

Relation Extraction in the Biological Domain

Using Simple Rule Generation and Pairwise
Classification to Extract Genic Interactions from
Bibliographical Databases

Brian Harrington

St. Cross College

`brian.harrington@stx.ox.ac.uk`



Submitted in Partial Completion of the Requirements
for the Degree of Master of Science

Computing Laboratory

Oxford University

Sept 2005

Abstract

This paper presents two Relation Extraction (RE) systems designed to identify genic interactions in unstructured text. The systems are trained, tested, and evaluated using the data sets from the Learning Language in Logic (LLL05) Genic Interaction Extraction Challenge.

The first system uses the $(LP)^2$ algorithm as implemented in the information extraction program Amilcare [Ciravegna, 2003] to identify agent and target entities. The rules developed by this system independently identify entities as either agents or targets; a secondary process is then used to link the agents and targets into binary relations.

The second system uses well-established classification techniques on entity pairs within the text. For each pair of entities an attribute list is created consisting of the POS categories of the words separating the entities, and the linguistic relations connecting the entities. The attribute lists are then used as input for classifiers which attempt to classify the existence and direction of relations between the entity pairs.

Despite the relative simplicity of both systems, they perform very well with respect to the official results of the challenge, receiving top scores in all testing categories.

Acknowledgements

I would like to thank my supervisor, Stephen Clark, for his tremendous assistance throughout the project. From providing the initial impetus for this particular project, to correcting my (all too numerous) grammatical errors, Stephen has been an amazing help at every stage of the development of this project, and has certainly gone above and beyond the call of duty in his role as supervisor.

I would also like to thank my parents. Without their moral (not to mention financial) support, I certainly would not have been able to pursue my academic career to this stage.

Contents

1	Introduction	1
1.1	Information Retrieval	1
1.2	Information Extraction	2
1.3	Relation Extraction	3
1.4	Information Extraction Motivation	5
1.5	Biological Motivation	5
2	The Problem	7
2.1	The LLL05 Challenge	7
2.1.1	The Challenge Data	8
2.2	Challenge Results	11
3	Methods	15
3.1	Amilcare	15
3.2	Pairwise Classification	17
3.2.1	Classification	18
3.2.2	Weka	18
3.2.3	Linguistic Relation Information	19
3.2.4	Morpho-Syntactic (POS) Categories	22
3.2.5	Weighted Voting	23
3.2.6	Classifiers Used	24
4	Results	29
4.1	Amilcare Results	29
4.2	Pairwise Classification Results	31
4.3	Comparison of Results	33
5	Conclusions	35
5.1	Links to Coursework	36
5.1.1	Intelligent Systems	36
5.1.2	Information Retrieval	37
5.1.3	Computational Linguistics	37

A	Code	41
A.1	Amilcare System Code	41
A.1.1	Training Data Formatting Script	41
A.1.2	Test Data Formatting Script	44
A.1.3	Result Formatting Script	46
A.2	Pairwise Classification Code	48
A.2.1	Training Data Formatting Script	48
A.2.2	Test Data Formatting Script	54
A.2.3	Weighted Voting & Results Formatting Script	59

Chapter 1

Introduction

1.1 Information Retrieval

Information retrieval (IR) covers a broad range of applications, from document retrieval (e.g. internet search engines), to database querying, to information extraction (IE). Information retrieval can include searching for information within documents, searching for the documents themselves, searching in highly structured databases, searching metadata which describes the documents, and even searching non textual information such as images or video files.

While information retrieval is a topic much too large to be covered here, there are three particular concepts related to IR which should be introduced at this stage: precision, recall and F-measure. These measures are used across all of IR to evaluate the ability of a system to perform an IR task.

Precision is the proportion of the answers obtained by the system which are correct, i.e. the number of answers which were correct divided by the total number of answers which were returned by the system. In the case of document retrieval we may define precision as:

$$P = \frac{\text{Number of relevant documents retrieved}}{\text{Total number of documents retrieved}} \quad (1.1)$$

Recall is the proportion of possible correct answers which the system manages to find, i.e. the number of correct answers the system finds divided by the total number of correct answers that exist in the space being searched.

In the case of document retrieval we may define recall as:

$$P = \frac{\text{Number of relevant documents retrieved}}{\text{Total number of relevant documents in search space}} \quad (1.2)$$

One of the key tasks in Information Retrieval is often balancing precision and recall. For example in the case of document retrieval, a system could easily get a very good score for precision if it only returned a very small number of documents and ensured that those documents were relevant. However in this case the recall score would be very low. Obtaining a high recall score is even simpler, all a system has to do to achieve perfect recall is to return all the documents in the search space. However this would result in a very poor precision value.

This brings us to the third concept, the F-measure. In order to balance precision and recall, we must have a final result which rewards high scores in both precision and recall, but penalizes a low score in either measure (even if it results in a high score in its counterpart). The F-measure is the harmonic mean of precision and recall, and is generally used as the primary indication of the success of an IR system. We define the F-measure as:

$$F = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (1.3)$$

1.2 Information Extraction

Information extraction (IE) is a sub-task of information retrieval where the goal is to extract structured information from unstructured text (e.g. “plain text” as found in newspapers or books). One example of an information extraction task is to extract named entities such as names of persons or companies from newspaper texts. More complex IE tasks might be to extract attributes referring to a specific entity, or relationships between entities.

The following tables adapted from the SAIC Information Extraction Website [[SAIC IE Website](#)] are examples of the results of different IE tasks. These tables show the different types of information that can be extracted from the following text.

Fletcher Maddox, former Dean of the UCSD Business School, announced the formation of La Jolla Genomatics together with his two sons. La Jolla Genomatics will release its product Geninfo in June 1999. Geninfo is a turnkey system to assist biotechnology

researchers in keeping up with the voluminous literature in all aspects of their field.

Dr. Maddox will be the firm’s CEO. His son, Oliver, is the Chief Scientist and holds patents on many of the algorithms used in Geninfo. Oliver’s brother, Ambrose, follows more in his father’s footsteps and will be the CFO of L.J.G. headquartered in the Maddox family’s hometown of La Jolla, CA.

Entity recognition is the task of extracting individual entity labels from the text as shown in Table 1.1. Notice that in the table multiple labels may refer to the same entity. Attempting to reconcile the labels with the entities to which they refer is a separate IE task known as coreference identification.

Table 1.1: Example of Entity Recognition

Persons:	Fletcher Maddox, Dr. Maddox, Oliver, Oliver, Ambrose, Maddox
Organizations:	UCSD Business School, La Jolla Genomatics, La Jolla Genomatics , L.J.G
Locations:	La Jolla, CA
Dates:	June 1999

Attribute recognition is the task of isolating the statements made about each entity as shown in Table 1.2 ¹. Notice that some of the descriptors are self contained (e.g. “former Dean of the UCSD Business School”), while others refer to secondary entities not given in the descriptor (e.g. “his son”). During attribute recognition, no attempts are made to “link up” the descriptors with the secondary entities to which they refer; that is done in another task to which the next section is devoted.

1.3 Relation Extraction

Once the entities in a section of text have been identified, the next step is to discover if and how they interact; this is the task of relation extraction. The goal of relation extraction is to extract semantic relationships between entities as expressed in unstructured text. Examples of relationships that might be extracted from from the text in the previous section are included in Table 1.3. Note that while some of the relations could be

¹For the sake of simplicity Table 1.2 assumes coreference identification has already taken place.

Table 1.2: Example of Attribute Recognition

Entity:	Fletcher Maddox	Oliver	Ambrose	La Jolla
Category:	PERSON	PERSON	PERSON	PLACE
Attributes:	former Dean of the UCSD Business School, his father, the firm's CEO	His Son, Chief Scientist	Oliver's Brother, the CFO of L.J.G.	the Maddox family's hometown

easily interpreted from the information surrounding the entity, others (such as *employee_of(Ambrose, La Jolla Genomatics, CFO)*) require information spanning several sentences.

Table 1.3: Example of Relation Extraction

<i>employee_of</i>		
PERSON	ORGANIZATION	POSITION
Fletcher Maddox	UCSD Business School	Dean
Fletcher Maddox	La Jolla Genomatics	CEO
Oliver	La Jolla Genomatics	Chief Scientist
Ambrose	La Jolla Genomatics	CFO
<i>location_of</i>		
ORGANIZATION	CITY	STATE
La Jolla Genomatics	La Jolla	CA

The majority of this paper will be concerned with relation extraction, and in particular a subset of relation extraction where the desired types of relations are known prior to processing. (It is also possible in certain situations to have tasks where one of the goals is to determine the type of relation between entities, as opposed to simply determining which if any relation they possess out of a list of possible relations known a priori.) This paper will also primarily concern itself with binary relations, although there are efficient ways to extend any binary relation extraction scheme to be useful for arbitrarily complex relations[[McDonald et al., 2005](#)].

1.4 Information Extraction Motivation

Recent work in information extraction has been heavily focused on relation extraction. This is partially due to relation extraction’s potential usefulness as a tool, and partially due to its difficulty as a task. With the simple named entity recognition F-measures in MUC-7 ² reaching above 90% [Marsh and Perzanowski, 1998], and more recent systems easily able to achieve F-measures in the 97-98% range for simple entity recognition [Maynard et al., 2002], the information extraction world is turning its attention to more complex tasks. Relation extraction is a much more difficult task than entity recognition, often requiring deep linguistic analysis methods to extract the relevant information, and complex algorithms to process the information once extracted. Furthermore relation extraction often encompasses other information extraction tasks such as entity recognition and coreferencing, which it requires as part of its initial processing of the data.

The development of better relation extraction techniques would be beneficial to many information retrieval tasks. The ability to extract structured information from unstructured text would be a tremendous help to a wide variety of projects. Potential application areas range from database creation from an unstructured corpus as described in this thesis, to tasks such as semantic web annotation [Iria and Ciravegna, 2005], to the development of question answering systems [Loper, 2000].

1.5 Biological Motivation

Bibliographic databases such as Medline are immensely valuable to biological researchers; however with their vast size and quick growth rates it is becoming increasingly difficult for researchers to find information efficiently in these databases, thus reducing their effectiveness as a research tool. For example if a biological researcher wishes to find information on transcription in *Bacillus Subtilis*, the query “Bacillus subtilis and transcription” returns 2,693 abstracts (up from 2,209 in 2002) [Nédellec, 2005]. Sorting through these documents for the few which may be useful to the particular information need requires a great deal of time. There is no structured information resource for much of this information, and thus biologists are often forced

²The Message Understanding Conferences (MUC) were a series of conferences and challenges held during the 1990s, which (among other things) challenged participants to perform various information extraction tasks, including named entity recognition. MUC 7 [Chinchor, 1998] was the 7th such conference, which focused on Named Entity Recognition, Information Extraction and Coreferencing tasks.

to read through hundreds or even thousands of abstracts before finding the information they require.

The development of methods to produce structured information from these databases of unstructured text would be extremely valuable to the biological community. A structured resource would allow our hypothetical biological researcher to write a single query to retrieve all the transcription relations in *Bacillus Subtilis*. Instead of the thousands of abstracts provided by querying the unstructured corpus, the query on the structured corpus might result in a few hundred well formed results; this would obviously save a tremendous amount of time and energy.

Chapter 2

The Problem

2.1 The LLL05 Challenge

This thesis details a system designed for and tested by the Learning Language in Logic (LLL05) Genic Interaction Extraction Challenge [[LLL05 Challenge Website](#)]. Although this system was not entered in the challenge, since the challenge occurred midway through the production of this thesis, the training and testing data from the challenge were used. Also, the system was evaluated using the challenge's evaluation program, which was made available on the challenge website.

The purpose of the challenge was to devise a system which would learn rules to extract protein/gene interactions from biology abstracts which were taken from the Medline database¹. Participants were given training data consisting of sentences from the Medline databases in which the agents and targets of genic interactions had been identified by expert biologists, a dictionary of all proteins and genes (along with all typographic variants and synonyms) used, and linguistic information for the training data including syntactic dependencies, lemmatization and word segmentation². After designing a system using this training data, participants in the challenge were given test data consisting of sentences from biological abstracts taken from the medline database, and asked to extract as many genic interactions as possible from

¹All abstracts collected referred to *Bacillus subtilis* (*Bs*), a model bacterium on which many papers have been written, particularly relating to gene interactions involved in sporulation.

²The linguistic data had been automatically produced by LinkParser [[Sleator and Temperley, 1993](#)], and corrected by specialists from the Mathématique Informatique & Génome (MIG) group of the Institut National de la Recherche Agronomique and the Laboratoire d'Informatique de l'université Paris-Nord (LIPN).

this data.

