RC24673 (W0810-101) October 16, 2008
Computer Science

IBM Research Report

CFE - A System for Testing, Evaluation and Machine Learning of UIMA Based Applications

Igor Sominsky, Anni Coden, Michael Tanenblatt
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

CFE – a system for testing, evaluation and machine learning of UIMA based applications

Igor Sominsky, Anni Coden, Michael Tanenblatt
IBM Watson Research Center
19 Skyline Dr., Hawthorne NY, 10532 USA
E-mail: sominsky@us.ibm.com, anni@us.ibm.com, mtan@us.ibm.com

Abstract

There is a vast quantity of information available in unstructured form, and the academic and scientific communities are increasingly looking into new techniques for extracting key elements - finding the structure in the unstructured. There are various ways to identify and extract this type of data; one leading system, which we will focus on, is the UIMA framework. Tasks that are often desirable to perform with such data after it has been identified are testing, correctness verification (evaluation) and model building for machine learning systems. In this paper, we describe a new Open Source tool, CFE, which has been designed to assist in both model building and evaluation projects. In our environment, we used CFE extensively for both building intricate machine learning models, running parameter-tuning experiments on UIMA components, and for evaluating a hand-annotated "gold standard" corpus against annotations automatically generated by a complex UIMA-based system. CFE provides a flexible, yet powerful language for working with the UIMA CAS - the results of UIMA processing - to enable the collection and classification of resultant data. We describe the syntax and semantics of the language, as well as some prototypical, real-world use cases for CFE.

1. Introduction

A wealth of information is captured in unstructured sources, ranging from text to streaming video. Analysis of these sources and extraction of knowledge from them is the goal of several frameworks currently in use within the research community. Two open source frameworks, the Gate system (http://www.gate.ac.uk) and the UIMA framework (http://incubator.apache.org/uima) have gained popularity. Although different in several aspects, both systems are modular, providing a mechanism for creating and executing a pipeline of components, known as "annotators". These annotators implement various algorithms, each of which performs a specific analysis task. In this paper, we will focus on textual unstructured data sources. Hence, examples of annotators are natural language processing (NLP) components, such as part-of-speech taggers and parsers, rule based annotators or named entity annotators based on a variety of machine-learning algorithms.

One of the challenges faced by all application developers is the testing and evaluation methodology. At a high level, the issues typically are regression testing and computation of accuracy metrics (e.g. precision/recall) against a "gold standard". There are many tools available (e.g., Knowtator (http://knowtator.sourceforge.net) and Callisto (http://callisto.mitre.org)) for manually annotating documents, both for building machine learning training data and for creating "gold standard" corpora to be used as a reference set in testing. Evaluation and testing involves comparing annotations from different executions. Within the UIMA framework, this can be accomplished by extracting and comparing values of properties of UIMA annotations. These annotations can be arbitrarily complex. Extraction of these properties, called *features*, is also one of critical sub-tasks in creating machine learning models, as the feature vectors for building the models can be generated from features values of UIMA annotation.

It should be noted that the term *features*, which is frequently used throughout this paper, is often used in different contexts. This term may refer to properties of UIMA annotation types or features that are used to build/evaluate models for machine learning algorithms. In this paper we will use the term *features* in relation to properties of UIMA annotations, while values of models for machine learning will be referred as *ML features*.

What we needed, but were not able to discover, was a tool that could be configured to extract specific portions of a UIMA CAS (Common Analysis Structure: the object-based data structure in which UIMA represents objects, properties and values), specifically a set of features from some set of annotations based on user specified conditions. Traditionally, application-specific "CAS Consumers" have been written to satisfy this requirement. While this approach is reasonable for a fixed (or nearly fixed) set of output requirements, it can be unwieldy when experimenting with different sets of features to be extracted, an underlying annotation model is in flux, or if two or more differing (yet equivalent) models need to be extracted and aligned. For these reasons, we created a system to perform these kinds of extraction tasks, and which provides a powerful declarative extraction specification language. The same functionality is also needed to generate *ML features* to build models that underlie machine learning algorithms. To accomplish the final steps of evaluation tasks, we combine the generalized feature extractor with a system within which accuracy metrics can be computed.

