NLP Lean Programming Framework: Developing NLP Applications More Effectively

Marc Schreiber
FH Aachen
Jülich, Germany
marc.schreiber@fh-aachen.de

Bodo Kraft
FH Aachen
Jülich, Germany
kraft@fh-aachen.de

Albert Zündorf
University of Kassel
Kassel, Germany
zuendorf@uni-kassel.de

Abstract

This paper presents NLP Lean Programming framework (NLPf), a new framework for creating custom natural language processing (NLP) models and pipelines by utilizing common software development build systems. This approach allows developers to train and integrate domain-specific NLP pipelines into their applications seamlessly. Additionally, NLPf provides an annotation tool which improves the annotation process significantly by providing a well-designed GUI and sophisticated way of using input devices. Due to NLPf's properties developers and domain experts are able to build domain-specific NLP applications more efficiently. NLPf is Open-source software and available at https://gitlab.com/schrieveslaach/NLPf.

1 Introduction

Nowadays more and more business models rely on the processing of natural language data, e. g. companies extract relevant eCommerce data from domain-specific documents. The required eCommerce data could be related to various domains, e. g. life-science, public utilities, or social media, depending on the companies' business models.

Furthermore, the World Wide Web (WWW) provides a huge amount of natural language data that provides a wide variety of knowledge to human readers. This amount of knowledge is unmanageable for humans and applications try to make this knowledge more accessible to humans, e. g. Treude and Robillard (2016) make natural language text about software programming more accessible through a natural language processing (NLP) application.

All these approaches have in common that they require domain-specific NLP models that have been trained on a domain-specific and annotated corpus. These models will be trained by using different NLP frameworks and these models have to be evaluated for every annotation layer. For example, named entity recognition (NER) of Stanford CoreNLP (Manning et al., 2014) might work better than NER of OpenNLP (Reese, 2015, Chapter 1); the chosen segmentation tool, e. g. UDPipe (Straka and Straková, 2017), might work better than Stanford CoreNLP's segmentation tool, and so on. Existing studies show that domain specific training and evaluation is a common approach in the NLP community to determine the best-performing NLP pipeline (Buyko et al., 2006; Giesbrecht and Evert, 2009; Neunerdt et al., 2013; Omran and Treude, 2017).

Developers of NLP applications are forced to create domain-specific corpora to determine the best-performing NLP pipeline among many NLP frameworks. During this process they face various obstacles:

- The training and evaluation of different NLP frameworks requires a lot of effort of scripting or programming because of incompatible APIs.
- Domain experts who annotate domain-specific documents with a GUI tool struggle with an insufficient user experience.
- There are too many combinations how developers can combine these NLP tools into NLP pipelines.
- The generated NLP models as a build artifact have to be integrated manually into the application code.

NLP Lean Programming framework (NLPf) addresses these issues. NLPf provides a standardized project structure for domain-specific corpora (see Section 2), an improved user experience for annotators (see Section 3), a common build process to train and evaluate NLP models in conjunc-

Proceedings of NAACL-HLT 2018: Demonstrations, pages 1–5
New Orleans, Louisiana, June 2 - 4, 2018.

tion with the determination of the best-performing NLP pipeline (see Section 4), and a convenient API to integrate the best-performing NLP pipeline into the application code (see Section 5).

2 Annotated Corpus Project Structure

Maven as a build management tool has standardized the development process of Java applications by standardizing the build life-cycle, standardizing the project layout, and standardizing the dependency management. These standardization are evolved by utilizing convention over configuration (CoC) as much as possible and developers have to make less decisions while developing software.

Such conventions are missing for the development of domain-specific NLP applications and developers have to make many decisions and have to write many scripts to build their applications. NLPf provides conventions by utilizing Maven and its project object model (POM). Listing 1 shows the basic project configuration to train and evaluate domain-specific NLP models with NLPf.

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>your.company</groupId>
  <artifactId>domain-specific-corpus</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>nlp-models</packaging>

  <build>
    <plugins>
      <plugin>
        <groupId>de.schrieveslaach.nlpf</groupId>
        <artifactId>nlp-maven-plugin</artifactId>
        <version>1.0.0</version>
        <extensions>true</extensions>
      </plugin>
    </plugins>
  </build>
</project>
```

Listing 1: POM of Annotated Corpus

Unlike standard Java projects this project uses the custom packaging method `nlp-models` which configures Maven to use NLPf's plugin (see `nlp-maven-plugin`) which trains and evaluates the domain-specific models. By convention, each document stored in `src/main/corpus` will be used as an input document for the training process and each document stored in `src/test/corpus` will used to evaluate the derived NLP models.

NLPf supports multiple document formats which need to be configured as Maven dependency (see `io-odt` in Listing 2). Most formats supported by DKPro Core[1] (de Castilho and Gurevych, 2014) are supported by NLPf but we recommend to use ODT documents because developers can just paste natural language text into the ODT documents and then annotate them without preparing specific document formats.