2.1.1 The Challenge Data

The training data consists of 3 distinct models of genic interactions:³,

- Explicit action: “*GerE stimulates cotD transcription*”
- Binding of the protein on the promoter of the target gene: “*Therefore, ftsY is solely expressed during sporulation from a sigma(K)- and GerE-controlled promoter that is located immediately upstream of ftsY inside the smc gene*”
- Membership to a regulon family: “*yvyD gene product, being a member of the sigmaB regulon [...]*”

The training data is divided into distinct sections by these three expressions; the testing data however is not, and thus it is necessary to process the testing data against rules for all three types of expressions.

Participants in the challenge were provided with a dictionary of all named entities (gene/protein names, gene family names, etc.). Since many of the entities can be referenced by many synonymous names and typographic variations (e.g. sigmaF, sigma(F) and sigma 28 are simply typographic variants of sigF) the dictionary provides a canonical form for each entity to remove ambiguity.

Participants were also provided with linguistic information for each sentence, which included lemmas and syntactic relations. The lemmas give the canonical forms of words such as the infinitive forms of verbs, and the canonical forms of named entities as given by the dictionary. The syntactic relations give relations between words in the sentence such as apposition, complementation, modification, and subject relationships⁴. For each word that was involved in a syntactic relation the morpho-syntactic category⁵ was also provided.

The training data is split into two subsets, one which contains coreferences, and one which does not. Applying IE to coreferenced sentences is assumed to

³Examples taken from [Nédellec, 2005]

⁴For a full list of relations provided, see [Aubin, 2005].

⁵Commonly known as “part-of-speech”, the morpho-syntactic category is the high-level linguistic category to which a word belongs, in this case defined as one of: verb, passive verb, noun, adjective, adverb.

be a more difficult task due to the indirectness of the link between target and agent terms. For example in the sentence “*We have shown previously that the transcription of degR is driven by an alternative sigma factor, sigmaD.*”, the target(*degR*) and the agent(*sigmaD*) are mediated by the phrase “*an alternative sigma factor*”. For a simpler example, consider the sentences “*The yellow dog sat down*” and “*The dog sat down, it was yellow*”. The first sentence contains no coreferences, and thus determining that the dog was yellow should be simple. In the second sentence, one must realize that “*it*” refers to “*The dog*” in order to deduce the dog’s colour, this is known as solving the coreference. This is an example of the extra difficulty created by coreferencing. The separation of the data into these two subsets allows systems to be evaluated on two distinct levels, the coreferencing adding an extra level of difficulty.

Below is an example entry from the training data. The format of the testing data is identical except for the absence of the agent, target, and genic_interactions sections.

```

ID          10515909-4
sentence    Expression of the sigma(K)-dependent cwIH gene
            depended on gerE.
words       word(0, 'Expression ', 0, 9)
            word(1, 'of ', 11, 12)
            word(2, 'the ', 14, 16)
            word(3, 'sigma(K) ', 18, 25)
            word(4, 'dependent ', 27, 35)
            word(5, 'cwIH ', 37, 40)
            word(6, 'gene ', 42, 45)
            word(7, 'depended ', 47, 54)
            word(8, 'on ', 56, 57)
            word(9, 'gerE ', 59, 62)
lemmas      lemma(0, 'expression ')
            lemma(1, 'of ')
            lemma(2, 'the ')
            lemma(3, 'sigK ')
            lemma(4, 'dependent ')
            lemma(5, 'cwIH ')
            lemma(6, 'gene ')
            lemma(7, 'depend ')
            lemma(8, 'on ')
            lemma(9, 'gerE ')
syntactic_relations  relation('subj:V-N', 7, 0)
                    relation('mod_att:N-ADJ', 6, 4)
                    relation('mod:ADJ-N', 4, 3)

```

```

relation ('mod_att:N-N',6,5)
relation ('comp_of:N-N',0,6)
relation ('comp_on:V-N',7,9)
agents  agent(3)
        agent(9)
targets target(5)
genic_interactions  genic_interaction(3,5)
                   genic_interaction(9,5)

```

The following is an explanation of the data fields:

- *ID*: The unique identification number for each sentence.
- *sentence*: The sentence exactly as it appeared in the abstract.
- *words*: Composed of the word number, the word itself, and placement of the first and last characters of the word in the sentence (e.g. word(6,'gene',42,54) means that 'gene' is the 6th word in the sentence, and that its first letter is the 42nd character in the sentence, while its last character is the 54th).
- *lemmas*: The word number and the lemmatization of the word (such as 'depend' the infinitive form of 'depended').
- *syntactic_relations*: Composed of a relation between two words as well as the POS categories and word order of those words (e.g. relation('mod_att:N-ADJ',6,4) means that 'dependent' is an attributive modifier of 'gene', also it tells us that 'gene' is a noun and 'dependent' is an adjective).
- *agents/targets*: The word number of an agent/target.
- *genic_interaction*: The word numbers of an agent-target pair that interact (e.g. genic_interaction(9,5) means that gerE is the agent and cwlH is the target of a particular interaction).

The training data set without coreferences consisted of 57 sentences containing:

- 70 examples of explicit action;
- 30 examples of binding of the protein on the promoter of the target gene;
- 6 examples of membership to a regulon family.

The training data containing coreferences consisted of 23 sentences containing:

- 42 examples of explicit action;
- 10 examples of binding of the protein on the promoter of the target gene;
- 7 examples of membership to a regulon family.

The testing data used for the challenge consisted of 87 sentences containing:

- 55 examples of explicit action;
- 23 examples of binding of the protein on the promoter of the target gene;
- 5 examples of membership to a regulon family.

It should be noted that the training data does not explicitly describe negative instances. However, approximately 50% of the test data was comprised of negative instances (i.e. sentences which contained no genic interaction). The generation of negative examples (if they were necessary for the learning algorithms used) was left up to challenge participants.

2.2 Challenge Results

6 groups participated in the LLL05 challenge, using a diverse array of methodologies and tools which achieved varying levels of success. Some groups chose to use the linguistic information provided by the challenge organizers, while others used tools to create their own linguistic data. While the official results of the challenge[Nédellec, 2005] were separated by whether or not the group used the provided linguistic data, for our purposes we have amalgamated those scores. (In cases where both a “basic” and an “enriched” score were reported, we will take the score of higher value). Some groups

used the provided linguistic information in addition to linguistic information generated by other means, some groups used one or the other of these types of linguistic information, and some groups used neither. This made comparison across groups very difficult, and did not provide a great deal of extra information relevant to our purposes. Therefore the scores were amalgamated across these groupings.

Brief descriptions of the systems submitted by each group, and the scores, are given below:

- **Group 1:** [Hakenberg et al., 2005] This was system developed jointly by the Knowledge Management in Bioinformatics group at Humboldt-Universität, Germany and the Rebholz-Group of the European Bioinformatics Institute, Hinxton. It used sequence alignments, and finite-state automata optimized with a genetic algorithm. This system was trained on an additional 256 hand-annotated examples in addition to the challenge training data.
- **Group 2:** [Greenwood et al., 2005] This system developed at the University of Sheffield learns extraction patterns by comparing the tree structures of sentences against manually created *seed patterns*. This system was trained on 78 *weakly labelled* [Craven and Kumlien, 1999] sentences from the Medline database in addition to the training data supplied for the challenge.⁶
- **Group 3:** [Katrenko et al., 2005] This system developed at the University of Amsterdam created lexical-semantic-syntactic trees for each sentence, and then applied the Ripper algorithm[Cohen, 1995] to subtrees of the sentence for rule induction.⁷
- **Group 4:** [Popelínský and Blaťák, 2005] This system developed at the Knowledge Discovery Lab at Masaryk University in Borno, Czech Republic, separated the learning of simple (containing 2 dictionary words) and complex (containing 3 or more dictionary words) sentences in the training data. The Aleph ILP program [Srinivasan, 2003], and the RAP algorithm [Popelínský et al., 2003] were used to induce rules for both simple and complex sentences. The submitted scores are from the Aleph program.

⁶This data for group 2 in Table 2.1 is not consistent with that described in the official challenge results [Nédellec, 2005], as the scores reported there were for the “baseline” system which simply returned all possible combinations of agent/target pairs, and not for their “main” system, the scores for which are reported in this table.

⁷The challenge paper claims that the scores obtained by this system were in fact 39.2%(precision) 26.5%(recall) and 31.6%(F-measure), and the lower scores submitted to the official challenge were the result of a bug in the system.

- **Group 5:** [Goadrich et al., 2005] This system developed at the Department of Biostatistics and Medical Informatics and Department of Computer Sciences at the University Wisconsin-Madison also used Aleph [Srinivasan, 2003]; however unlike Group 4, the clauses generated by Aleph were then used as input to the randomized search method Gleaner[Goadrich et al., 2004], which selects the best point along a precision-recall curve to decide which clauses to use on the testing data. This system augmented the testing training data with 215 manually chosen predicates to provide additional information about aspects of the data such as sentence structure, lexical properties, word frequencies and ontology words.
- **Group 6:** [Riedel and Klein, 2005] This system developed at the Institute for Communication and Collaborative Systems at The University of Edinburgh used the CCG2sem parser [Clark and Curran, 2004][Bos, 2005] to convert the sentence into a set of semantic relations. They then used Markov Logic to create a set of weighted clauses which could be used to identify relations.⁸

The challenge results are split into three groups: those groups which reported scores only for the non coreferenced data (Table 2.1), those groups which only reported scores for the coreferenced data (Table 2.2), and those groups which reported a score for the union of both sets of data (Table 2.3). In the challenge, the testing data was the same for both groups, and the category was determined by passing a parameter to the evaluation program to instruct it as to which relations should be taken into account when calculating the score.

Table 2.1: Challenge Results without Coreference Only

Group #	Pre	Rec	F
1	50.0	53.8	51.8
2	22.2	11.1	14.8
4	37.9	55.5	45.1
5	25.0	81.4	38.2
6	60.9	46.2	52.6

⁸The challenge paper claims that after the manual addition of explicit rules for non-interaction, the scores rose from those reported in the official challenge results[Nédellec, 2005] to 68.4% (F-measure) for non coreferenced data and 64.7% (F-measure) for the joint data set.

Table 2.2: Challenge Results with Coreference Only

Group #	Pre	Rec	F
5	14.0	93.1	24.4

Table 2.3: Challenge Results with and without Coreferences

Group #	Pre	Rec	F
3	51.8	16.8	25.4
6	55.6	53.0	54.3

A noteworthy feature of these systems is the complexity of the algorithms used to achieve these scores. Most of the algorithms required deep natural language processing and computationally complex algorithms. We will show in the following chapters that similar or superior results can be obtained through much simpler algorithms, having lower computational costs and a much lower degree of complexity than many of the systems entered in the LLL05 challenge.

Chapter 3

Methods

3.1 Amilcare

Amilcare [[Amilcare Website](#)] is a program designed primarily for document annotation in the Semantic Web. It utilizes the $(LP)^2$ Algorithm [[Ciravegna, 2001](#)][[Ciravegna, 2003](#)] in order to perform named entity recognition based on XML markup and linguistic processing (such as lexical and semantic categorization) of the words surrounding entities.

The ultimate goal of Amilcare is to learn rules for the placement of XML tags in a document, based on where tags are placed in the training data. The algorithm attempts to build independent rules for each type of tag (including independent rules for opening and closing tags of the same XML type). Initially the rules are necessarily over-fit to the particular sentence, as seen in Table 3.1. All information available is used in the rule, including the words themselves, their order in the sentence, the lemmatization of the word (infinitive forms of verbs, canonical forms of dictionary words, etc.)¹, case information (upper case, lower case, or mixed) and Semantic Categories (as defined by a Gazetteer, augmented with entity names from the provided dictionary). These rules are obviously not of great value before generalization, as they over-fit the data to the extent that they will only be applicable to precisely the same sentence in the testing data.

The Generalization process of the $(LP)^2$ algorithm attempts to discover for each rule, which of the pieces of information (conditions) are most helpful for the rule to retain, and which conditions can be discarded. It does this by systematically testing all possible generalizations (a generalization in this

¹The canonical form of yfhR is fabL as defined in the challenge dictionary

Table 3.1: Example $(LP)^2$ Rule Before Generalization

	Word	Lemma	LexCat	Case	SemCat	Entity
1	yfhR	fabL	Other	mixed	Gene	agent
2	stimulates	stimulate	Active Verb	low		
3	cotD	cotD	Other	mixed	Protein	target
4	transcription	transcribe	Passive Verb	low		

case is defined as a rule with one or more conditions deleted) against the other sentences in the training set, only keeping the rules which provide an adequate balance of precision and recall. In Table 3.2 we see the same rule as shown in Table 3.1 after one particular generalization. This rule could now be read as “A Gene, followed by an active verb, followed by a Protein followed by a passive verb should be marked with the 1st word as the agent and the 3rd word as the target”².