This paper is organized as follows. In section 2, we will describe the challenges of testing and evaluating UIMA pipelines in detail and discuss why other testing and evaluation environments proved to be inadequate. The feature extraction specification language (FESL) – is introduced in section 3. Section 4 will describe a real-word use case of FESL performance evaluation of an NLP system and section 5 will demonstrate how FESL can be used for machine learning related processing. We conclude in section 6 with proposing some potential extensions.

2. Problem statement

Evaluation of an information extraction system consists of several steps: defining a baseline against which to compare, defining the comparison criteria, extracting relevant information from sources (e.g., the baseline and the system to be evaluated) and subsequent comparative analysis.

At a very general level, for a given textual document, a UIMA pipeline executes as shown in Figure 1.

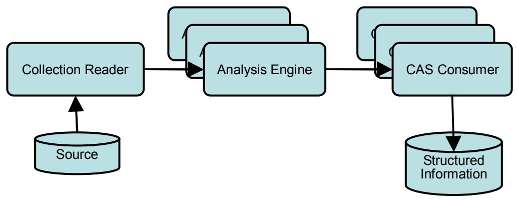
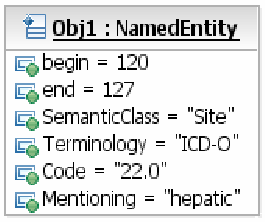


Figure 1: UIMA pipeline

First, the document is read into a Common Analysis System (CAS) structure. Next, a set of analysis engines (AEs) mark up this piece of text, producing *annotation* objects, each of which is usually associated with a span of text in that document. Finally, one or more CAS Consumers read these annotations, perform any necessary processing, and then output results.



Obj1 : NamedEntity
begin = 120
end = 127
SemanticClass = "Site"
Terminology = "ICD-O"
Code = "22.0"
Mentioning = "hepatic"

Figure 2: An abstract view of typical UIMA annotation

Each annotation (as shown for example in Figure 2) has properties associated with it. These properties contain specific information about the annotation, and as described in the introduction, are called *features*. Although the actual implementation of UIMA annotation objects is much more complex, this abstract view reflects information stored in these objects. The values of features are set by AEs and could either be modified or used without modification by subsequent annotation engines. In the example in Figure 2, the annotations are created with a dictionary lookup mechanism against a medical terminology, the attributes being the begin and end offsets of the relevant piece of text in the document that this annotation object is associated with, the semantic class of the named entity that is described by the annotation, the terminology name and code associated with it from that dictionary, and the actual text fragment.

The first step in the process of evaluation is the definition of equality between two types to be compared. This necessitates a specification of a set of features from both the test and reference sets that should be compared, and the criteria for the comparison. In the next step, the annotations of those types and their significant properties are extracted. We developed the language FESL to specify the details of this extraction. FESL contains sufficient semantics for expressing rules for generation of parameters for building machine learning models. The extraction can be implemented as part of a standard UIMA component (AE or CAS consumer) depending on particular application requirements. For the evaluation environment, we developed a tool that extracts required feature values using a CAS consumer. It performs the extraction from two CAS structures that are to be compared and loads the extracted information into a Microsoft Excel spreadsheet, where the final stages of the evaluations are executed, as described in section 4.

3. The Feature Extraction Specification Language (FESL)

To enable a high degree of flexibility and extensive functionality, we defined an XML-based specification language that expresses semantic rules for feature extraction. One of the key concerns in defining the language was to avoid any dependency upon any particular application of the extraction process. This allows reusing the same extraction semantics for different purposes, whether for comparative analysis, subsequent algorithm execution or machine learning related processing. The feature extraction process is independent of the representation of the feature in the final output. This enables different output formats for different use cases, such as machine learning or testing. As a simple example, extracted values for comparison could contain spaces in their representation, while the same values extracted for machine learning could replace spaces with underscore characters. The component also defines a destination for output. For instance, the analysis engine (AE) could store the extracted features values within a CAS structure and/or a subsequent CAS Consumer might output them to an external source such as a disk file or database.

