EDISON: Feature Extraction for NLP

Mark Sammons[1], Christos Christodoulopoulos[1], Parisa Kordjamshidi[1], Daniel Khashabi[1], Vivek Srikumar[2], Paul Vijayakumar, Mazin Bokhari[1], Xinbo Wu[1], and Dan Roth[1]

[1]Cognitive Computation Group, University of Illinois [2]School of Computing, University of Utah

CCG Software

Cognitive Computation Group has numerous existing NLP applications implemented using a mixture of LBJava, Edison, and illinois-core-utilities (data structure library). As we work to build on these tools and use them in new, more complex NLP tasks we want to standardize them to improve ease-of-use and reliability in new environments. We also want to expose reusable elements within each package and share them across applications.
We decided to unify our NLP tool suite with a single API based on illinois-core-utilities, and expand Edison's feature extraction libraries with the goal of making it easier for other researchers to work with our software.

Illinois-core-utilities and Edison

illinois-core-utilities specifies a range of generic data structures and algorithms to support Natural Language Processing (NLP), plus utilities such as corpus readers for a number of NLP tasks. Edison specifies feature extraction code using these data structures.

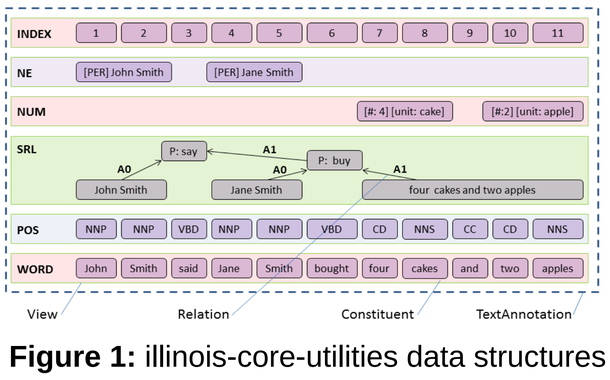


Figure 1: illinois-core-utilities data structures

https://github.com/IllinoisCogComp/illinois-cogcomp-nlp/tree/master/core-utilities

Edison Feature Extractor Search Interface

Edison users need to know what feature extractors are available, and what they do. Every feature extractor is named according to a standard to give some idea what Views they use and what features they generate. Each is documented with a clear description of its behavior, and each has a unit test that illustrates its use and specifies a representative output for that extractor.
The search interface allows users to search for NLP terms, keywords, and View names and retrieve matching Feature Extractors, displaying the unit test code for each selected extractor.

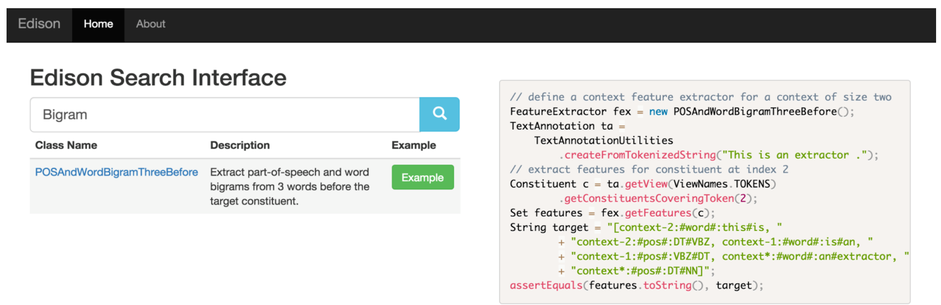


Figure 2: Screenshot of feature extractor search interface

http://bilbo.cs.illinois.edu:5900/

What is Feature Extraction?

Natural Language Processing systems such as Part of Speech taggers, Shallow Parsers, and Named Entity Recognizers are typically built around Machine Learning algorithms ("learners"). These algorithms build statistical models that take some representation of text as input, and predicts labels for elements of that text ("examples") as output. To work well, this representation must express useful abstractions over the text. The process of mapping from raw text to learner inputs is known as Feature Extraction.