Table 3.2: Example $(LP)^2$ Rule After Generalization

	Word	Lemma	LexCat	Case	SemCat	Entity Type
1					Gene	agent
2			Active Verb			
3					Protein	target
4			Passive Verb			

The chief interest in using Amilcare for this task is that it independently tags each entity, and so rather than creating a (necessarily more complex) template based on both agent and target identification, it simply creates a rule for the identification of the agent and a second rule for the identification of the target. In many respects this was an attempt to use technology created for a simpler task (entity recognition) on a more complex domain (relation extraction).

After training Amilcare and receiving its tagged output, a secondary processing stage was needed to match the appropriate agents to targets. In sentences where there was only 1 agent or 1 target, there was no need for an extended algorithm as either a single agent could be matched with all targets, or a single target could be matched with all agents. (In cases where

²This actually describes 4 rules according to the $(LP)^2$ algorithm, one rule for the placement of each of $\langle agent \rangle$, $\langle /agent \rangle$, $\langle target \rangle$ and $\langle /target \rangle$

there was either no agent or no target, obviously this stage can be completely omitted). The more difficult task lay in the matching within sentences that contained 2 or more agents and 2 or more targets. Several algorithms were considered for this processing, including matching by proximity, by primacy, or simply matching all possible pairs. After preliminary tests, matching all possible pairs was decided upon as the best strategy. This method raised the recall value enough to compensate for its hindrance to the precision value, and thus gave the highest overall F-measure. This is the method used to obtain the results given in Section 4.1.

3.2 Pairwise Classification

One group in the LLL05 competition [Greenwood et al., 2005] developed a “baseline system” which assumed that there was an interaction among every pair of named entities in a sentence. For example, if a sentence had 3 entities labelled *entityA*, *entityB* and *entity C*, this baseline program would report 6 interactions, namely $(entityA, entityB)$, $(entityA, entityC)$, $(entityB, entityA)$, $(entityB, entityC)$, $(entityC, entityA)$ and $(entityC, entityB)$. This system (as expected) had a very high recall score, but very low precision due to the large number of incorrect interactions reported. It was this baseline system that provided the initial inspiration for the pairwise classification method described in this section.

Rather than arbitrarily assuming that all pairs of entities necessarily interact, we considered whether it would be possible to devise a means of constructing for each pair of entities the probability that they interact, as well as the most likely direction of their interaction (i.e. which is the agent, and which the target of the interaction). The pairwise classification method described in this section performs precisely that function.

The task of identifying individual entities in this case was trivial, as all possible entities were provided by the challenge organizers in the dictionary. This allowed the systems in the competition to focus solely on relation extraction and not have their data skewed by using differing entity extraction methods.

Once all of the entities have been identified, all possible pairs of entities are created (since we are checking for direction as well as existence we will always choose to order the entities as they appear in the text) and then classified as either “NEG” (No relation exists) “AtoT” (the first entity is the agent, the second is the target) or “TtoA” (the first entity is the target the second is the agent).

3.2.1 Classification

In order to perform classification, each pair of entities was translated into a list of attributes and a classification. For example, if the attributes being used were number of words between entities, the POS categories of all words between entities, and all punctuation between entities, then the sentence:

“Expression of the sigmaK-dependent cwIH gene depended on gerE.”

would be translated to the following attribute lists:

Relation	Length	POS	Punctuation	Classification
(sigmaK,cwIH)	1	“ADJ”	“_”	AtoT
(cwIH,gerE)	3	“N,V”	None	TtoA
(sigmaK,cwIH)	5	“ADJ,N,N,V”	“_”	NEG

Where length denotes the number of words between the entities, POS denotes the POS categories of all words between the entities, punctuation denotes all non-alphanumeric characters between the entities, and classification denotes the type of relation (AtoT meaning the first entity is the agent, the second entity is the target, TtoA being the reverse, and NEG meaning that there is no relation between the entities).

3.2.2 Weka

Once the data had been translated into an attribute list, there were a wide variety of classification algorithms which could be used to determine the existence and direction of the relation for each pair of entities. For this purpose the Waikato Environment for Knowledge Analysis (WEKA)[[Witten and Frank, 2005](#)] tool was used.

WEKA allowed us to train several different classifiers on our data, and quickly receive feedback as to how well various classifiers worked for this particular task. While not all classifiers included with the WEKA tool were tested (some classifiers were incompatible with the data types used in this instance), the majority of the classifiers were evaluated on the training data using ten-fold cross validation testing for development purposes. From that testing, classifiers were chosen to be run against the testing data for

evaluation purposes ³.

The details of classifiers used are given later in this section, however at this stage it should be noted that the classifiers were not chosen on their prediction ability alone. We used results from multiple classifiers (see Section 3.2.5) to obtain our final results. Therefore it was in the best interest of the system to use classifiers which produced the best combined results and not simply choose classifiers based on individual performance. Since several of the classifiers were based on similar principles (e.g. The set of Bayesian classifiers), they were likely to produce similar results, and thus running several similar classifiers would produce results without much improvement over those obtained from a single classifier. For that reason, classifiers were chosen which not only gave good scores on the ten-fold cross validation, but that also, by inspection of results, appeared to give classifications dissimilar to other high scoring classifiers.

3.2.3 Linguistic Relation Information

The enriched training and testing datasets provided to the LLL05 challenge participants included syntactic relations linking pairs of words within the sentences. These relations were comprised of a relation type (complement, modifier, object or subject relations) and an optional specification (e.g. a modifier could be an attributive modifier, a post-posed modifier, or a predicative modifier). These relations were created by LinkParser [Sleator and Temperley, 1993], and corrected by specialists from MIG and LIPN.

If the words in a sentence are represented by nodes, and the syntactic relations are represented by directed edges (since the relations themselves were directed, undirected edges would have been insufficient to capture all the information given by the relations), we can view the combination of the two as a directed graph. For example the graph of the following sentence taken from the testing data can be seen in Figure 3.1 ⁴.

³Although at this time it would have been possible to evaluate the classifiers against the test data, we decided to choose this method of evaluating classifiers to simulate conditions in the actual challenge where participants did not have access to the evaluation programs.

⁴The edge labels in Figure 3.1 have been changed to their extended form for increased readability.

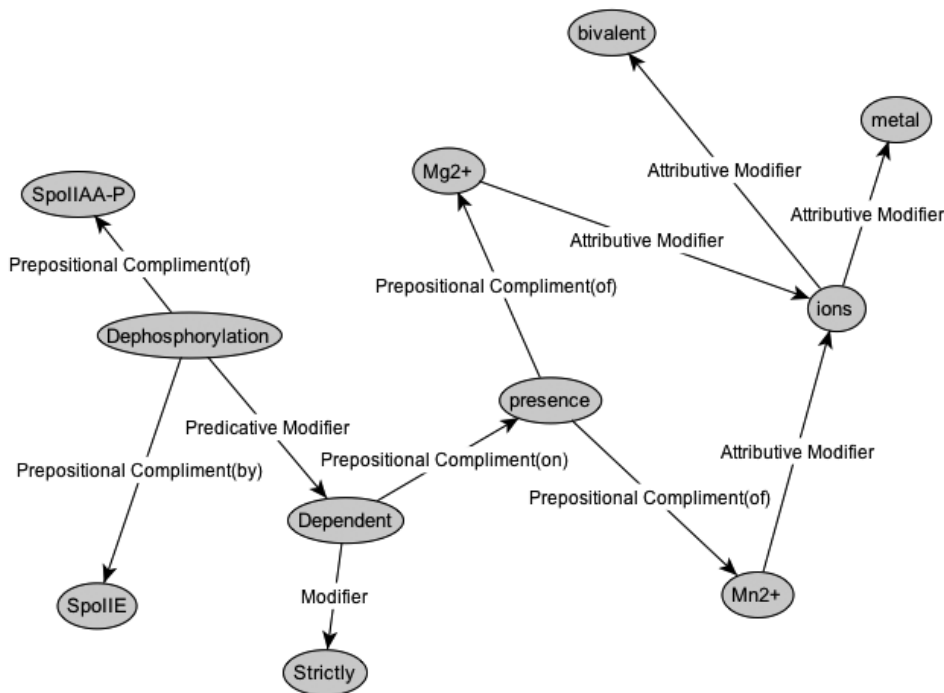
Sentence:

“Dephosphorylation of SpoIIAA-P by SpoIIE is strictly dependent on the presence of the bivalent metal ions Mn²⁺ or Mg²⁺.”

Syntactic Relations:

- relation('comp_of:N-N',10,16)
- relation('comp_of:N-N',0,2)
- relation('mod_att:N-N',15,14)
- relation('mod_pred:N-ADJ',0,7)
- relation('mod_att:N-ADJ',15,13)
- relation('comp_of:N-N',10,18)
- relation('comp_by:N-N',0,4)
- relation('mod:ADJ-ADV',7,6)
- relation('comp_on:ADJ-N',7,10)
- relation('mod_att:N-N',18,15)
- relation('mod_att:N-N',16,15)

Figure 3.1: Graph of Linguistic Relations

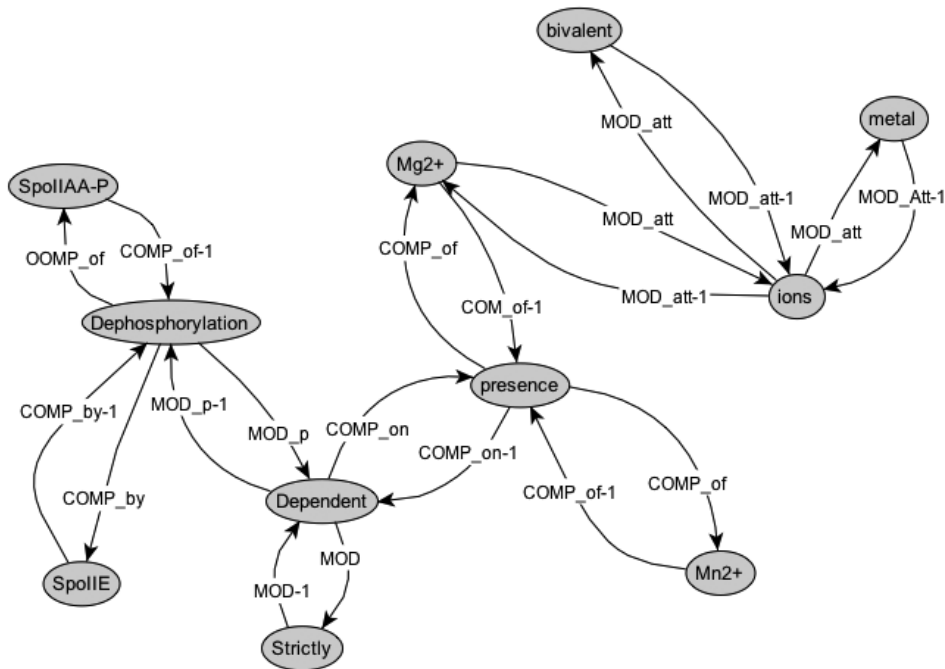


In order to use this information in our classification scheme, we decided to use as a list of attributes, all relations linking two entities (i.e. the list of edges in the graph linking the nodes representing the entities). Since the graphs may be cyclical there is the possibility of multiple paths between entities; in these cases we opted for using only the shortest path. A simple breadth first search beginning at the node representing the entity produced the shortest path (if such a path existed) to any other node, and in particular

to the node representing the entity with which our original node was to be paired.

The breadth first search method, once implemented, was quickly seen to have a fault insofar as that it very rarely produced any path information between nodes (e.g. in Figure 3.1 neither entity has an arrow originating from its node, thus no path could be constructed to connect the entities). However, a precursory examination of the data suggested a solution. The insertion of edges in the opposite direction of relations greatly increased the amount of useful information which was returned. This was very simple to implement: for each labelled edge $(a,b) = \textit{“label”}$ (where *“label”* is the name of a relation), we simply added an additional edge $(b,a) = \textit{“label-1”}$. As seen in Figure 3.2. This allowed the breadth first search to search relations in both directions, and allowed the algorithm to find paths between entities in many more instances.

Figure 3.2: Graph of Linguistic Relations With Backwards Edges



The attribute list obtained through this method for the sentence shown in figure 3.2 is therefore:

Relation	Path	Classification
(SpoIIAA-P,SpoIIE)	“COMP_of-1, COMP_by”	TtoA

With the obvious shortest path travelling through the node representing Dephosphorylation.

3.2.4 Morpho-Syntactic (POS) Categories

A set of attributes which was simple to extract, but proved highly useful was composed of the morpho-syntactic or Part of Speech (POS) categories of all words separating two entities. The POS categories were provided with the training and testing data; however they were not provided for every word, but only for the words participating in a linguistic relation (see section 3.2.3). This meant that there was a limited set of categories⁵, and that many words were not assigned to categories at all. Simple ad-hoc testing however showed that the lack of specificity in the categories was actually advantageous as it stopped the attribute lists from over-fitting the training data. The unassigned words that should have fallen into the categories used were sparse enough that the error introduced by using a part-of-speech tagger (as opposed to the manually corrected data given by the challenge organizers) appeared to be unwarranted.