The semantics of the specification language allow the definition of complex multi-parameter criteria that could identify a particular concept of interest. Such criteria allow locating the information expressed by any particular UIMA annotation and/or its features in a CAS structure, evaluating its value against one or more conditions and recording the results in an internal depository for post processing. The criteria for such search can be specified by a combination of the following conditional expressions, written with FESL:

a. type of an annotation object that contains the feature (in the general case, the feature does not have to be a property of the object, but should be accessible (i.e *on the path*) from its properties, as will be shown further down in this section)
b. surrounding (enclosing) annotation type and relative location of the object within the enclosure, as indicated by the *enclosingAnnotation* attribute of the *targetAnnotations* XML tag, shown in Figure 3 (the significance of the enclosing annotation is explained below)
c. path to the feature from the annotation object, as indicated by the *featurePath* attribute of the *featureMatchers* XML tags, as shown in Figure 3
d. type and value of the feature itself; the feature value can be evaluated against different constraints expressed with FESL, as explained further down in this section
e. values of any public Java *get*-style methods (methods that accept no parameters and return a value) implemented by the underlying class of the feature
f. location of the object or the feature on a specific path (in cases when it is required to select/bypass annotations if they are features of certain annotation types)

One of the key capabilities of FESL mentioned in items (a), (c) and (f) is an ability to specify a "path" to a feature from an annotation object. This path is a sequence of feature/method names, separated by the colon character, that mimics the sequence of Java method calls required, starting at the annotation object, in order to extract the feature value. It should be noted that, as UIMA annotations support arrays as feature types, FESL also provides the ability to extract values of features that are arrays or properties of annotations that are contained in arrays. Figure 5 contains a sample of how arrays are specified in FESL. In addition, special array semantics allow accessing elements of arrays by index and sorting them by offset before extraction.

Some applications require performing an extraction of information relevant to a certain concept within sentence boundaries; other may extend the scope of the extraction to a paragraph. As mentioned in item (b) FESL has the ability to define such a scope by specifying an enclosing annotation as illustrated in Figure 3.

Typically, values of UIMA annotation features are required to be extracted, but FESL also enables an extraction of non-UIMA properties of an object by using Java reflection mechanism. As specified by item (e), a value returned by any public method that has no arguments can be extracted and treated in the same way UIMA features are processed. As shown in Figure 3, *getCoveredText* is not a property of a UIMA Annotation type, but rather a method that this type defines.

As previously mentioned in item (d) the feature values can be evaluated by conditional expressions stated in FESL. Particularly, the feature values can be evaluated whether they:

i. are of a certain type
ii. belong to a specific set of values (vocabulary), where the set of values, as shown on Figure 3, is defined by the *enumFeatureValues* XML tag
iii. belong to a range of numeric values (inclusively or non-inclusively) as defined by the *rangeFeatureValues* XML tag
iv. match certain bits of a bit mask (integer values only); the *bitmaskFeatureValues* XML tag will contain an integer bitmask along with a flag indicating whether the bitmask should exactly match to a feature value
v. match a Java regular expression pattern, where the *patternFeatureValues* XML tag will contain a regular expression against which a feature value will be evaluated

The evaluation of the search criteria can be specified in disjunctive normal form. Conjunctions are bounded by FESL *groupFeatureMatcher* XML tags and are referred to as groups. Disjunction is implicit between multiple groups. This gives a powerful and flexible way of defining fairly complex criteria for a search of a required annotation and/or its value.

It should be noted that the semantics of FESL, as shown in Figure 3, separate the concept and specification of target annotations (*TA*) from feature annotations (*FA*). Although they use identical semantic rules for specifying the search criteria, the ways the results of the search are processed are different. In particular, *TAs* are used to locate a concept, while *FAs* are the annotations upon which the extraction of features is performed. Target annotations are specified by the *targetAnnotationMatcher* XML tag, and feature annotations by the *featureAnnotationMatcher* XML tag. During the extraction process, a *TA* is located according to its search criteria. Once the TA is found, *FAs* that correspond to the *TA,* and match to their own search criteria, are located and feature values are extracted from them. Additionally, the semantics allow the extraction of features from multiple *FA*s, where each *FA* is located by its specific context relative to the *TA*. This is particularly useful in machine learning related processing where it is often required to select features from annotations that are located "near" another annotation with certain properties.