```
<dependency>
  <groupId>de.schrieveslaach.nlpf</groupId>
  <artifactId>io-odt</artifactId>
  <version>1.0.0</version>
</dependency>

<dependency>
  <groupId>de.tudarmstadt.ukp.dkpro.core</groupId>
  <artifactId>
    de.tudarmstadt.ukp.dkpro.core.opennlp-asl
  </artifactId>
  <version>1.9.0</version>
</dependency>
```

Listing 2: Dependencies of Annotated Corpus

Additionally, NLPf supports different NLP frameworks which are also provided as Maven dependency (see `de.tudarmstadt.ukp.dkpro.core.opennlp-asl` in Listing 2). NLPf supports all NLP frameworks which provide trainer capabilities of DKPro Core[2]: Stanford CoreNLP, OpenNLP, and LingPipe. When the project has been configured, the annotators can start to annotate the documents.

3 Quick Pad Tagger: Annotate Documents

NLPf provides the annotation tool Quick Pad Tagger (QPT) which provides a well-designed GUI, drawing the attention to the essential GUI elements of the annotation task. Figure 1 provides a screenshot of the QPT, showing how the user annotates named entities (NEs) in a document. At the bottom of the GUI the part of the document will be displayed and at the top of the screen the QPT shows a stream of tokens while the user can select multiple tokens (see blue boxes) to assign a NE type. Through the spinner on top of the stream of tokens the user chooses a type for each of the NEs.

This design has been implemented consequently for each annotation layer and the design draws the attention to the actual important annotation task, e. g. assign NE types or part-of-speech (POS) tags to tokens. Figure 2 compares for the POS tag annotation layers of the state-of-the-art tool Webanno (Eckart de Castilho et al., 2016) and the QPT, showing that the QPT draws the attention

[1] `https://dkpro.github.io/dkpro-core/`

[2] More information is provided here: `https://github.com/dkpro/dkpro-core/pull/1114`

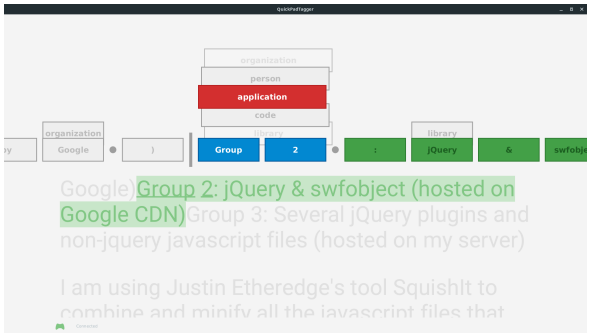
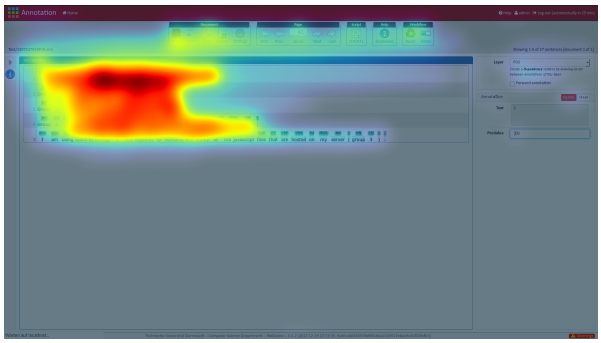
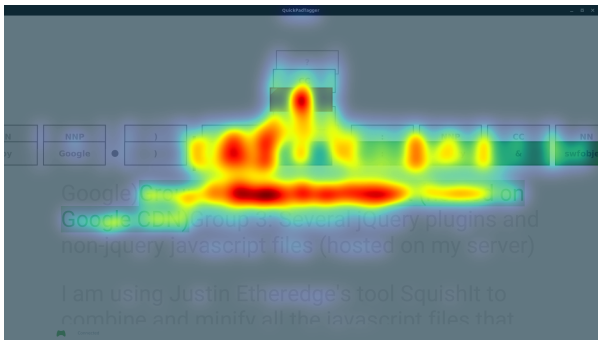


Figure 1: QPT Screenshot: NE Tagging



(a) Webanno Screenshot



(b) QPT Screenshot

Figure 2: Attention Map of POS Tagging Annotation Tools, Obtained Using the EyeQuant Attention Analytics Software (`www.eyequant.com`)

to the annotation task and the document text (the relevant parts) whereas Webanno draws the attention to all annotations at once.

The user can use a Xbox 360 controller to annotate the structure of natural language. This type of input device provides a more comfortable and playful user experience and in conjunction with the GUI design the annotation process is less painful and less exhausting. Additionally, the QPT provides a semi-automatic annotation process (Schreiber et al., 2015) which speeds up the annotation process further. In summary, the QPT reduces the required annotation time by half.

4 Install Best-performing NLP Pipeline Artifact

When documents of the corpus project have been annotated by annotators, developers can use a single command to train all available NLP tools, determine the best-performing NLP pipeline, and create an artifact which will be used in an NLP application (see Section 5). These steps will be performed by `mvn install` and the custom Maven plugin (see `nlp-maven-plugin` in Listing 1) passes following customized life-cycle:

- At first, the Maven plugin validates the annotated documents, for example, it ensures that every or no token of a document have been annotated with a corresponding POS tag.
- After that, the Maven plugin looks up all available NLP trainer classes which are available on the classpath (c. f. `de.tudarmstadt.ukp.dkpro.core.opennlp-asl` in Listing 2). Each discovered trainer class will be used to create a domain-specific NLP model if the required annotations are available and the configuration will be stored in the `target` directory. The configurations are stored in a format compatible to the Unstructured Information Management Architecture (UIMA) framework (Ferrucci and Lally, 2004).
- If NLP tools do not provide any training, e. g. the segmentation tool of Stanford CoreNLP, developers can provide engine factories which create configurations for these tools (see Listing 3) which will be stored in the `target` directory.
- All available configurations will be used to create all possible domain-specific NLP pipeline configurations and each NLP pipeline will be evaluated with F_1 score by running the pipelines on the test documents and by comparing the results on the provided test annotations. The configuration of the best-performing NLP pipeline will be stored into the target directory.
- Based on the previous steps the Maven plugin creates a Java archive (JAR) which contains the NLP models and configuration of the best-performing NLP pipeline.

- Finally, the created JAR artifact can be installed or deployed into any Maven repository.

