CogCompNLP: Your Swiss Army Knife for NLP

Daniel Khashabi[1], Mark Sammons[1], Ben Zhou[2] , Tom Redman[2]
Christos Christodoulopoulos[3] , Vivek Srikumar[4] , Nicholas Rizzolo[5] , Lev Ratinov
Guanheng Luo[2] , Quang Do[6] , Chen-Tse Tsai[7] , Subhro Roy[8] , Stephen Mayhew[1]
Zhili Feng[9] , John Wieting[10] , Xiaodong Yu[2] , Yangqiu Song[11] , Shashank Gupta[2]
Shyam Upadhyay[1] , Naveen Arivazhagan[5] , Qiang Ning[2] , Shaoshi Ling[2] , Dan Roth[1,2]

[1]Univ. of Pennsylvania [2]Univ. of Illinois at Urbana-Champaign [3]Amazon Research Cambridge [4]Univ. of Utah [5]Google
[6]Anduin Transactions [7]Bloomberg L.P. [8]MIT CSAIL [9]Univ. of Wisconsin-Madison [10]Carnegie Mellon University [11]HKUST

Abstract

Implementing a Natural Language Processing (NLP) system requires considerable engineering effort: creating data-structures to represent language constructs; reading corpora annotations into these data-structures; applying off-the-shelf NLP tools to augment the text representation; extracting features and training machine learning components; conducting experiments and computing performance statistics; and creating the end-user application that integrates the implemented components. While there are several widely used NLP libraries, each provides only partial coverage of these various tasks. We present our library CogCompNLP which simplifies the process of design and development of NLP applications by providing modules to address different challenges: we provide a corpus-reader module that supports popular corpora in the NLP community, a module for various low-level data-structures and operations (such as search over text), a module for feature extraction, and an extensive suite of annotation modules for a wide range of semantic and syntactic tasks. These annotation modules are all integrated in a single system, Pipeline, which allows users to easily use the annotators with simple direct calls using any JVM-based language, or over a network. The sister project CogCompNLPy enables users to access the annotators with a Python interface. We give a detailed account of our system's structure and usage, and where possible, compare it with other established NLP frameworks. We report on the performance, including time and memory statistics, of each component on a selection of well-established datasets. Our system is publicly available for research use and external contributions, at: `http://github.com/CogComp/cogcomp-nlp`.

1. Motivation

Natural Language Processing (NLP) is one of the fastest-growing fields both in research and industrial work. Tools that provide analysis of text by identifying syntactic and semantic elements (e.g. named entity recognizers, syntactic parsers, semantic role labelers) are widely used - typically many such components in combination - as inputs to a more specialized (and complex) application. However, the process of managing and aggregating these tools and their inputs and outputs is typically labor-intensive and error-prone, requiring significant engineering effort before the target application can be built and evaluated. With the introduction of new, complex tasks such as event detection or cross-document coreference, and the increasing commercial demand for text analysis, it is critical to build software frameworks that give easy access to a wide range of existing NLP annotators and support straightforward extension to others.

We introduce CogCompNLP, an NLP ecosystem published under an academic-use license, which is designed for management, aggregation, and application of NLP analytic components. It comes with a suite of NLP components for syntactic and shallow semantic analysis, but also word and phrase similarity metrics. It provides essential support for text processing applications, including classes for text cleaning and for reading a number of popular NLP corpora. In addition to describing some key functionalities, we illustrate the use of CogCompNLP components in developing a Semantic Role Labeling application: reading data from a corpus; augmenting the resulting data-structures with NLP components using the NLP pipeline; extracting features for input to machine learning algorithms; training classifiers using LBJava –another CogComp project (Rizzolo and Roth, 2010); serializing the system outputs; and adding the new SRL application to the pipeline for other applications to use.

2. Terminology

The CogCompNLP framework builds on the conceptual design and data-structures described in (Clarke et al., 2012; Sammons et al., 2016). Here we give a brief summary of the main structures and keywords used. A View is a data-structure which contains an annotation structure of a text; examples are tokens, lemmas or dependency parse trees. An Annotator is a class which produces a View given a text, and potentially some other Views. The main data-structure used is TextAnnotation, which contains a document (e.g. a phrase, a sentence, a paragraph) and its various Views.

3. Framework Design

A high-level view of the system is depicted in Figure 1. The boxes show modules and edges show the dependencies between them (with the targets being the dependencies). We describe each of these modules in the following sections.

Core Utilities. Contains the fundamental data-structures and operators; hence many of the other modules depend on it. A selection of important basic functionalities supported by this module are:

- SQL-like operations on TextAnnotation for extracting patterns
- Experiment utilities, such as P/R/F1 reporting, statistical significance testing and cross-validation helpers
- String pattern-matching algorithms
- Utilities for reading and writing files, resources and annotations.