Let us consider a quite common example taken from the machine learning domain: extracting "a bag of words within a window of size 5 centered around the word 'tumor', excluding prepositions, conjunctions, articles and punctuation". This could be understood as: search for token-based annotations that corresponds to the word "tumor" (*TA*), and on every match consider the 5 nearest token-based annotations (*FA*s) on both sides, and excluding tokens that have associated part-of-speech tags indicating they are of one of the following categories: preposition, conjunction, article or punctuation, then extract the token that corresponds to that *FA*. The FESL semantics allow the unambiguous specification of criteria for such a search that is shown in Figure 3.

```
<targetAnnotations className="BOW5Tumor"
    enclosingAnnotation="SentenceAnnotation">
  <targetAnnotationMatcher annotationTypeName="TokenAnnotation">
    <groupFeatureMatchers>
      <featureMatchers featurePath="getCoveredText" featureTypeName="String">
        <enumFeatureValues>
          <values>tumor</values>
        </enumFeatureValues>
      </featureMatchers>
    </groupFeatureMatchers>
  </targetAnnotationMatcher>
  <featureAnnotationMatchers annotationTypeName="TokenAnnotation"
      windowsizeLeft="5" windowsizeRight="5">
    <groupFeatureMatchers>
      <featureMatchers featurePath="getCoveredText" featureTypeName="String"/>
      <featureMatchers featurePath="pennTag" featureTypeName="String"
          exclude="true">
        <enumFeatureValues caseSensitive="true">
          <values>IN</values>
          <values>CC</values>
          <values>DT</values>
          <values>null</values>
        </enumFeatureValues>
      </featureMatchers>
    </groupFeatureMatchers>
  </featureAnnotationMatchers>
</targetAnnotations>
```

Figure 3: Bag of words extraction sample

In this figure, short versions of UIMA annotation type names are shown for better readability. In the example, all extracted feature values are assigned a label "BOW5Tumor" (the value of the *targetAnnotation*'s "className" attribute). The label could be used in subsequent processing for the grouping of related results of extraction. The search is limited to token annotations (*TokenAnnotation*) within the same sentence (*SentenceAnnotation*), as specified by *enclosingAnnotation* attribute. Also, annotations of type *TokenAnnotation* have a property called *pennTag* that contains their part-of-speech tags. As illustrated in this example, the *TokenAnnotation*'s *getCoveredText* attribute is evaluated if, and only if, that same *TokenAnnotation*'s *pennTag* contains a value in the set specified under *enumFeatureValues* XML tag.

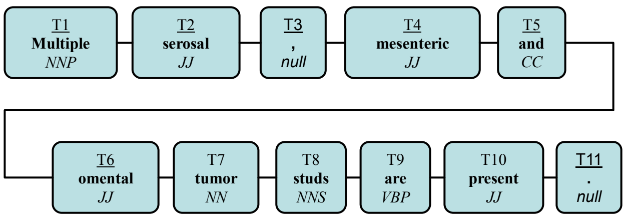


Figure 4: Tokenized sentence

To demonstrate how this FESL specification is applied consider the sentence from Figure 4. Each box on this figure corresponds to a single *TokenAnnotation*. These *TokenAnnotations* are all enclosed within a single *SentenceAnnotation*. Each *TokenAnnotation* contains a unique label, a text string covered by this annotation and a POS tag. According to the FESL specification in Figure 3, first a *TA* of a *TokenAnnotation* type with covered text *tumor* is searched for. Once it is found (T7), a search is performed for 5 *FAs* of a type *TokenAnnotation* to the left from T7. Only annotations whose POS tag is not *IN, CC, DT or null* are selected during the search. Thus the selected *FAs* will be T6, T4, T2 and T1. The same algorithm applied on the right context of T7 will produce a selection of *FAs* labeled T8, T9 and T10. As has been mentioned earlier, the search for *FAs* is limited by sentence boundaries. For this reason, even though a *windowSizeRight* and *windowSizeLeft* are specified with the value 5, fewer than five *TokenAnnotations* are actually selected.

As opposed to this previous example for machine learning, in the case of feature extraction for a comparative analysis (e.g. evaluation), the *TA* and *FA* usually are the same.