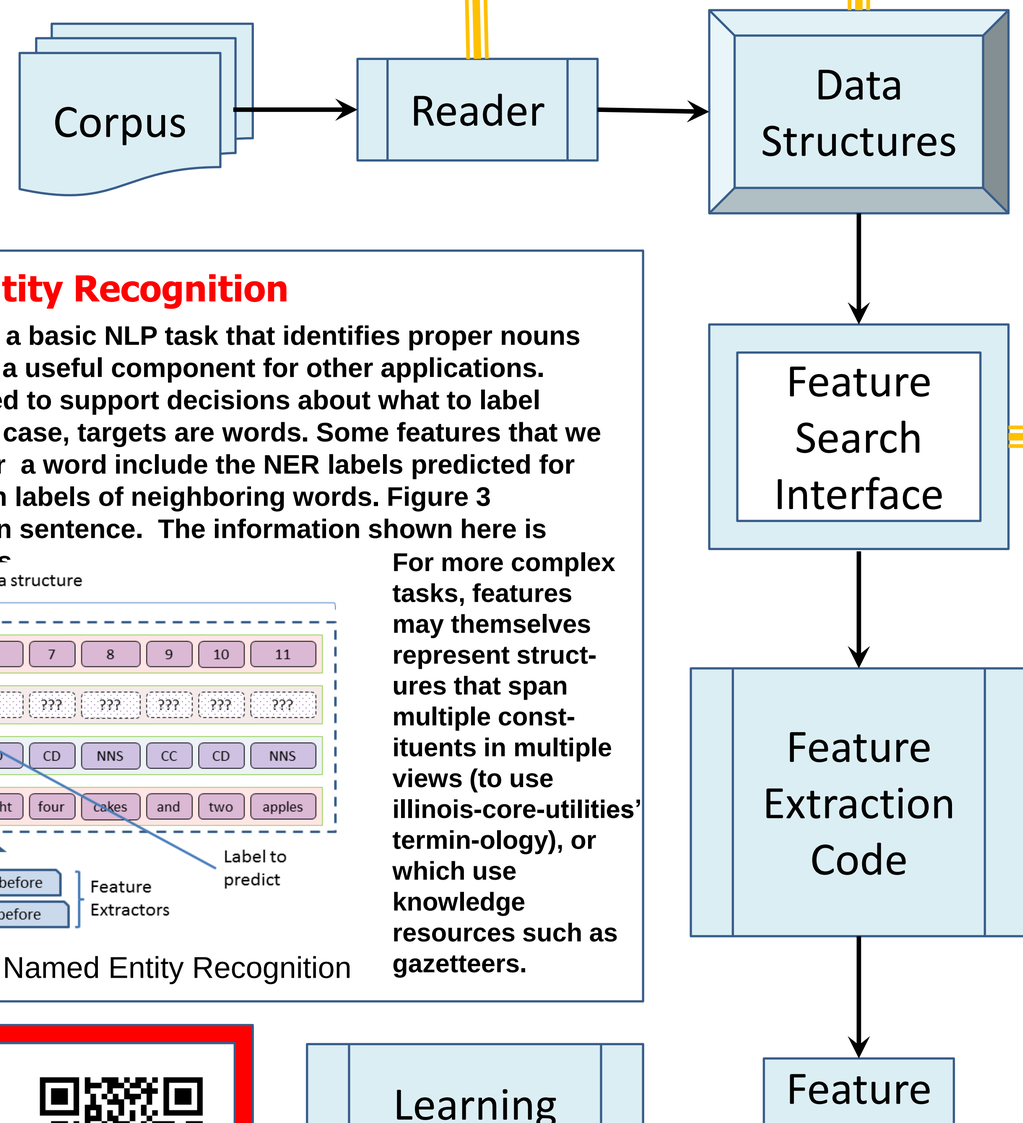
Feature Extraction Challenges

- Implementation is time consuming.
- Replicating other people's research is hard – their published descriptions of feature extractors may lack important details.
- There is much duplication of effort: many different versions of the same tools; and across different NLP applications, many similar/identical feature extractors.
- Even given existing feature extraction code, it may be hard to find the extractor you need.

Project Goals

- Speed up development by centralizing feature extraction and making it easy for developers to find existing implementations for feature extractors they need.
- Clarify individual project code and reduce maintenance overhead by using the same reference implementation of features where appropriate.
- Share reference implementation of feature extraction for specific applications/ publications to support duplication of results by other researchers.
- Initiate an open source project that can be improved and used by other researchers.

