDump(input, output);
    }


}
```

Coreference Resolution

- Stanford NLP CoreRef to find the representative mention
- Replace pronouns and possessive pronouns

Example

- John met Judy in 1960. He married her during his college year.

  $\Rightarrow$ John met Judy in 1960. John married Judy during John's college year.

Named Entity Recognition

- Requesting Spotlight Demo and parsing JSON response to Entity class

```
public class Entity {

    private ArrayList<String> types = new ArrayList<String>();

    private String uri;

    private String surfaceForm;

    private int offset;

    public Entity(String types, String uri, String surfaceForm, int offset) {
        this.toList(types);
        this.setUri(uri);
        this.setSurfaceForm(surfaceForm);
        this.setOffset(offset);
    }

    private void toList(String t) {
        String[] comma = t.split(",");
        for(String typeLink: comma) {
            String[] type = typeLink.split(":");
            if(type.length == 2) types.add(type[1]);
        }
    }

    public ArrayList<String> getTypes() {
        return types;
    }

    public String getUri() {
        return uri;
    }

    public void setUri(String uri) {
        this.uri = uri;
    }

    public String toString() {
        return "URI: " + uri + "- Type:" + types + "- Surface Form: " + surfaceForm;
    }

    public String getSurfaceForm() {
        return surfaceForm;
    }

    public void setSurfaceForm(String surfaceForm) {
        this.surfaceForm = surfaceForm;
    }

    public int getOffset() {
        return offset;
    }

    public void setOffset(int offset) {
        this.offset = offset;
    }
}
```

Entity Disambiguation

Frameworks

- Spotlight: Integrated disambiguation with two approaches:
  - The information (context) next to a candidate's surface forms is used to find the most likely disambiguation. The best match determines the selection.
  - Weigh words on their ability to disambiguate between the resources

  [Mendes, Jakob, Garca-Silva, Bizer, 2011]
- Fox: AGDISTIS

$\Rightarrow$ Done by given Frameworks

Relation Extraction

- Two ways of extracting Relations:
  - Fox in 're' mode, performing entity recognition as well as relation extraction
    - Saving triple statements
  - Our RelationExtraction Method using OpenIE and Spotlight
- Limited to the first part of each article (summarizes the facts)

Relation Extraction: Fox

```
private void getTriple(Statement statement, StmtIterator iterator) {
    Resource subject = ResourceFactory.createResource(statement.getObject().toString());
    Statement next = iterator.next();
    Property predicate = ResourceFactory.createProperty(next.getObject().toString());
    next = iterator.next();

    RDFNode object = null;
    if(next.getObject().isResource()) {
         object = ResourceFactory.createResource(next.getObject().toString());
    } else {
        object = ResourceFactory.createStringLiteral(next.getObject().toString());
    }
    Statement triple = ResourceFactory.createStatement(subject, predicate, object);
    System.out.println(triple);
    graph.add(triple);
}


private void getRelations(String article) throws MalformedURLException, ProtocolException, IOException, ParseException {
    count = new AtomicInteger(0);

    final List<CoreMap> sentences = PARSER.getSentences(article);
    final ScheduledExecutorService executor = Executors.newScheduledThreadPool(1);

    Runnable askFox = new Runnable() {
        public void run() {
            try {

                Model model = ModelFactory.createDefaultModel() ;
                System.out.println(sentences.get(count.get()));
                model.read(new ByteArrayInputStream(SERVICE.extract(sentences.get(count.get()).toString(), "en" , "re").getBytes()),null, "TTL");
                StmtIterator iterator = model.listStatements();

                while(iterator.hasNext())
                {
                    Statement s = iterator.next();
                    if(s.getPredicate().toString().contains("subject")) {
                        getTriple(s , iterator);
                    }
                }

                if(count.incrementAndGet() == sentences.size()) {
                    executor.shutdownNow();

                    StmtIterator foundTriples = graph.listStatements();

                    while(foundTriples.hasNext())
                    {
                        System.out.println(foundTriples.next());
                    }
                }
            } catch (Exception e) {
                executor.shutdownNow();
                e.printStackTrace();
            }
        }
    };

