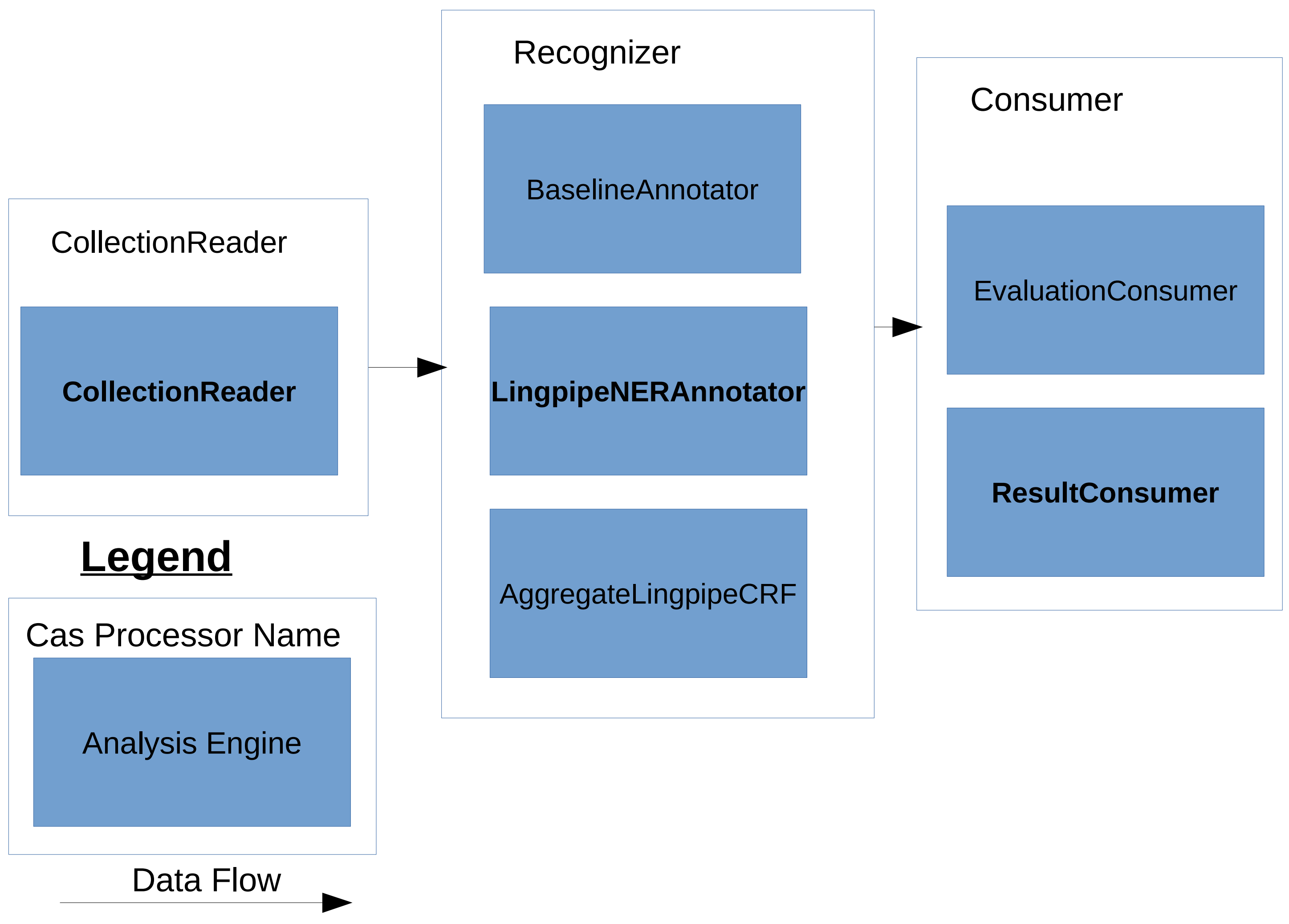
HW 1 Report

Architecture



The overall architecture of my NER system is a pipeline with 3 steps. First, a collection reader that just reads the given input from a parameter name. Then the recognizer stage with 3 possible implementations. The AE written in bold is the one that is executed by the supplied CpeDescriptor.xml. The baseline implementation is just a wrapper to the PosTagNamedEntityRecognizer. The LingpipeNERAnnotator is just the HMM NER chunker of lingpipe. It reads the model as a resource dependency. The AggregateLingpipeCRF is an aggregate analysis engine applying a CRF for finding GeneMentions. It runs 3 analysis engines: LingpipeTokenizer, LingpipePOSTagger (reading the PoSmodel as a resource dependency), and the LingpipeCRFApplicator that reads a custom learned CRF tagging model as a resource dependency and tags the text and converts it to GeneMentions. I think this is a good application for an AAE because tokenization, PoSTagging, and applying the CRF are distinct tasks. Furthermore I was able to reuse the first two stages identically in the training pipeline! (see below)

The Consumer stage has 2 options. First the EvaluationConsumer uses the resources.EvalMap interface to read the gold standard data as a resource and then calculates precision, recall, and F1 and outputs it to STDOUT. The ResultConsumer outputs the file to the given name (parameter) in the prescribed format.