		Sentence splitting	Tokenizing	Lemmatizing	Part of Speech tagging	Shallow Parsing	NER (4 labels)	Extended NER (15-20 labels)	Constituency Parsing	Dependency Parsing	Quantity Normalization	Verb-sense Classification	Temporal Normalization	Mention Detection	Comma SRL	Preposition SRL	Propbank (verb) SRL	Nombank (nominal) SRL	Coreference resolution	Relation Extraction	Sentiment	OpenIE	Wikifier
	CogCompNLP (ours)	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓
Java	CoreNLP	✓	✓	✓	✓		✓	✓	✓	✓			✓	✓					✓	✓	✓	✓	
Java	OpenNLP	✓	✓	✓	✓	✓	✓		✓										✓				
Python	SpaCy	✓	✓	✓	✓	✓	✓	✓	✓	✓													
Python	NLTK	✓	✓	✓	✓	✓	✓																
Python	TextBlob	✓	✓	✓	✓	✓															✓		

Table 1: Summary of major annotators included in well-known NLP packages,

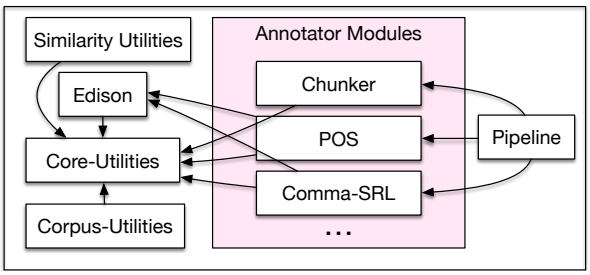


Figure 1: Diagram of the components in CogCompNLP and how they depend on each other. Each module to which an arrow is pointing at, is a dependee project. The arrows indicate the direction of dependence; for example Chunker depends on Core-Utilities.

The key data-structures for interaction between components – TextAnnotation, View, and Constituent – are illustrated in Figure 2. The TextAnnotation contains the raw source text together with its tokenization and other annotation layers added either by reading corpora annotations or applying NLP components. Token indexes are used to align constituents from different views, but each Constituent tracks its character offsets in the raw text. The figure shows Part-of-Speech, Named Entity, Quantity, and Semantic Role Label annotations, each in a separate View.

Corpus Utilities. While often overlooked and purely an engineering effort, implementing code that reads data correctly and efficiently can be a time-consuming task. CogCompNLP's corpus reader module includes NLP corpus readers that populate TextAnnotation objects. A few of the important datasets supported by this module are:

- Propbank Semantic Role Labeling
- Treebank Shallow Parse
- PennTreebank Constituency Parse
- Nombank Semantic Role Labeling
- ACE 2004/2005 Named Entity, Relation, Co-reference, and Event
- Ontonotes 5.0 POS, Named Entity, Syntactic Parse, and Semantic Role Labeling
- TAC/ERE Event, Relation, and Named Entity

Similarity Utilities. For calculating semantic similarity between words, phrases, and entities using both structured and distributional representations. Each similarity function compares objects (words, phrases, named entities, sentences) and returns a score indicating how similar they are. Depending on the inputs, different algorithms are available:

- *Word Similarity:* For computing the similarity between two words. The following representations are currently supported: `word2vec` (Mikolov et al., 2013), `paragram` (Wieting et al., 2015), `esa` (Gabrilovich and Markovitch, 2007), `glove` (Pennington et al., 2014), `wordnet` (Do et al., 2009), `phrase2vec` (Yin and Schütze, 2014). Here is a sample usage:

```
String representation = "esa";
WordSim ws = new WordSim(representation);
ws.compare("word", "sentence"); // 0.37
```

- *Named-Entity Similarity:* Comparing named entities requires a different class of algorithm. CogCompNLP's current algorithm is based on (Do et al., 2009):

```
NESim nesim = new NESim();
nesim.compare("Donald Trump", "Trump"); // 0.9
```

- *Phrasal Similarity:* Algorithms to combine lexical-level systems to make sentence-level decisions (Do et al., 2009):

```
Metric llm = new LLMStringSim(config);
String s1 = "Jack bought Alex's car";
String s2 = "Alex sold his car to Jack.";
llm.compare(s1, s2); // 0.75
```

Edison. Feature extraction is a crucial element in design of NLP systems. Edison (Sammons et al., 2016) is a feature extraction framework that uses the data-structures of CogCompNLP core-utilities to extract features to be used by machine learning algorithms. Edison enables users to define feature extraction functions that take as input the Views and Constituents created by CogCompNLP's Annotators. This makes it possible to not only develop feature sets like words, n-grams, and paths in parse trees, which work with a single View, but also more complex features that combine information from several Views.

This library has been successfully used to facilitate the feature extraction for several higher level NLP applications like Semantic Role Labeling (Punyakanok et al., 2005), Coreference resolution (Rizzolo and Roth, 2016) and, Textual Entailment (Sammons et al., 2010), which use information across several Views over text to make a decision.