To demonstrate another set of capabilities of FESL, consider a case where it is necessary to process annotations that implement hierarchical models, (i.e., annotations containing other annotations, which may themselves contain annotations, etc.), with multiple levels of containment. The set of particular features that are required for extraction depend on where in the hierarchy the annotation is located and how it is related to the higher level annotation. As an example, we consider a case where it is required to distinguish between a dimension of a surgical margin and dimensions of a tumor. Figure 5 illustrates these capabilities, where the requirement is to extract values of features of *Dimension* annotations that are constituents of *Size* annotations which in turn are properties of two different containing UIMA annotations: *PrimaryTumor* annotations and *MetastaticTumor* annotations. An additional requirement is to extract feature values of all other *Dimension* annotations under a separate label. Figure 5 illustrates how these complicated requirements can be specified with FESL:

```
<targetAnnotations className="PrimaryTumorDimension"
    enclosingAnnotation="SentenceAnnotation">
  <targetAnnotationMatcher annotationTypeName="Size" fullPath="
      PrimaryTumor:Size"/>
  <featureAnnotationMatchers annotationTypeName="Size"
      windowsizeInside="1">
    <groupFeatureMatchers>
      <featureMatchers featurePath="Dimensions:toArray:Unit"
          featureTypeName="String"/>
      <featureMatchers featurePath="Dimensions:toArray:Extent"
          featureTypeName="String"/>
    </groupFeatureMatchers>
  </featureAnnotationMatchers>
</targetAnnotations>
<targetAnnotations className="MetastaticTumorDimension"
    enclosingAnnotation="SentenceAnnotation">
  <targetAnnotationMatcher annotationTypeName="Size"
      fullPath="PrimaryTumor:Size"/>
  <featureAnnotationMatchers annotationTypeName="Dimension"
      windowsizeInside="3">
    <groupFeatureMatchers>
      <featureMatchers featurePath="Unit" featureTypeName="String"/>
      <featureMatchers featurePath="Extent" featureTypeName="String"/>
    </groupFeatureMatchers>
  </featureAnnotationMatchers>
</targetAnnotations>
<targetAnnotations className="Processed"
    enclosingAnnotation="SentenceAnnotation">
  <targetAnnotationMatcher annotationTypeName="Dimension"
      fullPath="PrimaryTumor:Size:Dimensions:toArray"/>
</targetAnnotations>
<targetAnnotations className="Processed"
      enclosingAnnotation="SentenceAnnotation">
  <targetAnnotationMatcher annotationTypeName="Dimension"
      fullPath="MetastaticTumor:Size:Dimensions:toArray"/>
</targetAnnotations>
<targetAnnotations className="OtherDimension"
    enclosingAnnotation="SentenceAnnotation">
  <targetAnnotationMatcher annotationTypeName="Dimension"/>
  <featureAnnotationMatchers annotationTypeName="Dimension"
      windowsizeInside="1">
    <groupFeatureMatchers>
      <featureMatchers featurePath="Unit" featureTypeName="String"/>
      <featureMatchers featurePath="Extent" featureTypeName="String"/>
    </groupFeatureMatchers>
  </featureAnnotationMatchers>
</targetAnnotations>
```

Figure 5: Dimension extraction sample

In the example above, a path to features of interest that are properties of feature annotations (*FA*) is specified by a sequence of properties/methods that are required in order to locate the final feature. For example:

```
fullPath="PrimaryTumor:Size:Dimensions:toArray"
```

specifies that *PrimaryTumor* contains a property called "*Size*" of a type that has an array of dimensions, and elements of that array should be of type *Dimension* as enforced by the *annotationTypeName* attribute. The first target annotation (TA) with a class label *PrimaryTumorDimension* is specified to be of a type *Size* and located on a path *PrimaryTumor:Size.* This specification ensures that only Size annotations that are constituents of *PrimaryTumor* annotations are matched. Once the *TA* is located, a feature annotation (*FA*) of the same type *Size* is searched for within the offset boundaries of the *TA,* which is enforced by *windowsizeInside* attribute. In this example, the value of *windowsizeInside* attribute is set to 1, guaranteeing that the same *Size* annotation that was previously selected as the *TA* will also be selected as the *FA*. The same rules apply to the processing of target annotations referenced by the *MetastaticTumorDimension* class label. Also in this example, a specification of an arbitrary label "Processed" with no *FA* specification should be noted. This illustrates the functional feature of FESL of excluding annotations (*TAs*) that have been matched during the previous search from further processing. Thus, dimensions matched for tumor sizes will not by considered during the search specified by criteria with label *OtherDimension*.