To further reduce over-fitting of the training data, recurring categories of the same type were amalgamated into a single attribute. For instance, if the data contained several nouns in succession, only one noun attribute would be entered into the appropriate position in the list. Thus the attribute list:

Relation	Length	POS	Classification
(sigmaK,cwIH)	12	“N,ADJ,ADJ,N,N,N,V,V,V,N”	NEG

would be modified to:

Relation	Length	POS	Classification
(sigmaK,cwIH)	12	“N,ADJ,N,V,N”	NEG

⁵in some cases words which did not fit into any of the categories were placed a more “generalized” category. e.g. pronouns were often put into the noun category.

3.2.5 Weighted Voting

Initial tests with the various classifiers were promising; however, when attempting to decide on which classifier was best, it became apparent that each classifier had its benefits and its limitations. While some were very precise but lacked recall, others had high recall but would choose one alternative over another due to only a fraction of a percentage difference in their classification probabilities.

A second problem was that several of the classifiers appeared to have systematic difficulties in their classification schemes. Several classifiers weighed too heavily on the larger number of negative instances provided in the training data, and as a result were constantly giving inflated probabilities for negative classifications. Others over-fit the data so that they would provide very high probability answers in instances very similar to examples they had been trained on, but provided almost no distinction between classifications for instances that were too dissimilar.

All classifiers in WEKA return a list of probabilities for each attribute list in the testing set. One probability is associated with each classification which denotes the likelihood of that classification being correct. For example, if for a particular pair of entities, a classifier returned $P(\text{AtoT}) = 90\%$ $P(\text{TtoA}) = 3\%$ $P(\text{NEG})=7\%$, then the classifier has strongly identified that these particular entities have a relation, and that the first entity is the agent, while the second is the target. However if the classifier returned $P(\text{AtoT}) = 47\%$ $P(\text{TtoA})=46\%$ $P(\text{NEG})=7\%$, then once again the classifier has a strong level of confidence in the fact that there is a relation; however it has been unable to identify to any degree of accuracy in what direction that relation may lie. In this case, it will return the same result (i.e. the first entity is the agent, the second is the target), but we cannot be nearly as confident in these results.

The obvious answer to this problem was to train multiple classifiers [Larkey and Croft, 1996], and use all of them to classify the testing data, using the average of their predictions as a final decision on the existence and direction of relations. In doing this we increased the confidence of the classifications, and also guarded against the possibility of systematic weaknesses in any one classifier hurting the final results of the system. A simple perl script was designed to read in the output of a number of different classifiers, and report the average score⁶ for each classification. The classification with the

⁶An admittedly ad-hoc adjustment was made at this point, as the joint score for the negative classification was squared. This was done as a result of initial low recall scores, and later proved to increase the recall score without having a noticeable detrimental effect on precision.

highest probability at this point was taken as the final result, and the output formatted accordingly.

3.2.6 Classifiers Used

IBk Classifier

This classifier is based on the k-nearest neighbour algorithm [Aha et al., 1991] and uses a vector space model to determine the distance between an entity pair and the the k closest entity pairs of a given classification.

This algorithm first computes the distance between a test attribute list and all training attribute lists in the vector space using Equation 3.2. Once all pairwise distances have been computed, a “nearest neighbour” distance is computed based on the k attribute lists with minimum distance to the test attribute list as in Equation 3.1. This distance is computed once for each classification, the classification producing the lowest distance is returned as the most likely classification for that test example.

$$f(l_{new}) = \sum_{i=1}^k \frac{1}{w(l_{new}, l_i)} \quad (3.1)$$

$$w(l_a, l_b) = \sqrt{\sum_{j=1}^T (t_{ja} - t_{jb})^2} \quad (3.2)$$

Symbol Definitions

$w(l_a, l_b)$	Distance between list a and b
l_a, l_b	Attribute lists
T	Number of attributes in lists
t_{ja}	Weight of term j in list a
$f(d_{new})$	k nearest neighbours distance from d_{new}

The IBk classifier uses what is known as a “memory based” or “lazy” learning method because it defers processing of any training data until it is attempting to classify a test example.

Decision Table Classifier

Decision Table Classification is generally quite simple to implement, but can be very powerful in certain domains[Kohavi, 1995]. Decision table classification is perhaps best understood through example. Consider the simplified version of the classification problem in this paper where there is only a single entry for POS category (perhaps for the first word in between the entities) and there are at most three words between the entities. A decision table could then be constructed as seen in Table 3.3⁷.

Table 3.3: A Simple Decision Table

Length	POS	Classification
1	V	NEG
1	V-Pass	AtoT
1	N	AtoT
1	ADJ	NEG
1	ADV	TtoA
2	V	AtoT
2	V-Pass	NEG
2	N	NEG
2	ADJ	NEG
2	ADV	TtoA
3	V	NEG
3	V-Pass	TtoA
3	N	NEG
3	ADJ	NEG
3	ADV	NEG

In this table *Length* represents the number of words between the entities and *POS* represents the POS category (Verb, Passive Verb, Noun, Adjective and Adverb respectively). From this table we can now easily evaluate a given test example. For instance if we came across an example that had a length of 2, and a POS category of adjective, reading from the table would give us a result of “NEG” or no relation.

In more complicated examples it is possible that not all combinations would have been seen in the training data, or that several classifications could be seen for the same series of attributes. In instances such as these, the decision table classifier will return the most probable results based on the available

⁷The classification data here is given for demonstration purposes only and bears no relevance to the actual LLL05 challenge data.

data. For example, let us assume that in the previous situation, we had never encountered an instance with length 2 and POS V, and furthermore we had encountered four instances with length 1 and POS V-Pass, three of which were classified as NEG and one of which was classified as AtoT. In this situation we would produce the decision table seen in Table 3.4.

Table 3.4: A Decision Table Based on Imperfect Training Data

Length	POS	Classification
1	V	NEG)
1	V-Pass	NEG(75%), AtoT(25%)
1	N	AtoT
1	ADJ	NEG
1	ADV	TtoA
2	V	?
2	V-Pass	NEG
2	N	NEG
2	ADJ	NEG
2	ADV	TtoA
3	V	NEG
3	V-Pass	TtoA
3	N	NEG
3	ADJ	NEG
3	ADV	NEG

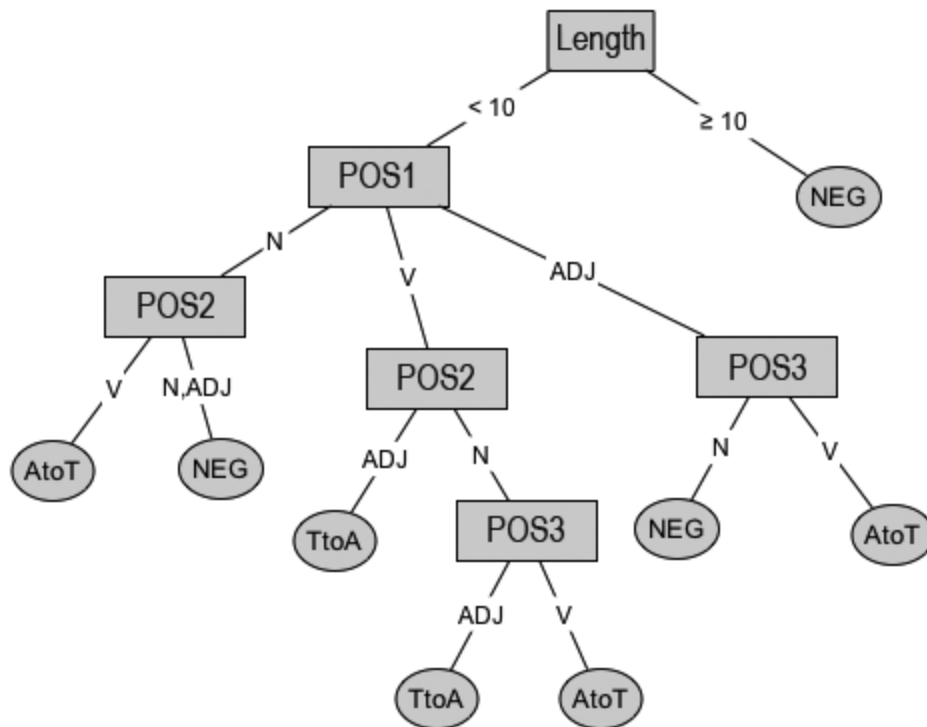
If after the production of Table 3.4, we were to come across an instance in the testing data where two entities had one word in between them, and the lexical category was a passive verb, we would report a result of NEG, but with decreased confidence (due to the NEG classification only appearing in 75% of training instances rather than the desired 100%). Furthermore if we were to encounter an instance with a length of 2 and a verb for its POS attribute, we would take the most likely answer based on the available situation. In this case, instances with length=2 are classified as NEG 75% of the time, and instances with a V under POS are classified as a NEG in all other instances, thus we would return a value of NEG, but again with less confidence than if we had seen an example that directly fit this particular instance. In some cases, decision table classifiers may have to combine these probabilistic estimation techniques to produce an answer.

Decision Stump Classifier

A decision stump is a form of decision tree with only one test at the root [Russell and Norvig, 2003]. Decision tree learning attempts to induce a tree from the training data, which can be followed to make decisions regarding the testing data.

As an example, let us consider another simplified version of the classification problem given in this paper, where we will only consider three possible POS categories (N, V, ADJ), and also only consider a maximum of three POS categories. One possible decision tree for this scenario can be seen in Figure 3.3.

Figure 3.3: Example Decision Tree



Once a decision tree has been created based on the training data, classification of the test data is very simple. If in our example we encounter an instance in the testing data with length=5, POS1=V, POS2=N, and POS3=V we can immediately classify this instance as type “AtoT” (i.e. the first entity is the agent, the second entity is the target).

In decision stump classification, the tree created only has one decision node

(here represented by the rectangular nodes) leading to two or more classification nodes (here represented by the elliptical nodes). Furthermore in both decision tree and decision stump learning algorithms, classification nodes do not need to represent a single classification, they can instead represent a probability distribution of classifications.

Naive Bayesian Classifier

Bayesian classifiers are based on Bayes' Rule which is shown in Equation 3.3

$$P(b|a) = \frac{P(a|b)P(b)}{P(a)} \quad (3.3)$$

Where $P(a|b)$ means "The Probability of a being true, assuming b to be true".

The naive bayes classifier assumes that all attributes are conditionally independent, and the only dependency relations that are taken into account are the relations between individual attributes and the classification. For instance, the naive bayesian classifier assumes that knowing the first POS category attribute does not give us any additional information in guessing what the second POS category might be. While this is not necessarily true, it is often the case that the probability relations between attributes are not as important to obtaining a correct classification as the probability relations between those attributes and the classification. Furthermore, this assumption allows us to use the very simple formula shown in Equation 3.4.

$$P(Class, Att_1, \dots, Att_n) = P(Class) \prod_i P(Att_i|Class) \quad (3.4)$$

This equation allows us to compute the most likely classification given a set of attributes by simply computing $P(Class)$ (number of instances of a given classifier in the training data, divided by all instances in the training data) and $P(Att_i|Class)$ (the number of times this particular value of the i^{th} attribute was seen in instances of this classification divided by the total number of instances of this classification). Both of these can be easily calculated from the training data.

Chapter 4

Results

4.1 Amilcare Results

Before reporting the scores obtained by the Amilcare based system, it must be noted that the system which produced these scores did have manual assistance in one aspect. In several places throughout the output, the XML tags inserted by Amilcare were moved to correctly surround the named entities identified. For example, in one instance, the tagged sentence:

These studies indicate that <target>spo0H</target>is transcribed by the major vegetative RNA polymerase, <agent>E sigma</agent>A.

was manually corrected by moving the closing agent tag as follows:

These studies indicate that <target>spo0H</target>is transcribed by the major vegetative RNA polymerase, <agent>E sigma A</agent>.

The necessity of manual correction is most likely linked to a failure in the gazetteer. Amilcare's default gazetteer had been disabled in the version we had obtained, and for this reason we were forced to create an alternative solution. We added tags in both the training and testing data to delimit dictionary words, and configured Amilcare to recognize these tags and add them as constraints in its rule generation process. Thus amilcare was still able to distinguish dictionary words, but was unable to consolidate multiple

word dictionary entries into single references (as it would have, had the gazetteer been functioning). We believe that is likely to be the primary cause of this problem.

These changes were only made when the tags were misplaced by no more than two characters, and in most cases one of the (opening or closing) tags was correctly placed, while the other was slightly misplaced. Furthermore in approximately 7 instances throughout the testing data, tags were placed where they did not surround a dictionary word at all. In these instances, the tags were ignored, as they would not have been accepted by the evaluation program. This adjustment was necessary to compensate for the failure of the gazetteer to provide a reasonable comparison of this system to other systems which used the provided dictionary. Despite the fact that this particular procedure was easily accomplished by a small script which simply deletes any tags not surrounding a dictionary word, it was admittedly ad-hoc, and thus should be reported at this stage.

The scores for the Amilcare based system are reported in the following tables, and the scores of competing systems in the LLL05 challenge are given for comparison.

Table 4.1: Amilcare Results without Coreference Only

Group #	Pre	Rec	F
1	50.0	53.8	51.8
2	22.2	11.1	14.8
4	37.9	55.5	45.1
5	25.0	81.4	38.2
6	60.9	46.2	52.6
Amilcare	28.8	27.7	28.3

Table 4.2: Amilcare Results with Coreference Only

Group #	Pre	Rec	F
5	14.0	93.1	24.4
Amilcare	25.5	37.9	30.5

Table 4.1 shows the score as evaluated only on instances not using coreference, Table 4.2 shows the scores as evaluated only on instances using coreference, and Table 4.3 shows the scores as evaluated on all instances in the training data.

Table 4.3: Amilcare Results with and without Coreferences

Group #	Pre	Rec	F
3	51.8	16.8	25.4
6	55.6	53.0	54.3
Amilcare	35.1	31.3	33.1

One thing to be noted is that the score reported in Table 4.1 is lower than the score reported in Table 4.2 (i.e. the system performed better on the “harder” coreferenced data). While this is most likely attributable to a simple statistical anomaly, it is interesting that this particular system had no more difficulty with the “harder” coreferenced data than with the “easier” non coreferenced data. This is one of the chief advantages of learning rules separately for the agent and target entities, separating the rules means that coreferencing does not increase the difficulty of the task, and thus this system performs equally well on both coreferenced and non coreferenced data.

The overall scores for the Amilcare based system, while not in contention with the higher scoring systems in the competition, were higher than several of the systems which performed much deeper linguistic processing of the data. In particular the performance on the coreferenced data was much higher than would have been expected for a system based on an algorithm designed primarily for entity recognition.

4.2 Pairwise Classification Results

The following tables provide the results for the pairwise classification algorithm using the official LLL05 evaluation program. Scores for three different variants of the algorithm are given. *PC:POS* refers to the case in which the attribute lists were composed of only the POS categories of the words between the entities in the data, *PC:Ling-Rel* refers to the case in which the attribute list is composed of only shortest path through the graph defined by the linguistic relations connecting the entities and *PC:POS & Ling-Rel* refers to the case in which the attribute lists were the union of the lists defined by the other two cases. All cases also included a numerical attribute for the number of words separating the entities in the sentence.

The scores for the individual classifiers on the full data set are given in Table 4.4.

Table 4.4: Individual Classifier Results

Classifier	PC:POS			PC:Ling-Rel			PC:POS & Ling-Rel		
	Pre	Rec	F	Pre	Rec	F	Pre	Rec	F
Bayesian	36.3	66.6	47.0	28.2	44.5	34.5	34.0	40.9	37.1
IBk	22.5	21.6	22.0	48.2	65.0	55.3	42.8	36.1	39.2
D. Table	15.7	38.5	22.3	30.4	38.5	34.0	35.7	36.1	35.9
D. Stump	22.6	31.3	26.2	21.4	38.5	27.5	38.9	55.4	45.7

The following tables are based on the weighted voting of all four classifiers as described in section 3.2.5. Scores for competing systems in the LLL05 challenge and for the Amilcare based system are included here for comparison.

Table 4.5: Pairwise Classification Results without Coreference Only

Group #	Pre	Rec	F
1	50.0	53.8	51.8
2	22.2	11.1	14.8
4	37.9	55.5	45.1
5	25.0	81.4	38.2
6	60.9	46.2	52.6
Amilcare	28.8	27.7	28.3
PC: POS	43.8	72.2	54.5
PC: Ling-Rel	44.5	68.5	54.0
PC: POS & Ling-Rel	43.4	61.1	50.7

Table 4.6: Pairwise Classification Results with Coreference Only

Group #	Pre	Rec	F
5	14.0	93.1	24.4
Amilcare	25.5	37.9	30.5
PC: POS	24.2	58.6	34.3
PC: Ling-Rel	30.6	65.5	41.7
PC: POS & Ling-Rel	27.4	48.2	35.0

Table 4.7: Pairwise Classification Results with and without Coreferences

Group #	Pre	Rec	F
3	51.8	16.8	25.4
6	55.6	53.0	54.3
Amilcare	35.1	31.3	33.1
PC: POS	44.0	67.4	53.3
PC: Ling-Rel	48.2	67.4	56.2
PC: POS & Ling-Rel	47.9	56.6	51.9

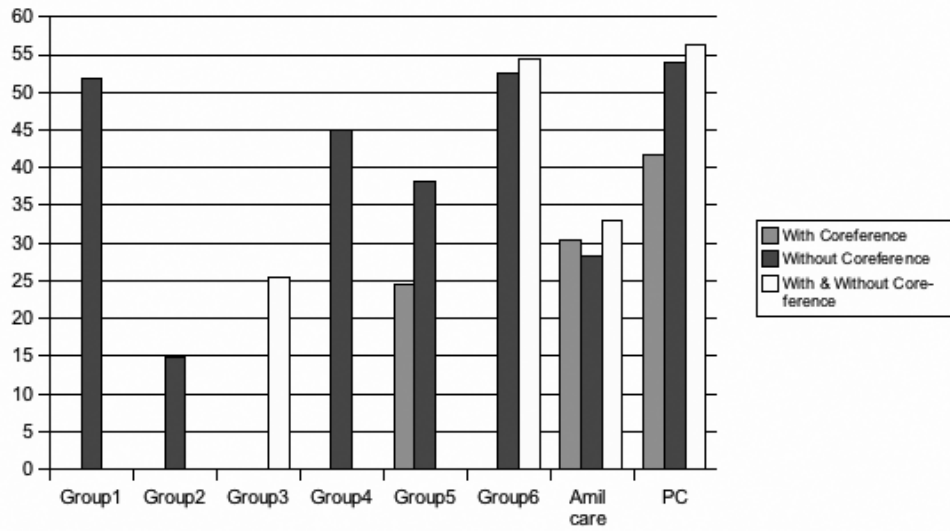
Table 4.5 shows the score as evaluated only on instances not using coreference, Table 4.6 shows the scores as evaluated only on instances using coreference, and Table 4.7 shows the scores as evaluated on all instances in the test data.

4.3 Comparison of Results

Figure 4.1 provides a summation of the results for the two systems described in this paper as well as for the LLL05 challenge participant systems. F-measure scores are given for all three categories where provided¹.

¹The F-measure score for the Pairwise Classification system (identified as PC in Figure 4.1) is for the linguistic relation based system identified as PC: Ling-Rel in Tables 4.5 through 4.7.

Figure 4.1: F-Measure vs. Category



One interesting feature to note is that some of the systems had a large difference in performance between the coreferenced and non coreferenced data (as seen in the scores of Group 5 and the Pairwise Classification System), while others had little or no difference (as seen in the Amilcare based system).

Chapter 5

Conclusions

The remarkable aspects of the systems presented in this paper are not the scores they achieved, but the relative simplicity of the information and algorithms that allowed them to achieve these competitive scores. The Amilcare based system performed competitively using very simple rule generation techniques originally intended for the simpler task of entity recognition, and the pairwise classification system performed even more competitively using nothing but POS tags and distance separation of entities.

The Amilcare based system used very simplistic techniques to develop basic rules for the identification of entities and managed to forgo any complex rule generation. The $(LP)^2$ algorithm was not designed for relation extraction, but for the much simpler task of entity recognition. However we managed to use this “simple” algorithm to achieve scores which were only surpassed by the very top systems in the LLL05 Genic Interaction Extraction Challenge.

The Pairwise Classification POS based system was even more remarkable in that it used only the number of words separating entities in a sentence and their POS categories to achieve scores which surpassed any system in the challenge in all but one category. This system did not use any complex tools or computationally expensive algorithms, and the classifiers used were unaltered publicly available tools with no “tweaking” to this particular task.

The Pairwise Classification Linguistic Relation based system was by far the best scoring system overall. The POS based system had a slightly higher score on the simplest data set, but the linguistic relation based system surpassed all others when it came to extracting relations from data which contained coreferences. This algorithm was also relatively simple computationally, and used the same publicly available classification tools as the POS based system.

The systems developed for the LLL05 Genic Interaction Extraction Challenge used state of the art algorithms and (in many cases) domain specific tools and techniques to achieve their scores. It is certainly interesting that an algorithm as simple and general as pairwise classification was able to achieve competitive scores in this challenge. Furthermore the algorithms presented in this paper do not use any tools which were designed specifically for this domain. This should increase our confidence that these techniques will be easily portable to new domains.

The problem of extracting relations from unstructured text is far from solved. The scores presented in this thesis and in the LLL05 challenge are promising, and indicate that significant developments are being made in this field; however, with the top scores only reaching into the 50-60% range, there is still a great deal more research to be done in this area. The results presented in this thesis indicate that future research could benefit from examining simplistic rule generation, and classification techniques; these models are simple and easy to apply to any domain, and we have shown that building upon these basic models can produce scores which are competitive with any other systems currently available.

5.1 Links to Coursework

The Computing Laboratory at Oxford University offers very enriching courses on a wide variety of subjects. Three of these courses in particular directly relate to this project, and I certainly would have been unable to complete this task satisfactorily if it were not for the information gained in these courses.

5.1.1 Intelligent Systems

This course covered the machine learning and classification techniques which were used extensively in this project. In particular without this course I would not have had sufficient knowledge to understand the workings of the various classifiers, and would have been at a loss as to how to best utilize them.

5.1.2 Information Retrieval

Relation extraction in many senses is a subset of information retrieval. Many of the core concepts of information extraction and retrieval were covered in this course. In particular aspects of the project such as the balance of precision and recall would have been much more difficult without the knowledge gained in this course.

5.1.3 Computational Linguistics

Without this course, I would not have known what morpho-syntactic categories, part-of-speech tags, or lexical relations were, much less how to utilize them to the benefit of the system. This course covered material very close to the core of this project, and without it there is little doubt that all aspects of this project would have been much more difficult.

Bibliography

- David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.
- Amilcare Website. Amilcare: Adaptive IE tool. <http://nlp.shef.ac.uk/amilcare/>, July 2005.
- Sophie Aubin. Challenge LLL syntactic analysis guidelines. In *Learning Language in Logic (LLL05) Challenge*, March 2005.
- J. Bos. Towards wide-coverage semantic interpretation. In *Proceedings of IWCS-6*, Tillburg, The Netherlands, 2005.
- Nancy A. Chinchor. Overview of MUC-7/MET-2. In *proceedings of the seventh Machine Understanding Conference*, April 1998.
- Fabio Ciravegna. Adaptive information extraction from text by rule induction and generalisation. In *17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, August 2001.
- Fabio Ciravegna. $(LP)^2$, rule induction for information extraction using linguistic constraints. Technical Report CS-03-07, University of Sheffield, September 2003.
- Stephen Clark and James R. Curran. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL '04)*, pages 104–111, Barcelona, Spain, 2004.
- William W. Cohen. Fast effective rule induction. In Armand Prieditis and Stuart Russell, editors, *Proc. of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, July 9–12, 1995. Morgan Kaufmann. ISBN 1-55860-377-8.
- M. Craven and J. Kumlien. Constructing biological knowledge bases by extracting information from text sources. In *Proceedings of the Fourth International Conference on Computational Linguistics and Intelligent Text Processing*, pages 77–86, Heidelberg, Germany, 1999. AAAI Press.

- M. Goadrich, L. Oliphant, and J. Shavlik. Learning ensembles of first-order clauses for recall-precision curves: A case study in biomedical information extraction. In *Proceedings of the 14th International Conference on Inductive Logic Programming (ILP)*, Porto, Portugal, 2004.
- Mark Goadrich, Louis Oliphant, and Jude Shavlik. Learning to extract genic interactions using gleaner. In *Proceedings of the Learning Language in Logic (LLL05) workshop*, August 2005.
- Mark A. Greenwood, Mark Stevenson, Yikun Guo, Henk Harkema, and Angus Roberts. Automatically acquiring a linguistically motivated genic interaction extraction system. In *Proceedings of the Learning Language in Logic (LLL05) workshop*, August 2005.
- Jörg Hakenberg, Conrad Plake, Ulf Lesser, Harald Kirsch, and Dietrich Rebholz-Schuhmann. LLL'05 challenge: Genic interaction extraction - identification of language patterns based on alignment of finite state automata. In *Proceedings of the Learning Language in Logic (LLL05) workshop*, August 2005.
- Jos Iria and Fabio Ciravegna. Relation extraction for mining the semantic web. In *Proceedings of the Dagstuhl Seminar on Machine Learning for the Semantic Web*, February 2005.
- Sophia Katrenko, M. Scott Marshall, Marco Roos, and Pieter Adriaans. Learning biological interactions from medline abstracts. In *Proceedings of the Learning Language in Logic (LLL05) workshop*, August 2005.
- Ron Kohavi. The power of decision tables. In Nada Lavrac and Stefan Wrobel, editors, *Proceedings of the European Conference on Machine Learning*, Lecture Notes in Artificial Intelligence 914, pages 174–189, Berlin, Heidelberg, New York, 1995. Springer Verlag.
- Leah S. Larkey and W. Bruce Croft. Combining classifiers in text categorization. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 289–297, New York, NY, USA, 1996. ACM Press. ISBN 0-89791-792-8.
- LLL05 Challenge Website. LLL05 genic interaction extraction challenge. <http://genome.jouy.inra.fr/texte/LLLchallenge/>, July 2005.
- E. Loper. Applying semantic relation extraction to information retrieval, 2000. Master's thesis, Massachusetts Institute of Technology. 56.
- E. Marsh and D. Perzanowski. MUC-7 evaluation of IE technology: Overview of results. In *proceedings of the seventh Machine Understanding Conference*, April 1998.

- Diana Maynard, Valentin Tablan, Hamish Cunningham, Cristian Ursu, Horacio Saggion, Kalina Bontcheva, and Yorick Wilks. Architectural elements of language engineering robustness. *Nat. Lang. Eng.*, 8(3):257–274, 2002. ISSN 1351-3249.
- Ryan McDonald, Fernando Pereira, Seth Kulick, Scott Winters, Yang Jin, and Pete White. Simple algorithms for complex relation extraction with applications to biomedical IE. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 491–498, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.
- Claire Nédellec. Learning language in logic - genic interaction extraction challenge. In *Proceedings of the Learning Language in Logic (LLL05) workshop*, August 2005.
- L. Popelínský, J. Blažák, and M. Nepil. Feature construction with RAP. In *ILP'03 Works in Progress*, pages 1–10, 2003.
- Luboš Popelínský and Jan Blažák. Learning genic interactions without expert domain knowledge: Comparison of different ILP algorithms. In *Proceedings of the Learning Language in Logic (LLL05) workshop*, August 2005.
- Sebastian Riedel and Ewan Klein. Genic interaction extraction with semantic and syntactic chains. In *Proceedings of the Learning Language in Logic (LLL05) workshop*, August 2005.
- Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.
- SAIC IE Website. SAIC information extraction website. http://www.itl.nist.gov/iaui/894.02/related_projects/muc/index.html, July 2005.
- D. Sleator and D. Temperley. Parsing english with a link grammar. In *Third International Workshop on Parsing Technologies*, Tilburg, Netherlands, August 1993.
- Ashwin Srinivasan. *The Aleph Manual*, 2003. URL <http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.html>.
- Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.

Appendix A

Code

A.1 Amilcare System Code

A.1.1 Training Data Formatting Script

```
#!/usr/local/bin/perl

#####
#Training Data Formatting Script #
#####
#Brian Harrington - June 2005 #
#####
#This script formats the basic training data set available at: #
#http://genome.jouy.inra.fr/texte/LLlchallenge/ into an XML markup text that can#
#be used as a training corpus for Amilcare #
#####

#####
#Define I/O files to be used #
#####
#The basic training data from the challenge
$infile = "./basic_training_data.txt";
#the output XML formatted training data
$outfile = "./amilcare_training_data.txt";
#The dictionary of named entities
$dictfile = "./dictionary.txt";

#####
#Create The Dictionary #
#####
open(DICTFILE, "<$dictfile") or die "Cannot open $dictfile";
#For each line of the dictionary file, parse out each typographical variant of
#the entity, and create an entry in an associative array for that variant which
#points to the canonical form of the word. This allows us to easily retrieve
#the canonical form of any dictionary word
while($dictLine = <DICTFILE>){
    #retrieve canonical form of word
```

```

($canonical,$others) = $dictLine =~ /(.*?)[\n\t](.*)/;
#create an entry for the canonical form pointing to itself
$dictionary{$canonical} = $canonical;
#for each variant
while($others ne ""){
    $hold = $others;
    #retrieve the variant
    ($next,$others) = $others =~ /(.*?)[\n\t](.*)/;
    #create the entry for this variant
    if($next ne ""){
        $dictionary{$next} = $canonical;
    }else{
        #this is only performed for the final variant in the list
        $dictionary{$hold} = $canonical;
    }
}
}
close(DICTFILE);

open(INFILE, "< $infile") or die "cannot open file $infile";
open(OUTFILE, "> $outfile") or die "cannot open file $outfile";
#For each line of input
while($IDline = <INFILE>){
    unless($IDline =~ /^ID(\s+)(.*)/){
        next;
    }
    #The line of text
    $sentline = <INFILE>;
    #The line with the numbers of each words
    $wordline = <INFILE>;
    #The line identifying the agents
    $agentline = <INFILE>;
    #The line identifying the targets
    $targetline = <INFILE>;
    #The line identifying the interactions
    $interactionline = <INFILE>;
    #A blank line
    $dummyline = <INFILE>;

    #split the Information in each line from their labels
    ($dummy,$sentence) = $sentline =~ /^sentence(\s+)(.*)/;
    ($dummy,$words) = $wordline =~ /^words(\s+)(.*)/;
    ($dummy,$interactions) = $interactionline =~ /^genic_interactions(\s+)(.*)/;

    #####
    #For each interaction, record the position of the target and agent in #
    #the sentence #
    #####

    %tags=%dummy;
    #For each interaction
    while($interactions ne ""){
        #Parse out the agent and target word numbers
        ($agentNum,$targetNum,$interactions) = $interactions =~
            /^genic_interaction\((\d+),(\d+)\)(.*)/;
        ($dummy,$interactions) = $interactions =~ /^(.+)(.*)/;
        #Keep a list of targets & agents
        $tags{$agentNum} = "<agent>";
        $tags{$targetNum} = "<target>";
    }
    $lastPrinted = 0;
}

```

```

#####
#For each word, if it is a dictionary word, translate it into canonical #
#form, and if it should be surrounded by XML tags, then insert the tags #
#####

@wordSpecArray = split(/\t/, $words);
foreach $wordSpec (@wordSpecArray){
    #Parse out the word number, start and end position
    ($wordNum, $thisWord, $wordStart, $wordEnd) = $wordSpec =~
        /^word\((\d+), '(.*?)', (\d+), (\d+)\)/;

    #All the punctuation before the word
    $outString = substr($sentence, $lastPrinted, $wordStart - $lastPrinted);
    #The word itself
    $thisWord = substr($sentence, $wordStart, ($wordEnd - $wordStart) + 1);
    #Translate to canonical form and mark with XML if this word is a
    #dictionary word
    if(exists $dictionary{$thisWord}){
        $thisWord = "<gene>$dictionary{$thisWord}</gene>";
    }
    #if this word is a target or an agent, also put the appropriate
    #XML tags surrounding it
    if ($tags{$wordNum} eq "<agent>"){
        $outString = $outString. "<agent>".$thisWord."</agent>";
    }elseif($tags{$wordNum} eq "<target>"){
        $outString = $outString. "<target>".$thisWord."</target>";
    }else{
        $outString = $outString . $thisWord;
    }
    $lastPrinted = $wordEnd + 1;
    #print the word plus all of the punctuation and XML tags
    print OUTFILE $outString;
}
#print any punctuation at the end of the sentence
$outString = substr($sentence, $lastPrinted, length($sentence) - $lastPrinted);
print OUTFILE $outString;
print OUTFILE "\n";

}
#Close all open files
close(INFILE);
close(OUTFILE);

```

A.1.2 Test Data Formatting Script

```
#!/usr/local/bin/perl

#####
#Test Data Formatting Script #
#####
#Brian Harrington - June 2005 #
#####
#This script formats the basic test data set available at: #
#http://genome.jouy.inra.fr/texte/LLLchallenge/ into an XML markup text that can#
#be used as a test corpus by Amilcare #
#####

#The challenge test corpus
$infile = "./basic_test_data.txt";
#The output to be used as a test corpus by Amilcare
$outfile = "./amilcare_test_data.txt";
#The dictionary of named entities
$dictfile = "./dictionary.txt";
#The list of all IDs for use in reconstituting data for evaluation
$idfile = "./id_list.txt";

#####
#Create The Dictionary #
#####
open(DICTFILE, "<$dictfile") or die "Cannot open $dictfile";
#For each line of the dictionary file, parse out each typographical variant of
#the entity, and create an entry in an associative array for that variant which
#points to the canonical form of the word. This allows us to easily retrieve
#the canonical form of any dictionary word
while($dictLine = <DICTFILE>){
    #retrieve canonical form of word
    ($canonical,$others) = $dictLine =~ /(.*?)[\n\t](.*)/;
    #create an entry for the canonical form pointing to itself
    $dictionary{$canonical} = $canonical;
    #for each variant
    while($others ne ""){
        $hold = $others;
        #retrieve the variant
        ($next,$others) = $others =~ /(.*?)[\n\t](.*)/;
        #create the entry for this variant
        if($next ne ""){
            $dictionary{$next} = $canonical;
        }else{
            #this is only performed for the final variant in the list
            $dictionary{$hold} = $canonical;
        }
    }
}
close(DICTFILE);

#Open the appropriate files
open(INFILE, "< $infile") or die "cannot open file $infile";
open(OUTFILE, "> $outfile") or die "cannot open file $outfile";
open(IDFILE, ">$idfile") or die "cannot open file $idfile";
while($IDline = <INFILE>){
    #If this is not a proper line of text (e.g. it is a comment line) ignore it
    unless($IDline =~ /^ID(\s+)(.*)/){
        next;
    }
}
```

```

#output the ID of this sentence to the appropriate file
print IDFILE $IDline;
#The line of text
$sentline = <INFILE>;
#The line with the numbers of each words
$wordline = <INFILE>;
#A blank line
$dummyline = <INFILE>;

#split the Information in each line from their labels
($dummy,$sentence) = $sentline =~ /^sentence\s+(.*)/;
($dummy,$words) = $wordline =~ /^words\s+(.*)/;

#####
#For each word, if it is a dictionary word, translate it into caninical #
#form, and if it should be surrounded by XML tags, then insert the tags #
#####
@wordSpecArray = split(/\t/, $words);
foreach $wordSpec (@wordSpecArray){
    #Parse out the word number, start and end position
    ($wordNum,$thisWord,$wordStart,$wordEnd) = $wordSpec =~
        /^word\((\d+),'(.*?)',(\d+),(\d+)\)/;
    #All the punctuation before the word
    $outString = substr($sentence,$lastPrinted,$wordStart - $lastPrinted);
    #The word itself
    $thisWord = substr($sentence,$wordStart,($wordEnd - $wordStart) + 1);
    #Translate to canonical form and mark with XML if this word is a
    #dictionary word
    if(exists $dictionary{$thisWord}){
        $thisWord = "<gene>$dictionary{$thisWord}</gene>";
    }
    $outString = $outString . $thisWord;
    $lastPrinted = $wordEnd + 1;
    #print the word plus all of the punctuation and XML tags
    print OUTFILE $outString;
}
#print any punctuation at the end of the sentence
$outString = substr($sentence,$lastPrinted,length($sentence) - $lastPrinted);
print OUTFILE $outString;
print OUTFILE "\n";
}
#Close all open files
close(IDFILE)
close(INFILE);
close(OUTFILE);