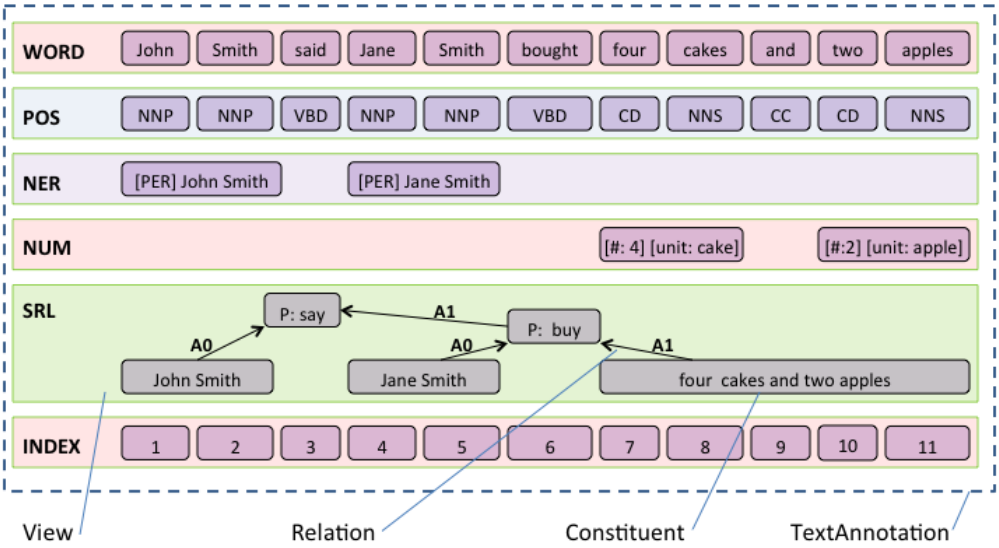


Figure 2: Illustration of the TextAnnotation, View, Constituent, and Relation data-structures in the Core Utilities module.

```
// assume 'srlTa' is a partially annotated text that comes from an earlier step
TextAnnotation srlTa = ...
AnnotatorService pipeline = PipelineFactory.buildPipeline(ViewNames.POS, ViewNames.NER_CONLL);
TextAnnotation augmentedSrlTa = pipeline.annotateTextAnnotation(srlTa);
List<Constituents> list = augmentedSrlTa.getView(ViewNames.POS).getConstituents();
System.out.println(list);  // (NNP Pierre) (NNP Vinken) (, ,) (CD 61) (NNS years) (JJ old) (, ,)...
```

Figure 3: Code snippet for using pipeline

Annotators. Given a set of corpus readers that load input data into unified data-structures, we build NLP annotators which share many features from EDISON and train with LBJAVA (Rizzolo and Roth, 2010) or Illinois-SL (Chang et al., 2015). This means that they are very easy to retrain or adapt to new domains and new languages. COGCOMPNLP includes a wide range of annotators and supporting functionality. Many of these annotators are state-of-the-art and are widely usable across different tasks.

The complete list of syntactic and semantic annotations we currently support is provided in Table 1, which also shows NLP components of other NLP frameworks. To the best of our knowledge this is the only NLP framework with such a large variety of syntactic and semantic components (in comparison, Stanford's CORENLP offers 14 components, OPENNLP 8, SPACY 9).

All of the the modules in our systems are actively updated to improve their quality, while maintaining their interoperability. These systems typically began as individual projects in previous work and have evolved over the years. In some cases the performance of components has been reduced for engineering purposes (e.g. faster speed, smaller memory footprint, etc). In Table 2 we present a qualitative assessment of the major components in COGCOMPNLP. The components have state-of-the-art quality or very close to the best existing results.

Pipeline. With all the Annotators generating the same data-structures, the PIPELINE project provides a simple interface to access Annotator components either individually or as a group, with a single function call. Use of PIPELINE is illustrated in Figure 3. A demo of PIPELINE is accessible online at `http://nlp.cogcomp.org`.

Accessibility from other programming languages. The majority of the implementation is in Java and hence COGCOMPNLP is easily accessible to JVM-based languages with direct calls.

To go beyond memory limitations of making direct calls to PIPELINE and make it available to other programming languages, the PIPELINE can be made accessible over a network. Using an internal web-server, PIPELINE can be made available over a network, making it accessible for a variety of programming languages. Essentially a user can instantiate an instance of PIPELINE server on a single machine (with sufficient memory), which can be shared by many users and queried from different languages.

We have created a Python interface, COGCOMPNLPY[1] which works with direct calls (using PyJnius and Cython (Behnel et al., 2011)) as well as over-network calls to the Java back-end. Here is an example snippet showing how to annotate a sentence with COGCOMPNLPY:

```
from ccg_nlpy import remote_pipeline
pipeline = remote_pipeline.RemotePipeline()
text = "Hello, how are you. I am doing fine"
ta = pipeline.doc(text)
print(ta.get_pos)
# (UH Hello) (, ,) (WRB how) (VBP are) (PRP you)...
```

4. Related Work

One important aspect of our work is the collection of the major NLP annotators. Table 1 contains a summary of components that exist in other well-established NLP libraries. CORENLP (Manning et al., 2014) is a popular

