SEPHARDI SRIDHARAN seshadrs@andrew.cmu.edu

PIPELINE OVERVIEW:

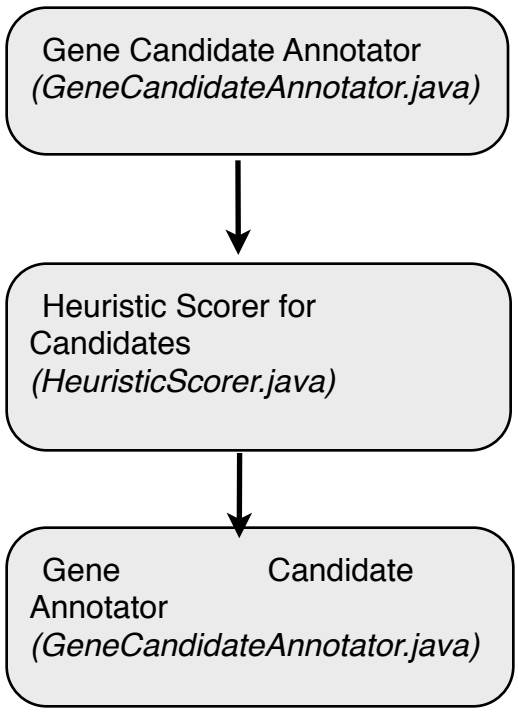
The gene mentions are noun phrases. So, a Part of speech tagger is essential. Also, on close inspection of the given sampe files, it appears that the genes can be distinguished reasonably well with heuristic functions. To accommodate different heuristics, I use a scoring method that assigns a score to every gene hypothesis and then use a pruning function to limit the number of gene labels, thereby improving recall. For modularity, I implemented the POS tagger, the Scorer and the Pruner them as separate analysis engines.

TYPE SYSTEM OVERVIEW:

With this design, I needed just a single custom type : GeneCandidate. It represents a string in a document that is being considered for a gene. It encapsulates all the information needed and manipulated by the collection reader, all the analysis engines, CAS Consumer. and the The type has the following features:

- ***length*** : length of the candidate string
- ***isGene*** : a boolean to indicate whether the candidate is labeled as an actual gene (needed so that the aCsConsumer can consider them seperately)
- ***documentID*** : the document to which the candidate belongs (used at the pruning stage to compare with other candidates from the same document, and at the aCsConsumer when output is written out)
- ***startOffset*** : the startOffset of the text (non whitespace char count before string)
- ***endOffset*** : the endpoint of the text (non whitespace char count before string ends)
- score : the value that indicates the likelihood of the candidate string being a gene.

ANALYSIS ENGINES OVERVIEW:



- Uses Stanford POS tagger to find “NN*” patterns which are considered as Gene Candidates
- Ignores bracket characters
- Sends the documents one by one through the pipeline

- Uses heuristics to score how probable a gene candidate is.
- Uses string length, whether the candidate has an acronym or not, whether the candidate matches common gene regular-expressions.

- Calculates the average score for candidates of a particular document.
- Labels candidates that have a score that is a particular ratio or higher as a gene. (The ratio is determined by experimenting -- later)

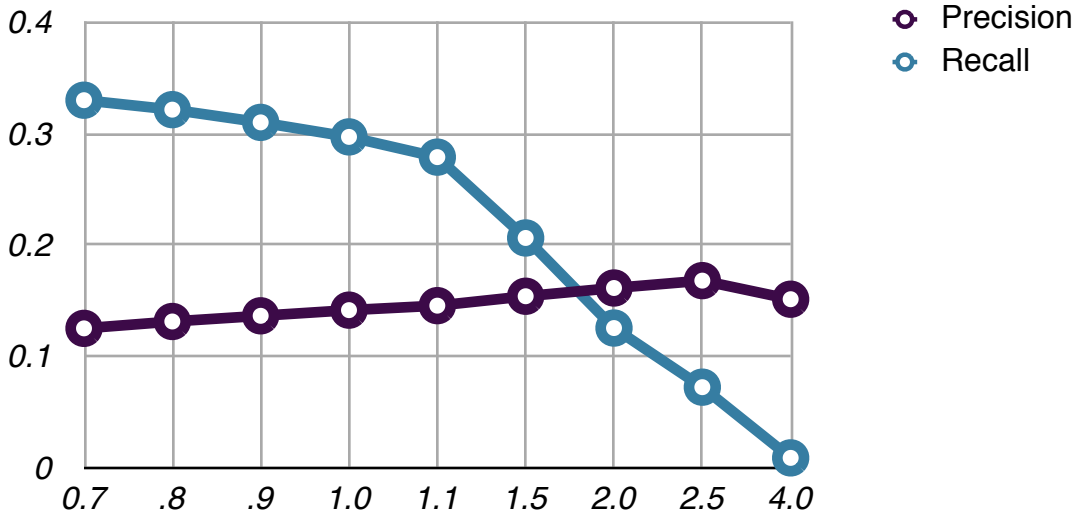
SEPHARDI SRIDHARAN seshadrs@andrew.cmu.edu

RUN TIME PERFORMANCE ANALYSIS: (on given input - 1500 sentences)

Gene Candidate Annotator *(GeneCandidateAnnotator.java)*	Runtime unknown. Uses CRF for POS tagging. 35481ms (87.48%). most expensive analysis engine.
Heuristic Scorer for Candidates *(HeuristicScorer.java)*	Runs in O(N), runs few heuristic scoring functions. 813ms (2%). least expensive analysis engine.
Gene Candidate Annotator *(GeneCandidateAnnotator.java)*	Runs in O(N2), has to loop twice to find average per document. 1092ms (2.69%)