4. Automated performance metrics evaluation

Comparison of results produced by a pipeline of UIMA annotators to a "gold standard" or results of two different NLP systems is a frequent task, and should be automated. Creating a uniform methodology that would not just simplify the comparison, but would also facilitate the identification of common sources of errors and measure performance improvements gained by correcting these errors, is crucial in the NLP research and development process.

Using FESL as an information extraction mechanism, we developed such a methodology that includes several steps:

- defining the comparison criteria
- extracting the relevant features
- extracting relevant information from two sources to be compared into a spreadsheet-compatible format
- comparative analysis of extracted information

Only the first step has to be done manually; all others can be completely automated.

4.1 Defining the comparison criteria

The definition of the comparison criteria between two CAS structures is a critical step in the evaluation process. Each CAS structure can have its own type system, and the information represented by an individual type from one type's system does not necessarily mirror the information stored in the corresponding type of a different type system. In fact, its constituent parts could be spread across multiple types. For complex types (types that include other types and are also a part of the comparison process), the relevant constituents to be used in the definition of equality must be defined. The result of this step is a set of FESL configuration files and set of custom comparison Excel spreadsheet templates (*CST*). The FESL configuration files specify the feature extraction, whereas the templates implement the comparison criteria.

It is within the CST's that the comparison between two CAS structures is executed. Information from both structures is loaded into a CST, and then the comparison is implemented with a set of macros that perform the following:

- compare two corpora based on the user defined equality criteria
- calculate performance metrics such as precision, recall and F-score.

In addition they could include macros to take into account errors in the "gold standard" or estimate the performance gain by fixing a specific algorithm or implementation errors in automated annotators.

4.2 Feature extraction

Feature extraction is performed using a custom UIMA CAS consumer that uses a FESL configuration file and a CAS structure as its input and outputs delimited files with feature values. This CAS consumer contains code which interprets and executes the FESL configuration. A sample of such an implementation will be released into Open Source as part of the Apache UIMA incubator project. The fundamental semantic rules implemented by FESL were covered in section 3. Features are extracted from both sources that are being compared, resulting in two delimited files that are merged into a single file. This process uses the offsets of annotations within the document to guide the merger. The merged file can be easily imported into a custom Excel spreadsheet for further analysis, as discussed in section 4.1. In our environment, the creation of a spreadsheet from two delimited files is completely automated. Figure 6 shows typical content of a merged file with feature values extracted from two sources. We used a vertical bar ("|") character as the value separator, since our data can never contain one—for use with other data sets, this can be customized accordingly:

```
$ head  set1-SizeDim-report.txt
18|24|4.0|cm|18|24|4.0|cm|gold/doc0.fve|medtas/doc0.fve
40|47|12.0|cm|40|47|12.0|cm|gold/doc0.fve|medtas/doc0.fve
106|118|6.5|cm|106|118|6.5|cm|gold/doc0.fve|medtas/doc0.fve
112|118|2.0|cm|112|118|2.0|cm|gold/doc0.fve|medtas/doc0.fve
249|261|3.8|cm|249|261|3.8|cm|gold/doc0.fve|medtas/doc0.fve
255|261|2.5|cm|255|261|2.5|cm|gold/doc0.fve|medtas/doc0.fve
275|281|5.0|cm|275|281|5.0|cm|gold/doc0.fve|medtas/doc0.fve
182|187|30|cm|182|187|30|cm|gold/doc10.fve|medtas/doc10.fve
211|216|20|cm|211|216|20|cm|gold/doc10.fve|medtas/doc10.fve
72|85|0.05|cm|72|85|0.05|cm|gold/doc100.fve|medtas/doc100.fv
```

Figure 6: Merged results of feature extraction

4.3 Comparative Analysis

Comparative analysis usually includes several steps – calculation of performance metrics, error analysis, and evaluation of the most effective ways of improving accuracy (e.g. identification of types of errors and corrections that would maximize accuracy). As was mentioned earlier, the calculations are done automatically by macros, while error analysis and evaluation, for the most part, must be done manually. One way that the evaluation can be partially automated is that one of the implemented macros allows errors to be classified according to a code (e.g., errors in the gold standard vs. errors in the automatic annotations) and performance metrics recalculated based on these error codes.

5. Using feature extraction for machine learning

Machine learning algorithms build and apply models to extract pertinent information from sources such as a text documents or images (Mitchell, 1997). In addition to the machine learning algorithm, the process of defining of ML feature set itself is a critical factor in building accurate models of the information to be identified. In general, extensive experimentation with a variety of parameters is done to create models which perform with the desired accuracy for a particular task.