[1] `https://github.com/CogComp/cogcomp-nlpy`

Task	Dataset	Measure	Setting	Result
Tokenization	MASC (Ide et al., 2010)	Accuracy	–	97
POS (Roth and Zelenko, 1998)	Penn Treebank (Bies et al., 2015)	F1	–	96.13
NER (Ratinov and Roth, 2009; Redman et al., 2016; Tsai and Roth, 2016)	CoNLL (T. Kim Sang and De Meulder, 2003)	F1	–	91.12
	Ontonotes (Hovy et al., 2006)	F1	–	84.61
	MUC-7 (Chinchor, 1998)	F1	–	88.37
	Enron (Shetty and Adibi, 2004)	F1	–	77.21
	TAC-KBP 2016 EDL shared task	F1	English	88.3
		F1	Spanish	85
		F1	Chinese	79.3
Shallow Parse (Punyakanok and Roth, 2000)	CoNLL 2000 (Sang and Buchholz, 2000)	F1	–	93.58
Temporal Normalization (Zhao et al., 2012)	TempEval3 (UzZaman et al., 2013)	Exact match F1 / Relaxed match F1	Temporal Span Extraction	79.35/ 83.4
		F1	Temporal normalization, given a predicted temporal span	70.45
Mention Detection	ACE-05 (Walker et al., 2006)	F1	Head detection	89.6
		F1	Boundary detection given the head	89.45
	ERE	F1	Head detection	81.7
		F1	Boundary detection given the head	88.74
Relation Extraction (Chan and Roth, 2011)	ACE 2005 (Walker et al., 2006)	F1	Gold mention - Coarse Type	62.54
		F1	Gold mention - Fine Type	58.35
Taxonomic Relations (hypernyms, hyponyms, and co-hypernyms)	Test-I of Do and Roth (2012)	Accuracy	–	86.1
Comma SRL (Arivazhagan et al., 2016)	(Arivazhagan et al., 2016)	F1	–	83.6
Preposition SRL (Srikumar and Roth, 2013)	(Srikumar and Roth, 2013)	F1	–	90.26
Verb SRL (Punyakanok et al., 2004)	PropBank (Palmer et al., 2005a)	F1	–	76.22
Nominal SRL(Punyakanok et al., 2004)	NomBank (Meyers et al., 2004)	F1	–	66.97
Coreference (Samdani et al., 2014)	CoNLL-12 (Pradhan et al., 2012)	Average of F1 score of MUC, B^3	Gold mentions	77.05
			Predicted Mentions	60
	ACE-04 (Doddington et al., 2004)	B^3	Gold mentions	79.42
			Predicted Mentions	68.27
Wikifier (Tsai and Roth, 2016)	TAC-KBP 2016 EDL shared task	F1	English: NER+ link to freebase	76.5
		F1	Spanish: NER+ link to freebase	75.1
		F1	Chinese: NER+ link to freebase	70.6

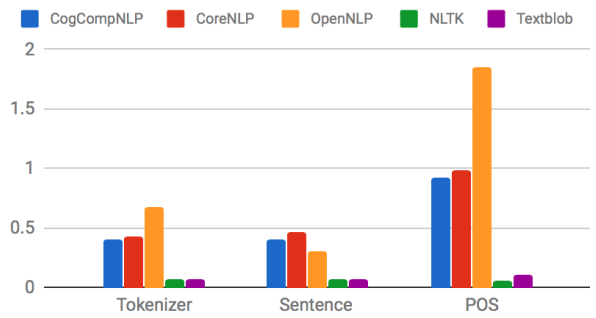
Table 2: Qualitative evaluation of major components included in CogCompNLP.

library which contains various NLP components. While missing a few modules (OpenIE and Constituency parsing), CogCompNLP contains high-level annotators for Semantic Role Labeling of verbs, nouns, commas, and prepositions, while also offering a much wider range of supporting functionality. Compared to OpenNLP (Baldridge, 2005), SpaCy[2], NLTK (Bird, 2006) and TextBlob (Loria et al., 2014), CogCompNLP has many extra annotators. While CogCompNLP is implemented in Java, its sister project CogCompNLPy makes it accessible in Python.

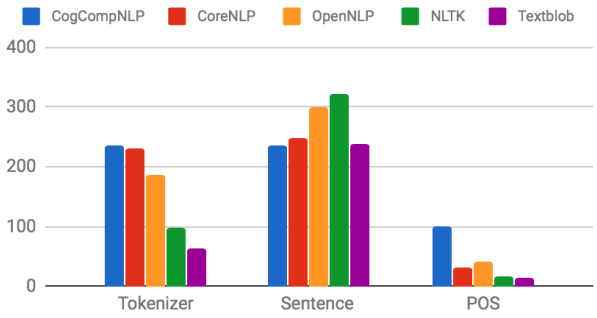
Memory and speed comparison. To measure speed and memory, we compile a collection of English raw text files extracted from a news corpus (NYT files from English Gigaword v5). The collected plain text corpus has 999 files, 665233 words (using linux `'wc -w'` command).
To measure time and memory, we use the GNU `time` command, which is programming-language agnostic. It reports the time that it takes to run the program from start to end, and it also captures the maximum resident set size of the process during its lifetime. In reporting time we use three definitions: (a) *wall-clock* time is the time that a clock on the wall (or a stopwatch in hand) would measure as having elapsed between the start and end of the process. (b) *user time* is the amount of time spent in user code, and (c) *system time* the amount of time spent in the kernel. For each of the systems we use their latest available version (summarized in Table 3).