```

A.1.3 Result Formatting Script

```
#!/usr/local/bin/perl

#####
#Results Formatting Script #
#####
#Brian Harrington - June 2005 #
#####
#This script takes XML formatted text, where agents and targets are identified #
#by appropriate XML tags, and produces output suitable for the LLL05 scoring #
#services, which can be found at #
#http://genome.jouy.inra.fr/texte/LLLchallenge/scoringService.php #
#####

#####
#Define I/O files to be used #
#####
#list of IDs for each line
$ID_file = "./id_list.txt";
#XML delimited text
$amilcare_file = "./amilcare_output.txt";
#results file to be used for evaluation
$output_file = "./amilcare_results.txt";

open(IDFILE, "< $ID_file") or die "Cannot open file $ID_file";
open(AMILFILE, "< $amilcare_file") or die "Cannot open file $amilcare_file";
open(OUTFILE, "> $output_file") or die "Cannot open file $output_file";

#Print the appropriate header, the coreference distinction can be changed
#based on whether or not we are testing with coreference.
print OUTFILE "% Participant name: Brian Harrington\n";
print OUTFILE "% Participant institution: Oxford University\n";
print OUTFILE "% Participant email address: brian.harrington@stx.ox.ac.uk\n";
print OUTFILE "% Format checked: YES\n";
print OUTFILE "% Basic data: YES\n";
print OUTFILE "% Coreference distinction: WITHOUT COREFERENCE\n";

#For each line of text
while($id_line = <IDFILE>){
    $thisline = <AMILFILE>;

    #reset all arrays
    @agents = @blank;
    @targets = @blank;
    #count of Agents in this sentence
    $agentsCount = 0;
    #count of Targets in this sentence
    $targetsCount = 0;

    #####
    #Count and store all the agents in this sentence #
    #####
    while($thisline ne ""){
        #Get the next tag in this sentence
        ($dummy,$thistag,$thisline) = $thisline =~ /(.*?)(<\w+>)(.*)/;
        #if the tag is empty, there are no more tags so don't print anything
        if($thistag eq ""){
            print "";
        }
        #if this is just a gene tag, ignore it and move on
    }
}
```

```

    elif($thistag eq "<gene>"){
        print "";
        #if this is an agent tag, keep the agent, and increase the count of
        #agents seen so far
    }elseif($thistag eq "<agent>"){
        ($thisAgent,$thisline) = $thisline =~ /^(>+)<[>]+>(.*)/;
        $agents[$agentsCount] = $thisAgent;
        $agentsCount +=1;
        #if this is a target tag, keep the target, and increase the count of
        #targets seen so far
    }elseif($thistag eq "<target>"){
        ($thisTarget,$thisline) = $thisline =~ /^(>+)<[>]+>(.*)/;
        $targets[$targetsCount] = $thisTarget;
        $targetsCount +=1;
    }else{
        #if we reach this point, there has been an error, output an error
        #message to the screen and quit
        print "ERROR $thistag is not a proper tag!";
        exit(1);
    }
}

#####
#Create the output for this sentence (if there is any to be output #
#####

#If we've seen at least 1 agent and 1 target, create an output line for
#this sentence
if($agentsCount >0 && $targetsCount >0){
    print OUTFILE$id_line;
    print OUTFILE "agents ";
    #Print all the agents
    for($i = 0; $i<$agentsCount;$i++){
        print OUTFILE "agent(\'$agents[$i]') ";
    }
    print OUTFILE "\n";
    print OUTFILE "targets ";
    #Print all the targets
    for($i = 0; $i<$targetsCount;$i++){
        print OUTFILE "target(\'$targets[$i]') ";
    }
    print OUTFILE "\n";
    print OUTFILE "genic_interactions ";
    #Print all agent/target combinations
    for($i = 0; $i<$agentsCount;$i++){
        for($j = 0; $j<$targetsCount;$j++){
            print OUTFILE "genic_interaction(\'$agents[$i]\'\'\'$targets[$j]\'') ";
        }
    }
    print OUTFILE "\n";
}
}

#Close all open files
close(IDFILE);
close(AMILFILE);
close(OUTFILE);

```

A.2 Pairwise Classification Code

A.2.1 Training Data Formatting Script

```
#!/usr/local/bin/perl

#####
#Training Data Formatting Script #
#####
#Brian Harrington - July 2005 #
#####
#This script formats the enriched training data set available at: #
#http://genome.jouy.inra.fr/texte/LLLchallenge/ into an attribute list which can#
#be used by most WEKA classifiers. #
#####

#####
#Define I/O files to be used #
#####

#The enriched training data set
$infile = "./enriched_training_data_coref.txt";
#The training file to be used in WEKA
$outfile = "./brian_training_data.arff";
#The dictionary of named entities
$dictfile = "./dictionary.txt";

#####
#Create The Dictionary #
#####
open(DICTFILE, "<$dictfile") or die "Cannot open $dictfile";
#For each line of the dictionary file, parse out each typographical variant of
#the entity, and create an entry in an associative array for that variant which
#points to the canonical form of the word. This allows us to easily retrieve
#the canonical form of any dictionary word
while($dictLine = <DICTFILE>){
    #retrieve canonical form of word
    ($canonical,$others) = $dictLine =~ /(.*?)[\n\t](.*)/;
    #create an entry for the canonical form pointing to itself
    $dictionary{$canonical} = $canonical;
    #for each variant
    while($others ne ""){
        $hold = $others;
        #retrieve the variant
        ($next,$others) = $others =~ /(.*?)[\n\t](.*)/;
        #create the entry for this variant
        if($next ne ""){
            $dictionary{$next} = $canonical;
        }else{
            #this is only performed for the final variant in the list
            $dictionary{$hold} = $canonical;
        }
    }
}
close(DICTFILE);

#####
#Open Necessary Files & Create Headers #
#####
```

```

open(INFILE, "< $infile") or die "cannot open file $infile";
open(OUTFILE, "> $outfile") or die "cannot open file $outfile";

#Create the attribute list header, *NOTE* this list has been shortened to only
#include 5 POS (morpho syntactic category) attributes and 1 relation attribute
#to facilitate easier printing
print OUTFILE (<<EOH);
\@relation genic_interaction
\@attribute LENGTH real
\@attribute POS1 { N, V, ADV, ADJ, V_PASS, NONE}
\@attribute POS2 { N, V, ADV, ADJ, V_PASS, NONE}
\@attribute POS3 { N, V, ADV, ADJ, V_PASS, NONE}
\@attribute POS4 { N, V, ADV, ADJ, V_PASS, NONE}
\@attribute POS5 { N, V, ADV, ADJ, V_PASS, NONE}
\@attribute REL1 {subj, subj-1, comp_for, comp_for-1, comp_to, comp_to-1, comp_of,
  comp_of-1, comp_in, comp_in-1, comp_with, comp_with-1, comp_on, comp_on-1,
  comp_within, comp_within-1, comp_by, comp_by-1, comp_beyond, comp_beyond-1,
  comp_unlike, comp_unlike-1, comp_like, comp_like-1, comp_near, comp_near-1,
  comp_at, comp_at-1, comp_upon, comp_upon-1, comp_via, comp_via-1, comp_through,
  comp_through-1, comp_under, comp_under-1, comp_than, comp_than-1, comp_during,
  comp_during-1, comp_as, comp_as-1, comp_from, comp_from-1, mod, mod-1, mod_pred,
  mod_pred-1, mod_att, mod_att-1, comp_between, comp_between-1, mod_post, mod_post-1,
  obj, obj-1, appos, appos-1, neg, neg-1, NONE}
\@attribute CLASSIFICATION { AtoT, TtoA, NEG}
\data
EOH

#For each sentence
while($IDline = <INFILE>){
  #If this is not a proper sentence line (e.g. it is a comment line) ignore
  #it and move on
  unless($IDline =~ /^ID(\s+)(.*)/){
    next;
  }
  chomp($IDline);
  #The line of text
  $sentline = <INFILE>;
  #The line with the numbers of each words
  $wordline = <INFILE>;
  #The lemmatizations of all words
  $lemmaline = <INFILE>;
  #The syntactic relations
  $relationsline = <INFILE>;
  #The line identifying the agents
  $agentline = <INFILE>;
  #The line identifying the targets
  $targetline = <INFILE>;
  #The line identifying the relations
  $interactionline = <INFILE>;
  #An empty line
  $dummyline = <INFILE>;

  #split the Information in each line from their labels
  ($dummy, $sentence) = $sentline =~ /^sentence(\s+)(.*)/;
  ($dummy, $words) = $wordline =~ /^words(\s+)(.*)/;
  ($dummy, $interactions) = $interactionline =~ /^genic_interactions(\s+)(.*)/;
  ($relations) = $relationsline =~ /^syntactic_relations(.*)/;

  #####
  #Processing the relations line, extract relations between words and #
  #POS categories #

```

```

#####

#The count of relations printed thus far
$relCount = 0;
#reset the arrays
@POSList = @blank;
@relList = @blank;
@countRelArray1 = @blank;
@countRelArray2 = @blank;
@relArray1st = @blank;
@relArray2nd = @blank;
@relArray1stsize = @blank;
@relArray2ndsize = @blank;
@edge = @blank;

#For each relation
while($relations ne ""){
    #Parse up the relation text to extract the indicies of the words in the
    #relation and the type of relation
    ($dummy,$relText,$indexA,$indexB,$relations) = $relations =~
        /\s+relation\('[^']*'\s+(\d+)\s+(\d+)\s+(.*)/;
    #Split the relation type from the POS categories
    ($relType,$posA,$posB) = $relText =~ /^[^:]+:([^-]+)-(.*)/;
    #If its the appos case, just create the relType variable, if not, then
    #update the POS tags
    if($relType eq ""){
        $relType = $relText;
    }else{
        $POSList[$indexA] = $posA;
        $POSList[$indexB] = $posB;
    }
    #only take the main part of the relation (Not in use in final system)
    #if($relType =~ /^[^_]+_(.*)/){
    #    ($relType,$dummy) = $relType =~ /^[^_]+_(.*)/;
    #}
    #Create a 2-dimensional array to store for each word, an array of
    #relations in which it is the first element
    $relArray1st[$indexA][$relArray1stsize[$indexA]] = $relType;
    $relArray1stsize[$indexA] +=1;
    #Create a 2-dimensional array to store for each word, an array of
    #relations in which it is the second element
    $relArray2nd[$indexB][$relArray2ndsize[$indexB]] = $relType;
    $relArray2ndsize[$indexB] +=1;

    #create the graph with the words as nodes, and label the edges with
    #the type of relation connecting them.
    $edge[$indexA][$indexB] = $relType;
    $edge[$indexB][$indexA] = $relType;
}

#####
#Processing the words line, extrat the words, convert dictionary words to #
#canonical form, and insert them into an array to be printed later #
#####

$lastprinted = 0;
#Number of dictionary words printed so far
$geneCount = 0;
#Array of dictionary words encountered by word number
@genesArray = @blank;
#Array of dictionary words encountered by canonical form of word

```

```

@geneNameArray = @blank;
#Array of word specifications (word, start/end positions, word number)
@wordSpecArray = split(/\t/, $words);
#Number of dictionary words seen so far
$dicWordCount = 1;
#Number of words seen so far
$wordCount=0;
#For each word in the sentence
foreach $wordSpec (@wordSpecArray){
    $wordCount +=1;
    #parse out the word, its start/end position, and its word number
    ($wordNum, $thisWord, $wordStart, $wordEnd) = $wordSpec =~
        /^word\((\d+), '(.*?)', (\d+), (\d+)\)/;

    #the punctuation preceeding the word
    $outString = substr($sentence, $lastprinted, $wordStart - $lastprinted);
    #the actual word as it appears in the sentence, with punctuation
    $thisWord = substr($sentence, $wordStart, ($wordEnd - $wordStart) + 1);
    #If this is a dictionary word, translate it to canonical form, and
    #put it into an array to be dealt with later
    if(exists $dictionary{$thisWord}){
        $dicWordCount +=1;
        $genesArray[$geneCount] = $wordNum;
        $geneNameArray[$geneCount] = $dictionary{$thisWord};
        $geneCount +=1;
    }
}

#####
#Printing the attribute list for all positive instances from the #
#interactions line #
#####
%tags=%dummy;
@interactionsArray = @blank;
#For each positive interaction
while($interactions ne ""){
    #Extract the word number of the agent and target words
    ($agentNum, $targetNum, $interactions) = $interactions =~
        /^genic_interaction\((\d+), (\d+)\)(.*)/;
    ($dummy, $interactions) = $interactions =~ /^(s+)(.*)/;
    #set the appropriate value in the 2-dimensional array
    $interactionsArray[$agentNum][$targetNum] = 1;
    chomp($IDline);
    #decide the direction of the interaction, based on whether the agent
    #or the target comes first in the sentence
    if($agentNum < $targetNum){
        $larger = $targetNum;
        $smaller = $agentNum;
        $direction = "AtoT";
    }else{
        $larger = $agentNum;
        $smaller = $targetNum;
        $direction = "TtoA";
    }
}

#Print the number of words between the entities in the sentence
print OUTFILE ($larger - $smaller). ",";
#Print the POS categories between the entities
printbetween($genesArray[$i], $genesArray[$j]);
#Print the shortest path through the graph of lexical relations
breadthfirst($genesArray[$i], $genesArray[$j]);
#Print the classification (AtoT or TtoA)
print OUTFILE "$direction";

