UML Design and Named Entity Recognition Implementation with UIMA SDK

11-791 Homework 1
Sukhada Palkar (spalkar)

Overview

A named entity recognition system is implemented in this task using the UIMA SDK. The named entities to be detected are gene names. A single file with sentence identifiers and sentences is read in as the input to the system. The output of the system is a file containing each detected gene name along with the sentence identifier and the span of the name in the sentence represented as integer offsets. Two algorithms are used simultaneously to detect gene names from the sentences. The details of the algorithms are described below. The result gene name set is the union of the results of the two algorithms.

System Architecture

The UIMA workflow consists of the Collection Processing Engine which requires a collection reader, a set of analysis engines and a consumer. Each of these components is described in detail below. In addition, a custom type system was defined for capturing the particular kind of text annotations generated and consumed by different parts of this system. The type system is defined in detail too. Lastly, a workflow diagram of all the components with their particular class names is added as a summary.

The components were designed keeping in mind a large sphere of requirements for an extensible and flexible gene name recognizer. Even though it was not possible to implement a lot of the envisioned features, the descriptions below include ways of extending and tuning the current system for a better performing and gene recognizer.

Collection Reader

Implementation: GeneCollectionReader.java

This is a basic collection reader that reads in the text file from configuration and processes it for downstream analysis. The file is split into multiple ‘documents’ where each document is a line in the file. The sentence identifier (as described in the overview section) is retained in the document for easy access by annotators.

Custom Type System

ner.annot.gene
start end sofa source docID

The type system for gene name recognition was very simple with a single type ‘gene’. Gene is an extension of the UIMA Annotation type. The base Annotation type consists of 3 items: a start offset, end offset and the text that is the subject of analysis. In addition to these three features, the gene annotation required the sentence identifier from which the annotation was derived. Hence, an identifier feature of type String was added to the Annotation type. Multiple annotators could be used for gene annotation (2 were used in this system). There are result aggregation algorithms that will benefit from identifying the different sources of an annotation. For this purpose, another ‘source’ feature of type String was added to the Annotation type.

Simple Gene Annotator

Implementation: SimpleGeneAnnotate.java

This annotator works with simple rules or heuristics to identify text spans that can be gene names. In the existing implementation, the annotator relies on matching a set of regular expressions to individual words in the input text to identify gene names. These regular expressions leverage the fact that gene names have certain patterns like numbers, acronyms with all uppercase letters etc. for a simplistic detection algorithm. The set of rules to check is easily extensible. Also, not every rule has to be applied to every word. For example, the rule that checks for first-letter capitalization is not applied to the first word in the input text.

POS based NER

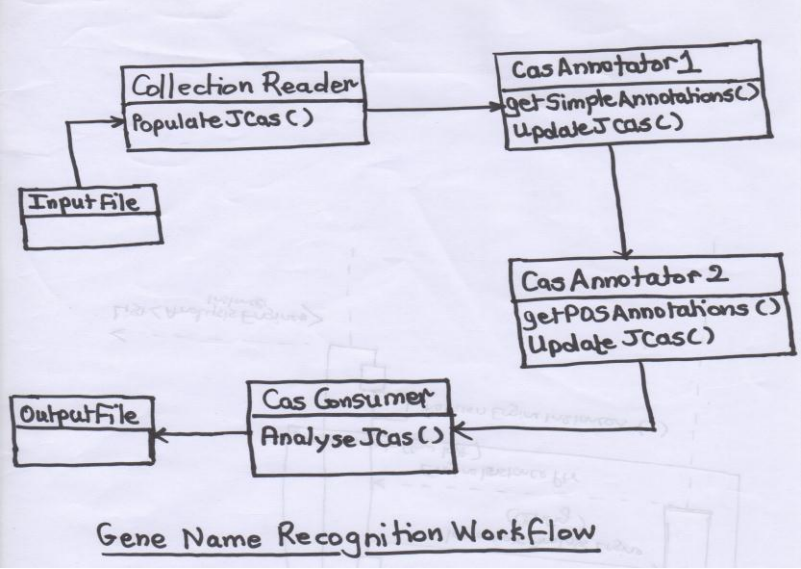
Implementation: POSGeneAnnotate.java

This annotator uses the code provided to mark gene names based on part of speech tags. The code provided uses the Stanford CoreNLP pipeline to tag the input with parts of speech. The word spans that are tagged as noun types are then returned. The annotator converts the returned list of word spans to gene annotations and sends them through the pipeline.

Cas Consumer

Implementation: GeneCasConsumer.java

The Cas consumer processes the annotations generated by the entire system into the required output file format. It simply reads the gene annotation types from the Cas and recalculates the annotation character offsets to suit the output requirements. It then writes the output to a file specified as a configuration parameter.



Technical Aspects of Gene Name Recognition

The focus of this project was on design of a system using the UIMA SDK. This system implements some basic name recognition techniques. Below is a discussion of these techniques and what could be done to improve on them.

Simple Rule Based Annotation

In this implementation we check each word in the input for 4 simple rules:

i. starts with an uppercase letter

ii. ends with a number

iii. starts with a number

iv. is composed of only uppercase letters and numbers

These rules capture a very small set of gene name characteristics and also fit a lot of non-gene names. However, since the domain of the text is rich with gene names, we can get some good recognitions with these rules.

For improving on this setup, the rules could be applied pre and post POS tagging. The set of rules can also be expanded to include real heuristics like the presence of certain word fragments (carbon, alkali, phosphate) apart from only regular expressions.

Another heuristic could be to check in a dictionary of common nouns and reject any word that is a common noun or check in a dictionary of known gene names and accept any word with a certain level of similarity to a known gene name.

The last set of heuristics mentioned could be applied to POS tagged words as well. In this system, we accept anything with a noun tag. If the words with noun tags were compared to a list of nouns, we could exclude the common ones and improve accuracy.

Aggregation of Annotators

The final output in this system accepts all annotations by both annotators. Instead, there could be voting or weighted combination of the outputs of different annotators. The final words tagged as gene names would be ones that get a high score after combining the results from different annotators.

Results

The system was tested on the sample input and labeled output files provided. These are the results.

Annotator	**Precision**	**Recall**
Simple	1.3%	7.8%
POS Based	9.3%	53%
Both Annotators	10.7%	60.8%