```
public class StanfordCoreNlpEngineFactory {

  public AnalysisEngineDescription
  ↪  createStanfordSegmenter() throws Exception {
    return createEngineDescription(
        StanfordSegmenter.class,
        StanfordSegmenter.PARAM_LANGUAGE_FALLBACK,
        ↪  "en");
  }
}
```

Listing 3: Engine Factory for NLP Tools Without Training Support

5 API Running the Best-performing NLP Pipeline

When the best-performing NLP pipeline is available as JAR artifact in a Maven repository, developers can integrate this artifact as Maven dependency into the NLP application. Therefore, developers insert the project coordinates of the domain specific corpus into the application's POM, illustrated by Listing 4. Additionally, developers need to add the `plumping` library which provides an API to execute the best-performing NLP pipeline.

```
<dependency>
  <groupId>your.company</groupId>
  <artifactId>domain-specific-corpus</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>

<dependency>
  <groupId>de.schrieveslaach.nlpf</groupId>
  <artifactId>plumbing</artifactId>
  <version>1.0.0</version>
</dependency>
```

Listing 4: Dependencies of NLP Application

The provided API integrates seamlessly into the API of the UIMA framework which provides an interface to run NLP components on unstructured data such as natural language text, c. f. method `runPipeline` in Listing 5. However, the best-performing NLP has to be configured manually. NLPf's `plumping` JAR artifact provides the method `createBestPerformingPipelineEngineDescription()` which reads the configuration of the JAR that contains the configuration and models of the best-performing NLP pipeline.

```
CollectionReaderDescription readerDescription =
↪  createReaderDescription(
    OdtReader.class,
    OdtReader.PARAM_SOURCE_LOCATION, new
    ↪  File("plain.odt"));

AnalysisEngineDescription writerDescription =
↪  createEngineDescription(
    OdtWriter.class,
    OdtWriter.PARAM_TARGET_LOCATION, new File("."),
    OdtWriter.PARAM_OVERWRITE, true);

runPipeline(readerDescription,
    createBestPerformingPipelineEngineDescription(),
    // integrate custom engine descriptions here
    writerDescription);
```

Listing 5: Example Application Java Code

The example code provided in Listing 5 performs following steps, executed by `runPipeline`:

- It reads an ODT file with the name `plain.odt`, c. f. `readerDescription`.
- Then, it runs the best-performing NLP pipeline which annotates the whole document with the natural language structure.
- Finally, it stores the annotations into an ODT file into the current directory, c. f. `writerDescription`.

Developers can integrate custom analyses as they require them (see `// integrate custom...` in Listing 5). Therefore, they need to implement UIMA annotators which use the typesystem of DKPro Core. The conjunction of UIMA, DKPro Core, and NLPf allows developers to implement NLP applications effectively.

6 Summary

This paper provides a demonstration of NLP Lean Programming framework (NLPf) which enables developers to create domain-specific NLP pipelines more effectively, making less decisions through CoC. NLPf provides a standardized environment and the well-designed annotation tool Quick Pad Tagger (QPT) with an improved input mechanism to improve the annotation process. Additionally, the best-performing NLP pipeline will be determine through the convenient build tool Maven and the resulting artifact can be integrated as Maven dependency into any application conveniently.

NLPf is Open-source software, released under the LGPL version 3, and available at `https://gitlab.com/schrieveslaach/NLPf`. All artifacts are available on Maven central and they can also be used with Jython in Python programs.

References

Ekaterina Buyko, Joachim Wermter, Michael Poprat, and Udo Hahn. 2006. Automatically adapting an nlp core engine to the biology domain. In *Proceedings of the Joint BioLINK-Bio-Ontologies Meeting. A Joint Meeting of the ISMB Special Interest Group on Bio-Ontologies and the BioLINK Special Interest Group on Text Data M ining in Association with ISMB*. pages 65–68.

Richard Eckart de Castilho and Iryna Gurevych. 2014. A broad-coverage collection of portable NLP components for building shareable analysis pipelines. In Nancy Ide and Jens Grivolla, editors, *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT at COLING 2014*. Association for Computational Linguistics and Dublin City University, Dublin, Ireland, pages 1–11.

Richard Eckart de Castilho, Éva Mújdricza-Maydt, Seid Muhie Yimam, Silvana Hartmann, Iryna Gurevych, Anette Frank, and Chris Biemann. 2016. A web-based tool for the integrated annotation of semantic and syntactic structures. In *Proceedings of the workshop on Language Technology Resources and Tools for Digital Humanities (LT4DH) at COLING 2016*. pages 76–84.

David Ferrucci and Adam Lally. 2004. Uima: An architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering* 10(3–4):327–348. https://doi.org/10.1017/S1351324904003523.

Eugenie Giesbrecht and Stefan Evert. 2009. Is part-of-speech tagging a solved task? an evaluation of pos taggers for the german web as corpus. In *Proceedings of the 5th Web as Corpus Workshop*. WAC5, pages 27–35.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*. pages 55–60.

Melanie Neunerdt, Bianka Trevisan, Michael Reyer, and Rudolf Mathar. 2013. Part-of-speech tagging for social media texts. pages 139–150. https://doi.org/10.1007/978-3-642-40722-2_15.

Fouad Nasser A Al Omran and Christoph Treude. 2017. Choosing an nlp library for analyzing software documentation: A systematic literature review and a series of experiments. In *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press, Piscataway, NJ, USA, MSR '17, pages 187–197. https://doi.org/10.1109/MSR.2017.42.

Richard M. Reese. 2015. *Natural Language Processing with Java*. Packt Publishing Ltd.

Marc Schreiber, Kai Barkschat, Bodo Kraft, and Albert Zündorf. 2015. Quick Pad Tagger: An Efficient Graphical User Interface for Building Annotated Corpora with Multiple Annotation Layers. *Computer Science & Information Technology* 4:131–143. https://doi.org/10.5121/csit.2015.50413.

Milan Straka and Jana Straková. 2017. Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. Association for Computational Linguistics, Vancouver, Canada, pages 88–99.

Christoph Treude and Martin P. Robillard. 2016. Augmenting api documentation with insights from stack overflow. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, New York, NY, USA, pages 392–403. https://doi.org/10.1145/2884781.2884800.