Custom CRF Training Pipeline

I built a separate CPE for training the custom CRF. It reuses the LingpipeTokenizer and LingpipePOSTagger Analysis Engines for preprocessing. In addition the trainCRF package provides 2 additional Analysis Engines: CalbcReader, which is a collection reader that reads sentence-delimited Iexml files (as resource dependency) from http://www.ebi.ac.uk/Rebholz-srv/CALBC/corpora/corpora.html and annotates the training data with the BIEWO scheme (W single word entity, E end of multi-word entity otherwise like BIO). And the CRFTrainer which uses Lingpipes ChainCRF estimate method (Stochastic Gradient Descent) to train the CRF and write the model file to the location and name given as parameter. Also the most important estimation parameters are given as parameters which would allow in the future to use automatic configuration space exploration.

The flow of this pipeline is as follows: CalbcReader reads and annotates the xml, LingpipeTokenizer creates tokens, LingpipePOSTagger adds PoS tags to the tokens, and the CRFTrainer learns the data by calling the estimate method once all CAS have been processed.

Type System

The type system is rather simple:

Type + Features	**Super type**
SentenceMetadata SentenceID :: String	TOP
Token Wordform :: String PartOfSpeech :: String	Annotation
GeneMention MentionText :: String	Annotation

The SentenceMetadata is simply the sentence id that is required for the output. It is not an annotation because it is a global property of a SOFA and not a specific span. The Token type is only used for the custom CRF option, where the text is tokenized and tagged in different AEs than it is analyzed in. The

GeneMention is the main output type against which is evaluated by the EvaluationConsumer. The text of the gene mention is captured in the mention text.

Algorithm and Resources

Baseline

The baseline algorithm just uses Stanford NLP's default resources.

Lingpipe NER

The pretrained NER uses Hidden Markov Models and there are 2 gene mention models pretrained on either Genia corpus or GeneTag (due to different annotation standards Genetag outperforms Genia). Source of the models: http://alias-i.com/lingpipe/web/models.html

Lingpipe Custom CRF

Since the pretrained models are HMMs, I decided to try and see how a simple CRF would perform. Lingpipe provides a CRF implementation and I wrote custom data formats and feature extractors for it (in the trainCRF.helpers package). The data comes from http://www.ebi.ac.uk/Rebholz-srv/CALBC/corpora/corpora.html which is a challenge to normalize gene mention annotations. I only considered the corpora there which already were split into sentences. There are 2: JNLPBA and Biocreative. The Biocreative data seems not to overlap with the given sample data. For the features I took ideas from: A Vlachos, *Tackling the BioCreative2 Gene Mention task with Conditional Random Fields and Syntactic Parsing,* *(*http://www.cl.cam.ac.uk/~av308/biocreative2_GM_vlachos.pdf)

Experiments

I evaluated all approaches on the given data. Since the custom CRF was also not trained on this data, this seems to be a fair comparison. The table values are in percent.

Method	**Precision**	**Recall**	**F1-score**
Baseline	10.25	54.67	17.27
LingpipeNER Genia	17.39	54.52	26.37
LingpipeNER Genetag	**76.85**	**84.88**	**80.67**
Custom CRF JNLPBA (old Feat)	37.70	37.19	37.44
Custom CRF Biocreative (old Feat)	41.87	9.21	15.10
Custom CRF JNLPBA (new Feat)	38.97	37.80	38.38
Custom CRF Biocreative (new Feat)	*48.09*	*50.65*	*49.34*

The baseline performs the poorest. The pretrained Genetag model performs very good and is the submitted version. The pretrained Genia model performs much poorer most likely because of different annotation standards.

For the custom trained CRFs there are 2 corpora, JNLPBA, which probably has also a different annotation standard and a selection of Biocreative (that seems not to overlap with the given sample data.). Without prefix and suffix features (old features) the performance on Biocreative is very poor. Adding the 2-4 character prefixes and suffixes increases performance for the Biocreative data to a respectable 50% on all measures. Interestingly these affix features do not influence the performance of JNLPBA much, maybe because of different annotation standards. It was expected that the custom CRF perform poorer than the pretrained ones, because the training corpora are quite small (Biocreative ~4000 sentences), and to save training time the maximum number of epochs was set to 150, which took about 30 minutes for training one model. Nevertheless, it was possible to train a custom CRF that had reasonable performance. Of course with Configuration Space Exploration as mentioned in the lecture, a better model might have been achieved.

Note that the training data is provided in the resources/data folder. For the Lingpipe models, only the best POStagger and both NER models are provided. For the custom trained models only the best model is provided. All of these are in the resources/models folder. To avoid accidentally overwriting a trained model the output of the training pipeline is in the current working directory.