The complexity of feature extraction varies, but it is desirable to have a comprehensive mechanism to rapidly extract them. CFE is such a mechanism for textual data sources. In section 3 we described FESL and its semantics, which can be used to specify which features should be extracted. In this section we will describe some details as they pertain to feature extraction within the machine learning domain.

In particular, information from a surrounding context of a specific term has to be taken into account, and additionally, that context can be constrained by multiple conditions specific to the task. Design of FESL takes such considerations into account by allowing specification of fairly complex and precise criteria for locating and extracting particular pieces of information. For Word Sense Disambiguation (WSD), in addition to the FESL configuration, we developed a CAS consumer that generates machine learning models and AEs that evaluate the models within a classification task.

One of the steps in building models for machine learning for textual data is generation of parameter sets from a text corpus. The syntax and semantic of FESL, as previously described, is sufficient for this task. The generated parameter set contain individual machine learning features (*MLFs* - not to be confused with UIMA features) whose symbolic names are constructed from values extracted according to FESL specification. In cases where more then one UIMA feature value is extracted for a particular *MLF,* the extracted values are concatenated to produce a MLF symbolic name. For instance when extracting size information from a context of a term to be disambiguated we could produce an MLF that is presented as "L1_Size_53_58_25_cm" which is a combination of an annotation type that the information was extracted from (Size), numeric extents for three dimensions (53, 58, 25) and a measurement unit (cm). It should be noted that prefix "L1" indicates that FESL configuration specified to include a position of a MLF relative to the term into the MLF name. A position is characterized by direction and distance, thus "L1" should be read as "first size annotation to the left from the term to be disambiguated". In cases where neither the distance nor the direction is required to be a part of an MLF name it will be prefixed with "X0". Figure 7 shows a typical content of an MLF file for WSD:

```
$  cat ml_feature.txt
X0_Dimension_12_cm
X0_Size_45_50_12_cm
X0_Dimension_45_cm
X0_Size_4_6_5_cm
X0_Dimension_10_cm
X0_Size_20_35_10_cm
```

Figure 7: Sample of MLF file for WSD

6. Conclusion

In this paper, we proposed CFE (Common Feature Extraction), a methodology and system for testing and evaluating complex NLP applications executed within the UIMA framework. The core of the system is a declarative language FESL, and a UIMA component that processes FESL specifications, using them to guide extraction of data from a UIMA CAS in a completely generalized way, and providing a method for subsequent processing to format the output as needed for any downstream use. In addition, CFE can be used to rapidly specify and extract features to build models for machine learning algorithms. The flexibility and ease-of-use of the system enables easy experimentation with different models in the machine learning space. CFE was used in quite different tasks: experimenting with large numbers of feature sets to build models for word sense disambiguation, evaluating a sizable set of parameters for dictionary lookup and evaluating the automatic filling of hierarchical knowledge models. The comparison spreadsheets proved to be invaluable in determining which algorithmic improvements would result in the most substantial improvements in precision and recall.

For a next step, a GUI for generating FESL configuration files is planned. Other possible extensions are automating the process of building refined models and automatically evaluating them. The CFE system, the FESL declarative language specification and the UIMA component to interpret it will be released into Open Source as part of the Apache UIMA incubator project.

7. Acknowledgements

We thank Rie K. Johnson for her support and Wei Guan for experimenting with CFE.

8. References

Coden A.R., Savova G.K., Buntrock J. D., Sominsky I.L., Ogren P.V. , Chute C.G., de Groen P.C. (2007) *Text* Analysis Integration into a Medical Information Retrieval System: Challenges Related to Word Sense Disambiguation: *Medinfo 2007*

Mitchell Tom. (1997). *Machine Learning*, McGraw Hill

Ogren, P.V. (2006). Knowtator: A Protégé plug-in for annotated corpus construction. Rochester, MN. *Abstract for HLT-NAACL 2006.*

Savova G.K., Coden A.R, Sominsky I.L., Johnson R.K., Ogren P.K., de Groen P.C. and Chute C.G. (2008). Word Sense Disambiguation across Two Domains: Biomedical Literature and Clinical Notes. To appear in *Journal of Biomedical Informatics*