    executor.scheduleWithFixedDelay(askFox, 0, DELAYSECONDS, TimeUnit.SECONDS);
}
```

Relation Extraction: Own Approach

- Parsing ontology and creating a list of properties
- Using Spotlight for NER, so we can run it concurrent to Fox
- Using Stanford CoreNLP
  - experimented with different approaches to find relations given two entities (search algorithms on dependency trees, semanticGraph)
  - decided to use OpenIE

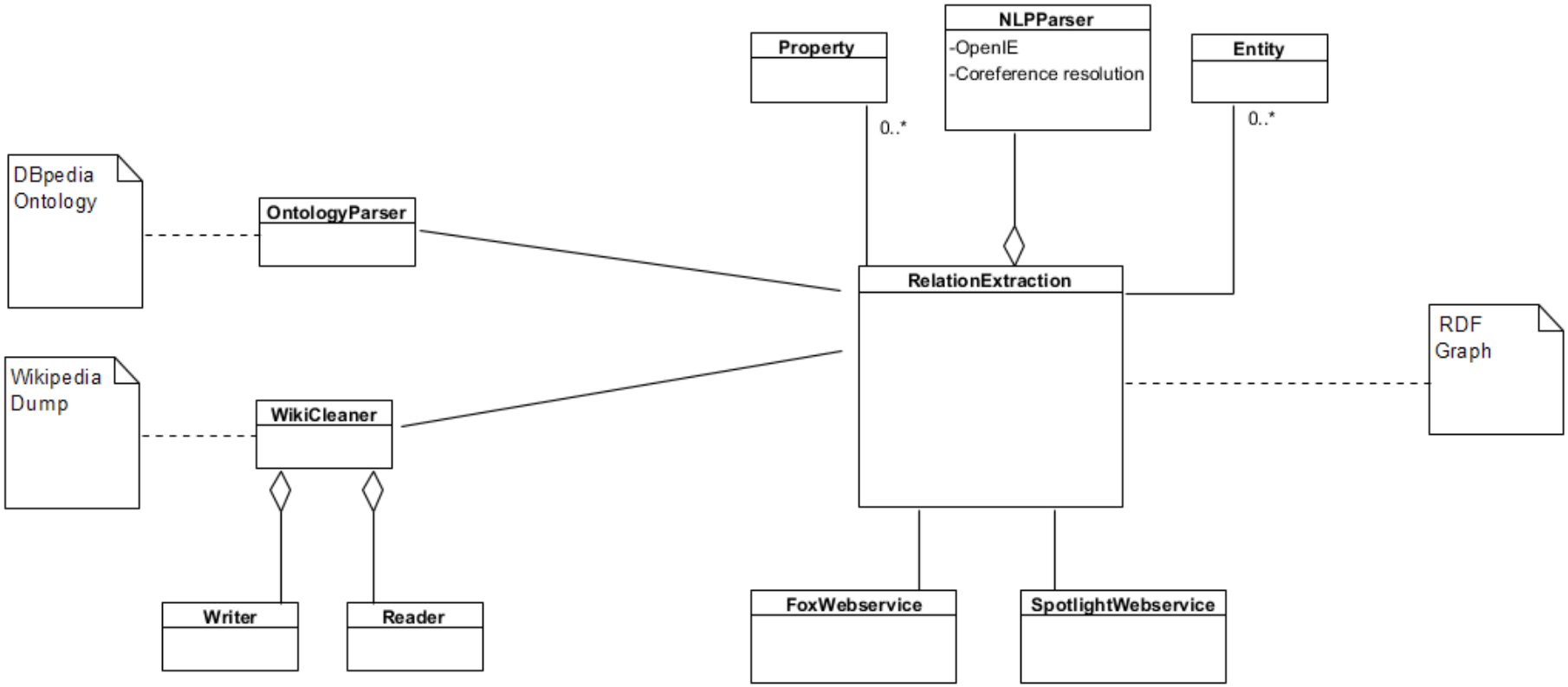
Relation Extraction: OpenIE

- Splitting sentences into shorter fragments, appeal to natural logic to maintain context
- Traverse dependency parse tree recursively
  [Angeli, Premkumar, Manning, 2015]
- Disadvantage in our case: have to filter for entity to entity relation
- After finding the binary relation (OpenIE Triple) between entities: Map them to DBpedia ontology

Relation Extraction: OpenIE

- Parse DBpedia properties to Java objects
- Search for a valid property (check domain and range)
- Map relation to keywords to find proper property
- Write triple to graph

Architecture



Upcoming Task

- Already able to build a knowledge graph, but thinking about further improvements for our own relation extraction
  - e.g. more keywords / heuristics for finding the best property
  - e.g. find entity to literal relations
  - long runtime due to response time of Fox/Spotlight and delay for scheduling Fox/Spotlight requests
    $\Rightarrow$ avoid running it before all improvements are made
- Benchmarking as soon as we are pleased with the result (whether improvements are needed)

References

Pablo N. Mendes, Max Jakob, Andrs Garca-Silva and Christian Bizer (2011)

DBpedia Spotlight: Shedding Light on the Web of Documents

Proceedings of the 7th International Conference on Semantic Systems (I-Semantics) 3.

Gabor Angeli, Melvin Johnson Premkumar, and Christopher D. Manning (2015)

Leveraging Linguistic Structure For Open Domain Information Extraction

In Proceedings of the Association of Computational Linguistics (ACL) 2.