Often NLP systems are implemented as a sequence of interdependent components, and one would ideally measure their specifications using only the subsequence up to and including the component being evaluated. However, not all systems have the same ordering of the components internally. In what follows, we show the speed/memory it takes

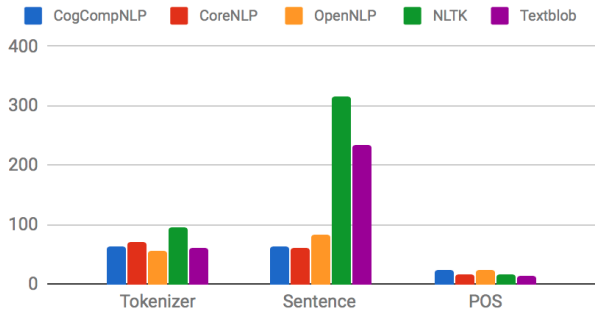
[2] `https://spacy.io`



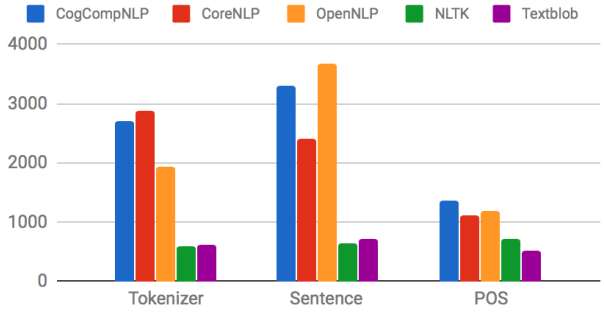
(a) Memory (GB): **lower is better**. Python-based tools have lower memory footprint.



(b) Speed: Wall clock time (thousand tokens / second): **higher is better**.



(c) Speed: User time (thousand tokens / second): **higher is better**.



(d) Speed: System time (thousand tokens / second): **higher is better**.

Figure 4: Speed and memory comparison between major NLP pipelines. SPACYis not shown in any of the figures, since its components are not easy to separate.

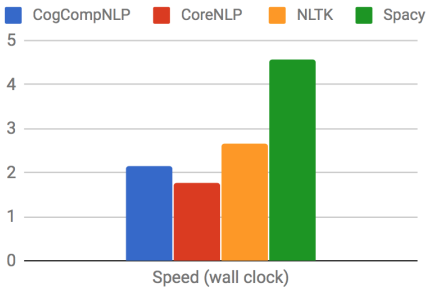


Figure 5: Speed: Wall clock time (thousand tokens / second): **higher is better**. The systems are set to produce *Tokenizing, Sentence Splitting, POS tagging and Named Entity Recognition*. TEXTBLOB is not presented since it does not have an NER.

	Tool	Version
Software	COGCOMPNLP	4.0.2
	CORENLP	3.8.0
	SPACY	2.0.5
	NLTK	3.2.5
	TEXTBLOB	0.15.1
	Maven	3.5.2
	Java	1.8.0-151
	Python	3.5.2
Hardware	CPU	12 × Intel Xeon six core 3.2GHz
	Memory	32GB

Table 3: Specs of the software and hardware used in our evaluation. For a fair comparison, we used the same machine to run all components.

to run the component being evaluated, with everything not needed being turned off.

The results of the speed evaluation for the three tasks of sentence-splitting, tokenizing, and POS tagging are summarized in Figure 4. Overall, JDK-based systems tend to need more memory (subfigure (a)). Differences in speed are often related to the task at hand; for example for POS tagging or tokenizing we do not see any significant difference.

In Figure 4 we do not show any results on SPACYsince unlike other tools, in SPACY we did not find a clean-cut way to study individual modules in isolation, In order to compare other tools with SPACY we perform another experiment where all the systems annotate Tokenizing, Sentence Splitting, POS tagging and Named Entity Recognition together. The results of this speed evaluation is depicted in Figure 5.

5. SRL: A Sophisticated Application

In this section we illustrate the ways COGCOMPNLP supports development and evaluation of complex NLP applications using the task of Semantic Role Labeling (SRL) as an example. SRL identifies predicate-argument structures in

```
String corpusDir = "/local/path/to/ontonotes5.0/corpus/";
OntonotesReader srlReader = new OntonotesSrlReader(corpusDir);
while (srlReader.hasNext()) {
    TextAnnotation documentSrl = srlReader.next();
    PredicateArgumentView srlView = (PredicateArgumentView) documentSrl.getView(ViewNames.SRL_VERB);
    for (Constituent verbPredicate : srlView.getPredicates()) {
        for (Relation argumentRelation : srlView.getArguments(verbPredicate)) {
            String argumentType = argumentRelation.getRelationName();
            Constituent argument = argumentRelation.getTarget();
            ...
    }
}
```

Figure 6: Code snippet for reading OntoNotes and iterating over SRL predicates

```
// assume our new component is called 'MySrlApp'
Annotator mySrlApp = new MySrlApp(); // initialize the SRL annotator
AnnotatorService pipeline = PipelineFactory.buildPipeline(ViewNames.Lemma, ViewNames.POS);
pipeline.addAnnotator(mySrlApp); // add the annotator to an pipeline
String text = "John Smith said Jane Smith bought four cakes and two apples.";
TextAnnotation outputTa = pipeline.createAnnotatedTextAnnotation(text); // annotate with annotators
boolean useJson = true;
SerializationHelper.serializeTextAnnotationToFile(outputTa, "outputFile.json", useJson);
```

Figure 7: Code snippet for adding a new component to the pipeline. The expected output is shown in Figure 2.

text; while nouns, prepositions, and other word types may express predicates we focus here on verb SRL (Palmer et al., 2005b), using the OntoNotes 5.0 corpus (Weischedel et al., 2013). Due to space requirements, we will focus on representative elements of the task in some steps, but the overall illustration should allow other NLP researchers to easily implement the relevant steps for themselves and the tasks they are interested in.