RECOGNITION PERFORMANCE ANALYSIS:

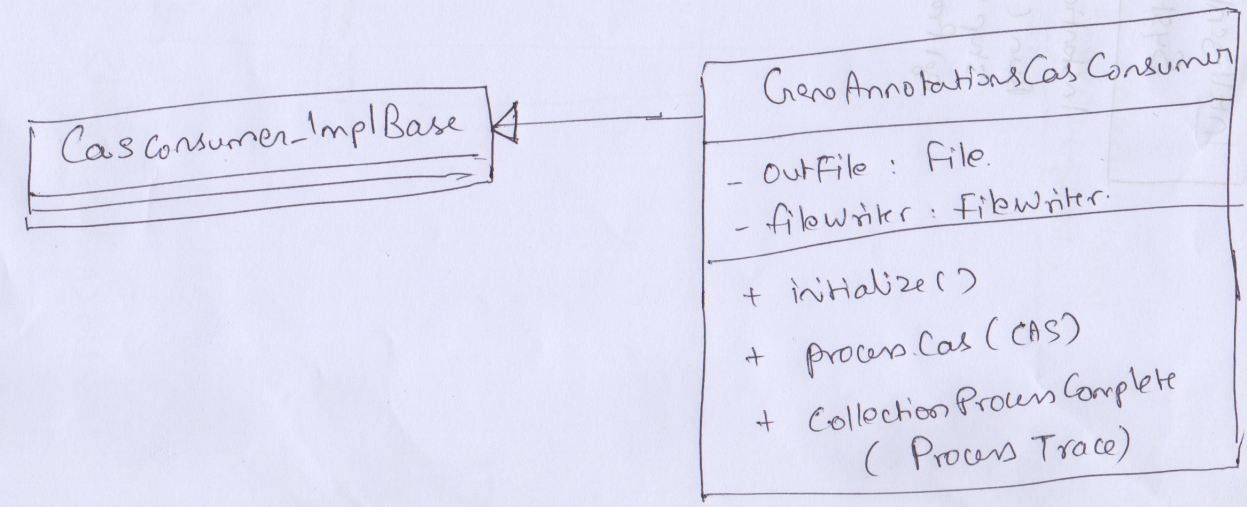
The methods used for recognition were very basic. So, the performance is rather mediocre. Although, usage of the pruning ratio makes room for a small performance experiment. As the pruning ratio is varied, I compute the precision and recall of the system (plotted below). As pruning ratio is increased, more candidates/hypotheses are discarded. So, recall comes down. But, the ones chosen have a higher likelihood of being a gene. So, accuracy goes higher. Towars the extremes, we observe unfavorable performances. A reasonable trade-off seems to be obtained at pruning ratio=1.8. This is the value I use in the system.



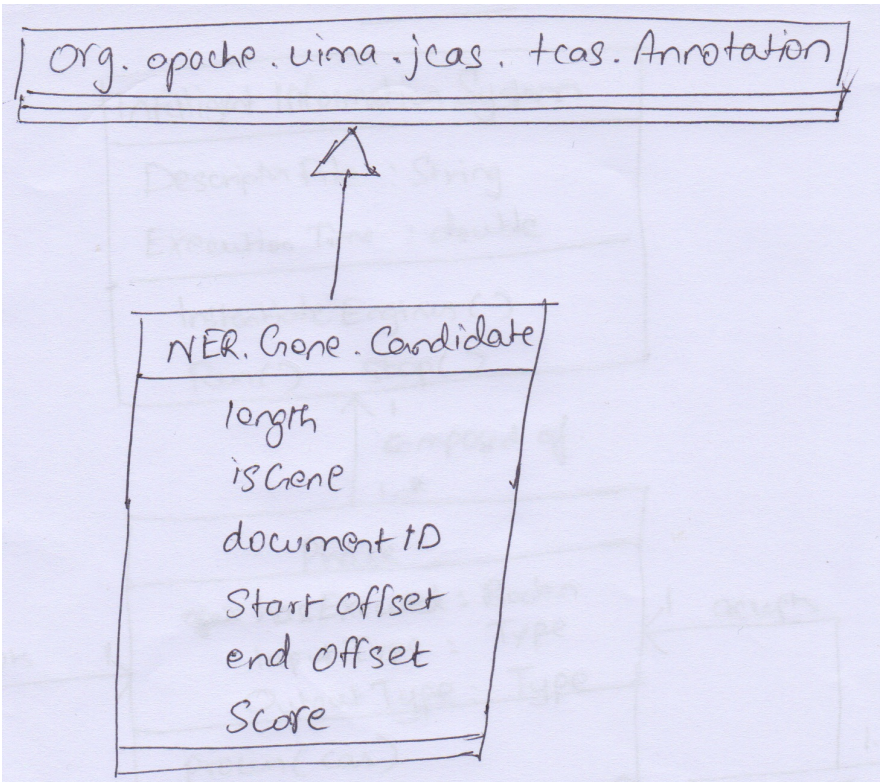
SEPHARDI SRIDHARAN seshadrs@andrew.cmu.edu

UML OVERVIEW:

Cas Consumer:



Collections Reader:



SEPHARDI SRIDHARAN seshadrs@andrew.cmu.edu

Analysis Engines: