INDIANA UNIVERSITY
BLOOMINGTON

Natural Language Processing Lab
https://nlp-lab.org/

Unification and Standardization of NLP-pipelines for Computational Semantics using Scalable Microservices

Damir Cavar, Oren Baldinger, Maanvitha Gongalla, Aarushi Bisht, Jagpreet Singh Chawla, Umang Mehta, Murali Kammili, Anurag Kumar, Chaitanya Patil

The NLP-Lab, Indiana University Bloomington

Problem

- **Lack of syntactic standardization** of the output format of Natural Language Processing (NLP) pipelines and components (syntactic variation: CoNLL, JSON, XML, etc.)
- **Lack of semantic standardization** and mapping of linguistic annotations, e.g.
  - Variation of tag-sets and labels: inconsistencies, coverage
  - Variation of tree structures and labels (constituent and dependency trees) based on differences in theories, corpora, data annotation standards
- **Gap between linguistic theory and models, and computational applications** of NLP (e.g. symbolic, probabilistic, and neural models using knowledge or data driven methods)
- **High variation of annotation quality and accuracy:** low precision with higher linguistic levels
- **Performance and scalability issues:** Experimental NLP mostly not robust, fails big data, High Performance Computing, general memory and run-time features for productive environments

Objective

- **Standardization of outputs** from different NLP pipelines
  - **Format Normalization:** Syntactic or format standardization (format normalization)
  - **Semantic Normalization:** Standardization of labels, tree formats, annotation mapping
- **Inclusive standard** to cover:
  - **Common data-driven approaches** with shallow annotations
  - **Advanced theoretically driven approaches** with deep linguistic annotation
- **Increased annotation quality and accuracy control** of NLP output
- **High Performance and scalable architecture and environment**

Solution

- **Standardization of outputs** using JSON-NLP (github.com/dcavar/JSON-NLP)
  - **Format Normalization** in JSON mapping common NLP-pipeline outputs
  - **Semantic Normalization** translation of labels, structures, annotations into broad coverage linguistic annotation schema
  - **Expansion of Annotations** translation of encoded linguistic information into atomic features, API-capable data-structures
- **Inclusive standard** covers:
  - **Common data-driven** treebank, PoS, entity annotation standards
  - **Advanced theoretical concepts** as for example empty categories, implied entities, gapping, ellipsis, traces
- **Increased annotation quality and accuracy control** using Unification of JSON-NLP structures
- **Performance and scalability:**
  - **Microservice Architecture**
  - **Process Optimization** via model caching, load optimization
  - **Big Data and High Performance Computing** adaptation using scalable Java-EE and WSGI environment, Hadoop and Spark capabilities, Stream-processing capable (Kafka, Twister 2)
  - **RESTful API** for broad language and architecture compatibility

Acknowledgements

Contributors: Yiwen Zhang, Shreejith Panicker, Aarnav, Abhishek Babuji, Gopal Seshadri, Shujun Liu, Peace Han, Rohit Bapat, Mahesh Latnekar, Boli Fang, Stefan Geißler, Joshua Herring.

JSON-NLP Schema

- **Annotation sections:**
  - Meta data: NLP pipeline, document, corpus, processing parameters like date, time, environment
  - Corpus information: multi-document architecture
  - Linguistic and NLP annotation:
    - Document and paragraph level segmentation, coreference analysis, anaphora resolution
    - Sentence level annotation: complexity, chunks, parse trees (constituent, dependency), semantic scope relations, voice, mood, tense, sentiment, speaker, etc.
    - Clause level segmentation and annotation: as for sentences, includes scope relations among clauses
    - Entity and Multi-Word annotation: entity, role, type, link to Knowledge Graphs
    - Token-level annotation: PoS, morpho-syntactic features, semantic properties, overt and implicit tokens, semantic scope relations, etc.
- **Validation Mechanisms** and implementation in Python, Java, etc.
- **Optimization of JSON-NLP Object Instantiation** for ease of implementation and performance optimization

JSON-NLP Pipelines as Microservices

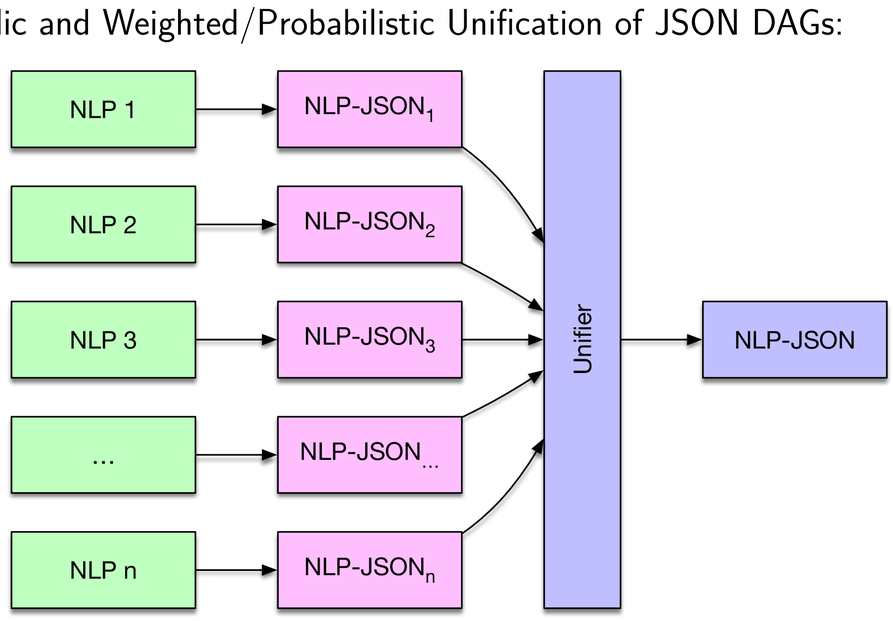
Microservice architecture based on Java-EE (JBoss/WildFly) and WSGI architecture.

- **Available Common Pipelines and Technologies:**
  - Java-based: OpenNLP, LingPipe, CoreNLP, MaltParser, Apache Jena, YAGO Knowledge Graph, Microsoft Concept Graph
  - Python-based: NLTK, spaCy, Flair, Polyglot, Xrenner, ConceptNet interface, ...
- **Our NLP Technologies:**
  - FST-based Morphologies (Java-JNI or Python interface)
  - Clause level segmentation and feature annotation: tense, voice, aspect, etc.
  - Advanced Parsing (using theoretical frameworks like LFG, HPSG, CxG)
  - Computational Semantics and Pragmatics Processing: Implicatures and Presuppositions
  - JSON-NLP Unification

See numerous *GitHub* repos (e.g. JSON-NLP, J-JSON-NLP, Py-JSON-NLP) and *PyPi* modules (e.g. polyglotjsonnlp, nltkjsonnlp, xrennerjsonnlp, flairjsonnlp, spacyjsonnlp, pyjsonnlp).

Unification of JSON-NLP Outputs

Symbolic and Weighted/Probabilistic Unification of JSON DAGs:



- Validation of generated linguistic annotations and scores, cross-comparison over NLP pipelines and technologies
- Computation of confidence scores based on ensembles of NLP pipelines
- Merger of NLP-output annotations and analyzes to uniform broad-coverage output