5.1. Reading the Data

The OntoNotes corpus has many layers of NLP annotations stored in a deep directory structure. The CogCompNLP Ontonotes reader loads each document and its annotations into a TextAnnotation data-structure (see the indicated Views in Figure 2, which shows only one representative sentence). The user can iterate over the predicate argument structures to process each in turn as indicated in the code snippet in Figure 6.

5.2. Adding more NLP information

In order to predict SRL structures, we want to use other syntactic and semantic information represented by the source text, such as the outputs from a syntactic parser and a named entity recognizer. Figure 3 illustrates the use of CogCompNLP's Pipeline to add these annotations to the same data-structure returned by the Ontonotes reader in section 5.1.

5.3. Candidate Extraction & Feature Generation

SRL requires a number of component predictions that are assembled to produce the final output. One such component identifies argument boundaries, and a typical approach is to generate a number of candidates and then predict for each whether it is a valid SRL argument. The code snippet in Figure 8 enumerates the syntactic parse constituents and applies Edison feature extractors to generate inputs for a machine learning component.

5.4. Training and Evaluating a Classifier

Now that we have features extracted, we can generate examples for a classifier by comparing candidate boundaries with gold standard SRL argument boundaries to determine the appropriate label. These examples can be collected and passed to a learning component defined using LBJava (Rizzolo and Roth, 2010), as shown in Figure 9. (see (Rizzolo and Roth, 2010) for a detailed overview and examples of LBJava).

The learner's performance can be easily computed using classes from CogCompNLP's core-utilities module. The library provides a range of supporting functionality for cross-validation and computation of experimental statistics that goes well beyond what we can illustrate here.

5.5. Using the New NLP Component

Once a new component/application has been developed, its outputs can easily be made available to other applications either via serialization, or by making the component an Annotator and adding it to the pipeline. Figure 7 shows how these tasks can be done.

6. Conclusion

CogCompNLP is a mature, well-architected, and largely well-documented NLP framework available under an academic license. It has been developed with the twin goals of making it easy to obtain NLP annotations from off-the-shelf components, and developing and evaluating new applications. In this paper we explain the NLP tools it contains; the pipeline application for applying these tools to input text; its word and phrase similarity metrics. In an earlier work (Sammons et al., 2016) we have described its support for feature extraction. We have illustrated its use to develop sophisticated NLP applications by showing how to implement Semantic Role Labeling, a key NLP component.

```java
 /** predicate for selecting candidate predicates (here, verbs) */
private static class IsVerb extends Predicate<Constituent> {
    // expects POS constituent
    @Override
    public Boolean transform(Constituent c) {
        return c.getLabel().startsWith("V");
    }
}

/** extract positive and negative examples for training. */
public static List<Pair<String, Pair<Constituent, Constituent>>> generateExamples(TextAnnotation ta) {
    List<Pair<String, Pair<Constituent, Constituent>>> examples = new ArrayList<>();
    PredicateArgumentView verbSrlView = (PredicateArgumentView) ta.getView(ViewNames.SRL_VERB);
    IQueryable<Constituent> verbs =
            new QueryableList<>(ta.getView(ViewNames.POS).getConstituents()).where(new IsVerb());
    TreeView parseView = (TreeView) ta.getView(ViewNames.PARSE_STANFORD);

    for (Constituent verb : verbs) {
        List<Constituent> allArgCandidates = getArgCandidates(parseView, verb.getStartSpan());
        for (Constituent candidate: allArgCandidates) {
            String label = "negative";
            if (findGoldArgMatch(candidate, verb, verbSrlView)) label = "positive";
            examples.add(new Pair(label, new Pair(candidate, verb)));
        }
    }
    return examples;
}

/** check in SRL View whether candidate argument, predicate match a gold argument, predicate pair */
public static boolean findGoldArgMatch(Constituent candidate, Constituent verb, PredicateArgumentView srlView) {
    List<Constituent> srlArgMatches =
            srlView.getConstituentsMatchingSpan(candidate.getStartSpan(), candidate.getEndSpan());
    Set<Constituent> srlPredMatches = new HashSet<>(
            srlView.getConstituentsMatchingSpan(verb.getStartSpan(), verb.getEndSpan()));
    srlPredMatches.retainAll(new HashSet<>(srlView.getPredicates()));

    for (Constituent srlArgMatch : srlArgMatches)
        for (Relation rel : srlArgMatch.getIncomingRelations())
            if (srlPredMatches.contains(rel.getSource())) return true;

    return false;
}

/** use syntactic parse to identify parse constituents near the predicate */
public static List<Constituent> getArgCandidates(TreeView parseView, int predicateIndex) {
    // consider siblings and child nodes of predicate
    List<Constituent> argumentCandidates = new ArrayList<>();
    Tree<Constituent> t = parseView.getConstituentTree(0);
    Tree<Constituent> predicateTree = t.getYield().get(predicateIndex).getParent();
    Tree<Constituent> predicateParent = predicateTree.getParent();

    for (Tree<Constituent> sibling : predicateParent.getChildren())
        argumentCandidates.add(sibling.getLabel());

    for (Tree<Constituent> child : predicateTree.getChildren())
        if (child != predicateTree) argumentCandidates.add(child.getLabel());

    return argumentCandidates;
}
```

Figure 8: Code snippet for generating verb SRL argument examples.

Acknowledgement

The authors would like to thank all the other contributors to the project. This material is partly based on research sponsored by DARPA under agreement number FA8750-13-2-0008. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. This work was also supported by Contract HR0011-15-2-0025 with the US Defense Advanced Research Projects Agency (DARPA). Approved for Public Release, Distribution Unlimited. This work was partly funded by a grant from the Allen Institute for Artificial Intelligence (allenai.org); by Google; by NSF grant BCS-1348522; and by NIH grant R01-HD054448. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any research sponsors listed above.

References

Arivazhagan, N., Christodoulopoulos, C., and Roth, D. (2016). Labeling the semantic roles of commas. In *Proc. of the Conference on Artificial Intelligence (AAAI)*, 2.

Baldridge, J. (2005). The OpenNLP project. *URL: http://opennlp. apache. org/index. html,(accessed 2 February 2012)*.

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., and Smith, K. (2011). Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39.

```
// a trivial classifier, to return a feature for a predicate-argument structure
public class SrlWordAndPos extends Classifier {
    private WordAndPos wordAndPos = new WordAndPos();
    @Override
    public FeatureVector classify(Object arg) {
        Pair<Constituent, Constituent> argAndPred = ((Pair<String, Pair<Constituent, Constituent>>) arg).getSecond();
        return FeatureUtilities.getLBJFeatures(FeatureUtilities.conjoin(
            wordAndPos.getFeatures(argAndPred.getFirst()), wordAndPos.getFeatures(argAndPred.getSecond())));
    }
}
```

```
// LBJava code, to define a classifier, its input features and the expected output
import SrlWordAndPos;

discrete ArgumentIdentifier(Pair<String, Pair<Constituent, Constituent>> argAndPred) <-
learn IsArgument
  using SrlWordsAndPos
  from new SrlArgumentReader(Constants.corpusHome) 50 rounds
  with SparseNetworkLearner {
    SparseAveragedPerceptron.Parameters p = new SparseAveragedPerceptron.Parameters();
    p.learningRate = .1;
    baseLTU = new SparseAveragedPerceptron(p);
  }
```

Figure 9: LBJava code for training and evaluating an SRL argument identifier classifier. The `ArgParser` argument is a class that extends LBJava's `Parser` interface, and returns an argument example (label 'positive' or 'negative', argument constituent, and predicate constituent) for each call of its `next()` method.

Bies, A., Mott, J., and Warner, C. (2015). English news text treebank: Penn treebank revised.

Bird, S. (2006). Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 69–72. Association for Computational Linguistics.

Chan, Y. and Roth, D. (2011). Exploiting syntactico-semantic structures for relation extraction. In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, Portland, Oregon.

Chang, K.-W., Upadhyay, S., Chang, M.-W., Srikumar, V., and Roth, D. (2015). Illinoissl: A java library for structured prediction. *arXiv preprint arXiv:1509.07179*.

Chinchor, N. A. (1998). Overview of muc-7/met-2. Technical report, SCIENCE APPLICATIONS INTERNATIONAL CORP SAN DIEGO CA.

Clarke, J., Srikumar, V., Sammons, M., and Roth, D. (2012). An nlp curator (or: How i learned to stop worrying and love nlp pipelines). In *Proc. of the International Conference on Language Resources and Evaluation (LREC)*, 5.

Do, Q. and Roth, D. (2012). Exploiting the wikipedia structure in local and global classification of taxonomic relations. *Journal of Natural Language Engineering (JNLE)*, 18(2):235–262, 4.

Do, Q., Roth, D., Sammons, M., Tu, Y., and Vydiswaran, V. (2009). Robust, light-weight approaches to compute lexical similarity. *Computer Science Research and Technical Reports, University of Illinois*, page 94.

Doddington, G. R., Mitchell, A., Przybocki, M. A., Ramshaw, L. A., Strassel, S., and Weischedel, R. M. (2004). The automatic content extraction (ace) program-tasks, data, and evaluation. In *LREC*, volume 2, page 1.

Gabrilovich, E. and Markovitch, S. (2007). Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *IJcAI*, volume 7, pages 1606–1611.

Hovy, E., Marcus, M., Palmer, M., Ramshaw, L., and Weischedel, R. (2006). Ontonotes: the 90% solution. In *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers*, pages 57–60. Association for Computational Linguistics.

Ide, N., Fellbaum, C., Baker, C., and Passonneau, R. (2010). The manually annotated sub-corpus: A community resource for and by the people. In *Proceedings of the ACL 2010 conference short papers*, pages 68–73. Association for Computational Linguistics.

Loria, S., Keen, P., Honnibal, M., Yankovsky, R., Karesh, D., Dempsey, E., et al. (2014). Textblob: simplified text processing. *Secondary TextBlob: Simplified Text Processing*.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S., and McClosky, D. (2014). The stanford corenlp natural language processing toolkit. In *ACL (System Demonstrations)*, pages 55–60.