Using the Edison Feature Extraction Library



Example Task: Named Entity Recognition

Named Entity Recognition (NER) is a basic NLP task that identifies proper nouns and their types in English text. It is a useful component for other applications. Features represent information used to support decisions about what to label target elements of the text – in this case, targets are words. Some features that we can use to predict the NER label for a word include the NER labels predicted for previous words, and part-of-speech labels of neighboring words. Figure 3 illustrates these features for a given sentence. The information shown here is simple, as are the feature extractors.

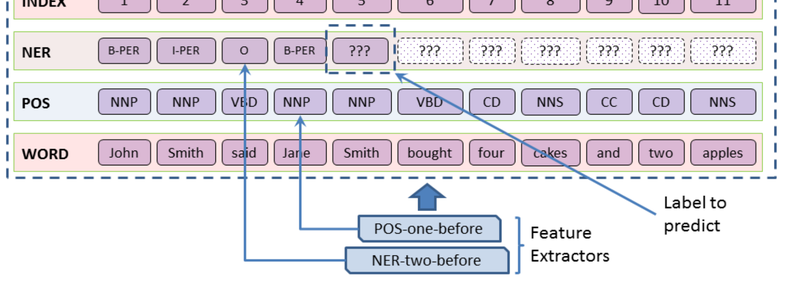


Figure 3: Feature extraction for Named Entity Recognition

For more complex tasks, features may themselves represent structures that span multiple constituents in multiple views (to use illinois-core-utilities' termin-ology), or which use knowledge resources such as gazetteers.

Find the source code at:

https://github.com/IllinoisCogComp/illinois-cogcomp-nlp/tree/master/edison

Use Feature Extractors Programmatically

LBJava and Saul

Edison's feature extractor classes can be directly incorporated in Java applications using the CCG data structures, and in programmatic learning environments.

LBJava is a self-contained extension to Java that supports machine learning. It provides a specification language that allows users to rapidly develop prototype machine learning algorithms.

Saul is a new Learning-Based Programming framework in Scala that supports rapid development of machine learning applications that use Structured Prediction methodologies. It generalizes the capabilities of LBJava.

LBJava: https://github.com/IllinoisCogComp/lbjava

Saul: https://github.com/IllinoisCogComp/saul

```
discrete NERLabel(Constituent word) <- { return word.getLabel(); }

discrete NERClassifier(Constituent word) <-
    learn NERLabel using Capitalization, WordContextBigrams,
         POSContextBigrams, ChunkContextBigrams
    with SparseNetworkLearner {
         SparseAveragedPerceptron.Parameters p =
            new SparseAveragedPerceptron.Parameters();
         p.learningRate = .1; p.thickness = 2;
         baseLTU = new SparseAveragedPerceptron(p);
}
```

Figure 4: Using Edison feature extractors in LBJava

```
object NERClassifier extends Learnable[Constituent](word) {
    def label = NERLabel
    override def feature = using(Capitalization, WordContextBigrams,
         POSContextBigrams, ChunkContextBigrams)
    override lazy val classifier = new SparseNetworkLBP
}
object NERDataModel extends DataModel {
    val word = node[Constituent]
    val NERLabel = property(word) { x: Constituent => x.getLabel }
    val surface = property(word) { x: Constituent => x.getSurfaceForm }
}
```

Figure 5: Using Edison feature extractors in Saul

Use Feature File Outputs

SVMLight format

Edison provides support for the use of popular machine learning packages such as Weka, Mallet, and SVMLight by providing classes to write out feature extractor outputs in the SVMLight data format. Edison generates a lexicon mapping feature types to integer values, and SVMLight input files that use these integer values.

```
#svmlight format
1 0:1 5:1 7:1 9:1 11:1 12:1 14:1 15:1 16:1 18:1 19:1
5 8:1 9:1 10:1 11:1 12:1 14:1 15:1 16:1 22:1 24:1
2 2:1 3:1 10:1 14:1 15:1 19:1 22:1 25:1 29:1 34:1
...

                #edison lexicon
                1    B-PER
                2    I-PER
                …
                12   NNP#&#John
                13   prefix-10:0101011100
```

Figure 3: Edison feature extractor output as SVMLight format and lexicon

This work is supported by the Army Research Laboratory (ARL) under agreement W911NF-09-2-0053, by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center contract number D11PC20155, and by DARPA under agreement number FA8750-13-2-0008.