# Increasing Maintainability of NLP Evaluation Modules Through Declarative Implementations

**Terry Heinze**
Research & Development Department
Thomson Corporation
Eagan, MN 55123
terry.heinze@thomson.com

**Marc Light**
Research & Development Department
Thomson Corporation
Eagan, MN 55123
marc.light@thomson.com

## Abstract

Computing *precision* and *recall* metrics for named entity tagging and resolution involves classifying text spans as true positives, false positives, or false negatives. There are many factors that make this classification complicated for real world systems. We describe an evaluation system that attempts to control this complexity through a set of rules and a forward chaining inference engine.

## 1 Introduction

Computing precision and recall metrics for named entity recognition systems involves classifying each text span that the system proposes as an entity and a subset of the text spans that the gold data specifies as an entity. These text spans must be classified as true positives, false positives, or false negatives.

In the simple case, it is easy to write a procedure to walk through the list of text spans from the system and check to see if a corresponding text span exists in the gold data with the same label, mark the text span as true positive or false positive accordingly, and delete the span from the gold data set. Then the procedure need only walk through the remaining gold data set and mark these spans as false negatives. The three predicates are the equality of the span's two offsets and the labels. This evaluation procedure is useful for any natural language processing task that involves finding and labeling text spans.

The question this poster addresses is how best to manage the complexity of the evaluation system that results from adding a number of additional requirements to the classification of text spans. The requirements may include fuzzy extent predicates, label hierarchies, confidence levels for gold data, and collapsing multiple mentions in a document to produce a single classification. In addition, named entity tasks often also involve resolving a mention of an entity to an entry in an authority file (i.e., record in a relational database). This extension also requires an interleaved evaluation where the error source is important.

We started with a standard procedural approach, encoding the logic in nested conditionals. When the nesting reached a depth of five (e.g., Figure 1), we decided to try another approach. We implemented the logic in a set of rules. More specifically, we used the Drools rules and forward chaining engine (http://labs.jboss.com/drools/) to classify text spans as true positives, false positives, and/or false negatives. The procedural code was 379 lines long. The declarative system consists of 25 rules with 150 lines of supporting code. We find the rules more modular and easier to modify and maintain. However, at this time, we have no experimental result to support this opinion.

## 2 Added Complexity of the Classification of Text Spans for Evaluation

**Matching extents and labels**: A system text span may overlap a gold data span but leave out, say, punctuation. This may be deemed correct but should be recorded as a fuzzy match. A match may also exist for span labels also since they may be organized hierarchically (e.g, cities and countries are kinds of locations). Thus, calling a city a location may be considered a partial match.

**Annotator Confidence**: We allowed our annotators to mark text span gold data with an attribute of "low confidence." We wanted to pass this information through to the classification of the spans so that they might be filtered out for final precision and recall if desired.

**Document level statistics**: Some named entity tagging tasks are only interested in document level tagging. In other words, the system need only decide if an entity is mentioned in a document: how many times it is mentioned is unimportant.

**Resolution**: Many of our named entity tagging tasks go a step further and also require linking each entity mention to a record in a database of entities. For error anal-



*Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, pages 1–2,
Columbus, Ohio, USA, June 2008. 

ysis, we wished to note if a false negative/positive with respect to resolution is caused by the upstream named entity tagger. Finally, our authority files often have many entries for the same entity and thus the gold data contains multiple correct ids.

```
for (annotations)
  if(extents & labels match)
    if(ids match => TP res)
      if(notresolved => TN res)
      else if(single id => TP res)
      else if(multiple ids => contitional TP res)
      else error
    else
      if(gold id exists)
        if(gold id uncertain => FP res low confidence)
        else => FP res
  else
    if(fuzzy extents & labels match)
      if(ids match)
        if(no gold id => TN res)
        else if(multiple ids => conditional TP res)
        else => fuzzy TP res
      else ...
```

Figure 1: Nested conditionals for instance classification

## 3 Using Rules to Implement the Logic of the Classification

The rules define the necessary conditions for membership in a class. These rules are evaluated by an inference engine, which forward chains through the rule set. In this manner, rules for fuzzy matches, for handling gold data confidence factors, and for adding exclusionary conditions could be added (or removed) from the rule set without modifying procedural code.

```
rule "truepositive" salience 100
        sa : SourceAnnotation( assigned == false )
        ta : TargetAnnotation( type == sa.type,
            beginOffset == sa.beginOffset, endOffset == sa.endOffset )
    then sa.setResult("TP");
rule "false positive" salience 90
        sa : SourceAnnotation( assigned == false )
        not TargetAnnotation( type == sa.type,
            beginOffset == sa.beginOffset, endOffset == sa.endOffset )
    then sa.setResult("FP");
rule "false negative" salience 80
        ta : TargetAnnotation( assigned == false )
        not SourceAnnotation( type == ta.type,
            beginOffset == ta.beginOffset, endOffset == ta.endOffset )
    then ta.setResult("FN");
```

Figure 2: Rules for instance classification

Three rules were needed to determine the basic collection level metrics. The results of these rules were then passed on to the next sets of rules for modification for conditional checks. We use agenda groups and rule salience to control the firing precedence within the rule sets. In Figure 2, we present an example of the sort of rules that are defined.

For example, the determination of true positives was made by firing the "true positive" rule whenever an annotation from the system matched an annotation from the gold data. This occurred if the entity type and offsets were equal. This rule was given higher salience than those for true negatives and false positives since it had the effect of removing the most candidate annotations from the working memory.

Note that because we use a Java implementation that adheres to JSR94, all of the rules apply their conditions to Java objects. The syntax for tautologies within the condition statements, refer to bean properties within the enclosing object.

In Figure 3, we show first, a modification to add a fuzzy metric rule that checks false negative annotations to see if they might be a fuzzy match. Second, we show a rule that removes false positives that are defined in a stop-word list.

```
rule"fuzzy check" agenda-group "FuzzyMatch"
    ta : TargetAnnotation( result == "FN" );
    sa : SourceAnnotation( type == ta.type, result == "FP",
      ta.beginOffset < endOffset, ta.endOffset > beginOffset );
    eval(ifFuzzyMatch(sa.getText(), ta.getText(), sa.getType()));
  then sa.setResult("FzTP");
rule "filter FP" salience 10 agenda-group "Filter"
    sa : SourceAnnotation( result == "FP" );
    eval(DexterMetrics.ifStopWord(sa.getText(), sa.getType()));
  then sa.setResult(sa.getResult() + "-ignored:stop word");
```

Figure 3: Rules for modified classification

## 4 Conclusion

We described some of the complexities that our evaluation module had to deal with and then introduce a rule-based approach to its implementation. We feel that this approach made our evaluation code easier to understand and modify. Based on this positive experience, we suggest that other groups try using rules in their evaluation modules.