Meyers, A., Reeves, R., Macleod, C., Szekely, R., Zielinska, V., Young, B., and Grishman, R. (2004). The nombank project: An interim report. In *Proceedings of the Workshop Frontiers in Corpus Annotation at HLT-NAACL 2004*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Palmer, M., Gildea, D., and Kingsbury, P. (2005a). The proposition bank: An annotated corpus of semantic roles. *Computational linguistics*, 31(1):71–106.

Palmer, M., Gildea, D., and Kingsbury, P. (2005b). The Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106, March.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Pradhan, S., Moschitti, A., Xue, N., Uryupina, O., and Zhang, Y. (2012). Conll-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes. In

Joint Conference on EMNLP and CoNLL-Shared Task, pages 1–40. Association for Computational Linguistics.

Punyakanok, V. and Roth, D. (2000). Shallow parsing by inferencing with classifiers. In *Proc. of the Conference on Computational Natural Language Learning (CoNLL)*, pages 107–110.

Punyakanok, V., Roth, D., Yih, W., and Zimak, D. (2004). Semantic role labeling via integer linear programming inference. In *Proc. of the International Conference on Computational Linguistics (COLING)*, pages 1346–1352, Geneva, Switzerland, 8.

Punyakanok, V., Roth, D., and Yih, W. (2005). The necessity of syntactic parsing for semantic role labeling. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1117–1123.

Ratinov, L. and Roth, D. (2009). Design challenges and misconceptions in named entity recognition. In *Proc. of the Conference on Computational Natural Language Learning (CoNLL)*, 6.

Redman, T., Sammons, M., and Roth, D. (2016). Illinois named entity recognizer: Addendum to ratinov and roth '09 reporting improved results. Technical report.

Rizzolo, N. and Roth, D. (2010). Learning based java for rapid development of nlp systems. In *Proc. of the International Conference on Language Resources and Evaluation (LREC)*, Valletta, Malta, 5.

Rizzolo, N. and Roth, D. (2016). Integer linear programming for co-reference resolution. In Roland Stuckardt Massimo Poesio et al., editors, *Anaphora Resolution: Algorithms, Resources, and Applications*. Springer-Verlag.

Roth, D. and Zelenko, D. (1998). Part of speech tagging using a network of linear separators. In *Coling-Acl, The 17th International Conference on Computational Linguistics*, pages 1136–1142.

Samdani, R., Chang, K.-W., and Roth, D. (2014). A discriminative latent variable model for online clustering. In *Proc. of the International Conference on Machine Learning (ICML)*.

Sammons, M., Vydiswaran, V., and Roth, D. (2010). Ask not what textual entailment can do for you... In *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL)*, Uppsala, Sweden, 7. Association for Computational Linguistics.

Sammons, M., Christodoulopoulos, C., Kordjamshidi, P., Khashabi, D., Srikumar, V., Vijayakumar, P., Bokhari, M., Wu, X., and Roth, D. (2016). Edison: Feature extraction for nlp, simplified. In Nicoletta Calzolari (Conference Chair), et al., editors, *Proc. of the International Conference on Language Resources and Evaluation (LREC)*. European Language Resources Association (ELRA).

Sang, E. F. T. K. and Buchholz, S. (2000). Introduction to the conll-2000 shared task chunking. In *CoNLL/LLL*, pages 127–132. Association for Computational Linguistics.

Shetty, J. and Adibi, J. (2004). The enron email dataset database schema and brief statistical report. *Information sciences institute technical report, University of Southern California*, 4(1):120–128.

Srikumar, V. and Roth, D. (2013). Modeling semantic relations expressed by prepositions. 1:231–242.

T. Kim Sang, E. F. and De Meulder, F. (2003). Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 142–147. Association for Computational Linguistics.

Tsai, C.-T. and Roth, D. (2016). Illinois cross-lingual wikifier: Grounding entities in many languages to the english wikipedia. In *Proc. of the International Conference on Computational Linguistics (COLING) Demonstrations*, 12.

UzZaman, N., Llorens, H., Derczynski, L., Allen, J., Verhagen, M., and Pustejovsky, J. (2013). Semeval-2013 task 1: Tempeval-3: Evaluating time expressions, events, and temporal relations. In *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, volume 2, pages 1–9.

Walker, C., Strassel, S., Medero, J., and Maeda, K. (2006). Ace 2005 multilingual training corpus. *Linguistic Data Consortium, Philadelphia*, 57.

Weischedel, R., Palmer, M., Marcus, M., Hovy, E., Pradhan, S., Ramshaw, L., Xue, N., Taylor, A., Kaufman, J., Franchini, M., et al. (2013). Ontonotes release 5.0 ldc2013t19. *Linguistic Data Consortium, Philadelphia, PA*.

Wieting, J., Bansal, M., Gimpel, K., Livescu, K., and Roth, D. (2015). From paraphrase database to compositional paraphrase model and back. *arXiv preprint arXiv:1506.03487*.

Yin, W. and Schütze, H. (2014). An exploration of embeddings for generalized phrases. In *Proceedings of the ACL 2014 Student Research Workshop*, pages 41–47.

Zhao, R., Do, Q., and Roth, D. (2012). A robust shallow temporal reasoning system. In *Proc. of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 6.