```

```

        print OUTFILE "\n";
    }
    $lastprinted = 0;

#####
#Printing the attribute list for all negative instances from the #
#remaining pairs of entities #
#####
#traverse the 2-dimensional array
for($i=0;$i<($geneCount-1);$i++){
    for($j = ($i+1);$j<$geneCount;$j++){
        #if this pair was in the positive interactions list, don't print
        #a negative attribute list
        if($interactionsArray[$genesArray[$i]][$genesArray[$j]] == 1
            || $interactionsArray[$genesArray[$j]][$genesArray[$i]] == 1){
            print OUTFILE "";
        }else{
            #Print the length attribute
            print OUTFILE ($genesArray[$j] - $genesArray[$i]).",";
            #Print the POS categories between the entities
            printbetween($genesArray[$i],$genesArray[$j]);
            #Print the shortest path through the graph of lexical relations
            breadthfirst($genesArray[$i],$genesArray[$j]);
            #Print NEG to indicate that this was a negative instance
            print OUTFILE "NEG";
            print OUTFILE "\n";
        }
    }
}
}
}
#Close all open files
close(INFILE);
close(OUTFILE);

#Print all the POS categories in the sentence from position
#$start to position $end
sub printbetween{
    my($start,$end) = @_;
    $lastPOS = "";
    $POSprintedCount = 0;
    for($myi=$start;$myi<=$end;$myi++){
        if($POSList[$myi] ne ""){
            #don't print the same category twice in a row
            if($lastPOS ne $POSList[$myi]){
                print OUTFILE "$POSList[$myi],";
                $POSprintedCount +=1;
            }
            $lastPOS = $POSList[$myi];
        }
    }
}
#fill out the attribute list with "NONE" until we've reached 16 attributes
#this is done because WEKA requires all lists have the same number of
#attributes
for($myi=$POSprintedCount;$myi<16;$myi++){
    print OUTFILE"NONE,";
}
}

```

```

#Perform a breadth first search to find the shortest path through the graph
#of lexical relations from $start to $end
sub breadthfirst{
  my($start,$end) = @_;
  my($myi);
  #reset the list of visited nodes
  %visited = %blank;
  $lastRelPrinted="";
  $totalPrinted = 0;
  #call the recursive versions of the BFS
  breadthfirst_rec($start,$end);
  #fill out the attributes with NONE until we have 40 attributes
  #this is done because WEKA requires all lists have the same number of
  #attributes
  for ($myi=$totalPrinted; $myi<40;$myi++){
    print OUTFILE "NONE,";
  }
}

#recursive breadth first search helper function for breadthfirst
sub breadthfirst_rec{
  my($start,$end) = @_;
  my($bfsString);
  my($myi);
  my($next);
  #set this node to be visited
  $visited{$start} = "true";
  #if we've made it to the end, return true
  if($start == $end){
    return "true";
  }
  @thisarray = @edge[$start];
  #for every node to which there may be an edge
  for($myi = 0;$myi<$wordCount;$myi++){
    #if there is an edge to this node
    if($edge[$start][$myi] ne "" && $visited{$myi} ne "true"){
      #recursively call BFS on that node
      if(breadthfirst_rec($myi,$end) eq "true"){
        #if we found a path to the end via that node, return true
        if($edge[$start][$myi] ne $lastRelPrinted){
          print OUTFILE "$edge[$start][$myi],";
          $lastRelPrinted="$edge[$start][$myi]";
          $totalPrinted +=1;
        }
        return "true";
      }
    }
  }
  return "false";
}

```

A.2.2 Test Data Formatting Script

```
#!/usr/local/bin/perl

#####
#Test Data Formatting Script #
#####
#Brian Harrington - July 2005 #
#####
#This script formats the enriched test data set available at: #
#http://genome.jouy.inra.fr/texte/LLchallenge/ into an attribute list which can#
#be used by most WEKA classifiers. #
#####

#####
#Define I/O files to be used #
#####

#The enriched test data set
$infile = "./enriched_testing_data.txt";
#The file to be used in WEKA
$outfile = "./brian_testing_data.arff";
#A listing of the IDs and entity names corresponding to each attribute
#lists
$outfile2 = "./brian_testing_IDlines.txt";
#The dictionary of named entities
$dictfile = "./test_dictionary.txt";

#####
#Create The Dictionary #
#####
open(DICTFILE, "<$dictfile") or die "Cannot open $dictfile";
#For each line of the dictionary file, parse out each typographical variant of
#the entity, and create an entry in an associative array for that variant which
#points to the canonical form of the word. This allows us to easily retrieve
#the canonical form of any dictionary word
while($dictLine = <DICTFILE>){
    #retrieve canonical form of word
    ($canonical,$others) = $dictLine =~ /(.*?)[\n\t](.*)/;
    #create an entry for the canonical form pointing to itself
    $dictionary{$canonical} = $canonical;
    #for each variant
    while($others ne ""){
        $hold = $others;
        #retrieve the variant
        ($next,$others) = $others =~ /(.*?)[\n\t](.*)/;
        #create the entry for this variant
        if($next ne ""){
            $dictionary{$next} = $canonical;
        }else{
            #this is only performed for the final variant in the list
            $dictionary{$hold} = $canonical;
        }
    }
}
close(DICTFILE);

#####
#Open Necessary Files & Create Headers #
#####
```

```

#####

open(INFILE, "< $infile") or die "cannot open file $infile";
open(OUTFILE, "> $outfile") or die "cannot open file $outfile";
open(OUTFILE2, "> $outfile2") or die "cannot open file $outfile2";

#Create the attribute list header, *NOTE* this list has been shortened to only
#include 5 POS (morpho syntactic category) attributes and 1 relation attribute
#to facilitate easier printing
print OUTFILE (<<EOH);
\@relation genic_interaction
\@attribute LENGTH real
\@attribute POS1 { N, V, ADV, ADJ, V_PASS, NONE}
\@attribute POS2 { N, V, ADV, ADJ, V_PASS, NONE}
\@attribute POS3 { N, V, ADV, ADJ, V_PASS, NONE}
\@attribute POS4 { N, V, ADV, ADJ, V_PASS, NONE}
\@attribute POS5 { N, V, ADV, ADJ, V_PASS, NONE}
\@attribute REL1 {subj, subj-1, comp_for, comp_for-1, comp_to, comp_to-1, comp_of,
  comp_of-1, comp_in, comp_in-1, comp_with, comp_with-1, comp_on, comp_on-1,
  comp_within, comp_within-1, comp_by, comp_by-1, comp_beyond, comp_beyond-1,
  comp_unlike, comp_unlike-1, comp_like, comp_like-1, comp_near, comp_near-1,
  comp_at, comp_at-1, comp_upon, comp_upon-1, comp_via, comp_via-1, comp_through,
  comp_through-1, comp_under, comp_under-1, comp_than, comp_than-1, comp_during,
  comp_during-1, comp_as, comp_as-1, comp_from, comp_from-1, mod, mod-1, mod_pred,
  mod_pred-1, mod_att, mod_att-1, comp_between, comp_between-1, mod_post, mod_post-1,
  obj, obj-1, appos, appos-1, neg, neg-1, NONE}
\@attribute CLASSIFICATION { AtoT, TtoA, NEG}
\data
EOH

#For each sentence
while($IDline = <INFILE>){
  #If this is not a proper sentence line (e.g. it is a comment line) ignore
  #it and move on
  unless($IDline =~ /^ID(\s+)(.*)/){
    next;
  }
  chomp($IDline);
  #The line of text
  $sentline = <INFILE>;
  #The line with the numbers of each words
  $wordline = <INFILE>;
  #The lemmatizations of all words
  $lemmaline = <INFILE>;
  #The syntactic relations
  $relationsline = <INFILE>;
  #an empty line
  $dummyline = <INFILE>;
  #split the Information in each line from their labels
  ($dummy, $sentence) = $sentline =~ /^sentence(\s+)(.*)/;
  ($dummy, $words) = $wordline =~ /^words(\s+)(.*)/;
  ($relations) = $relationsline =~ /^syntactic_relations(.*)/;

#####
#Processing the relations line, extract relations between words and      #
#POS categories                                                         #
#####

#The count of relations printed thus far
$relCount = 0;
#reset the arrays

```

```

@POSList = @blank;
@relList = @blank;
@edge = @blank;

#For each relation
while($relations ne ""){
  #Parse up the relation text to extract the indicies of the words in the
  #relation and the type of relation
  ($dummy,$relText,$indexA,$indexB,$relations) = $relations =~
    /\s+(\s+)relation\(['\`']+)',(\d+),(\d+)\)(.*)/;
  #Split the relation type from the POS categories
  ($relType,$posA,$posB) = $relText =~/^([:]+):([^-]+)-(.*)/;
  #If its the appos case, just create the relType variable, if not, then
  #update the POS tags
  if($relType eq ""){
    $relType = $relText;
  }else{
    $POSList[$indexA] = $posA;
    $POSList[$indexB] = $posB;
  }
  #only take the main part of the relation (Not in use in final system)
  #if($relType =~ /([^-]+)_(.*)/){
  #  ($relType,$dummy) = $relType =~ /([^-]+)_(.*)/;
  #}
  #Create a 2-dimensional array to store for each word, an array of
  #relations in which it is the first element
  $relArray1st[$indexA][$relArray1stsize[$indexA]] = $relType;
  $relArray1stsize[$indexA] +=1;
  #Create a 2-dimensional array to store for each word, an array of
  #relations in which it is the second element
  $relArray2nd[$indexB][$relArray2ndsize[$indexB]] = $relType;
  $relArray2ndsize[$indexB] +=1;

  #create the graph with the words as nodes, and label the edges with
  #the type of relation connecting them.
  $edge[$indexA][$indexB] = $relType;
  $edge[$indexB][$indexA] = $relType;
}

#####
#Processing the words line, extrat the words, convert dictionary words to #
#canonical form, and insert them into an array to be printed later      #
#####
$lastprinted = 0;
#Number of dictionary words printed so far
$geneCount = 0;
#Array of dictionary words encountered by word number
@genesArray = @blank;
#Array of dictionary words encountered by canonical form of word
@geneNameArray = @blank;
#Array of word specifications (word, start/end positions, word number)
@wordSpecArray = split(/\t/, $words);
#Number of dictionary words seen so far
$dicWordCount = 1;
#Number of words seen so far
$wordCount=0;
#For each word in the sentence
foreach $wordSpec (@wordSpecArray){
  $wordCount +=1;
  #parse out the word, its start/end position, and its word number
  ($wordNum,$thisWord,$wordStart,$wordEnd) = $wordSpec =~

```

```

/^word\((\d+),'(.*?)',(\d+),(\d+)\)/;
#the punctuation preceeding the word
$outString = substr($sentence,$lastprinted,$wordStart - $lastprinted);
#the actual word as it appears in the sentence, with punctuation
$thisWord = substr($sentence,$wordStart,($wordEnd - $wordStart) + 1);
#If this is a dictionary word, translate it to canonical form, and
#put it into an array to be dealt with later
if(exists $dictionary{$thisWord}){
    $dicWordCount +=1;
    $genesArray[$geneCount] = $wordNum;
    $geneNameArray[$geneCount] = $dictionary{$thisWord};
    $geneCount +=1;
}
}

#####
#Printing the attribute lists for all pairs of entities in the sentence #
#####
for($i=0;$i<($geneCount-1);$i++){
    for($j = ($i+1);$j<$geneCount;$j++){
        #Print the length attribute
        print OUTFILE ($genesArray[$j] - $genesArray[$i]).",";
        #Print the POS categories between the entities
        printbetween($genesArray[$i],$genesArray[$j]);
        #Print the shortest path through the graph of lexical relations
        breadthfirst($genesArray[$i],$genesArray[$j]);
        #Put a question mark in the classification attribute to tell WEKA
        #this is the attribute it must attempt to find
        print OUTFILE "?";
        print OUTFILE "\n";
        chomp($IDline);
        #print the other information to the secondary file, so that we
        #can recover this data later
        print OUTFILE2 "$IDline | $geneNameArray[$i] | $geneNameArray[$j] \n";
    }
}

}

#close all open files
close(INFILE);
close(OUTFILE);

#Print all the POS categories in the sentence from position
#$start to position $end
sub printbetween{
    my($start,$end) = @_;
    $lastPOS = "";
    $POSprintedCount = 0;
    for($myi=$start;$myi<=$end;$myi++){
        if($POSList[$myi] ne ""){
            #don't print the same category twice in a row
            if($lastPOS ne $POSList[$myi]){
                print OUTFILE "$POSList[$myi],";
                $POSprintedCount +=1;
            }
            $lastPOS = $POSList[$myi];
        }
    }
}
}

```

```

#fill out the attribute list with "NONE" until we've reached 16 attributes
#this is done because WEKA requires all lists have the same number of
#attributes
for($myi=$POSprintedCount;$myi<16;$myi++){
    print OUTFILE"NONE,";
}
}

#Perform a breadth first search to find the shortest path through the graph
#of lexical relations from $start to $end
sub bredthfirst{
    my($start,$end) = @_;
    my($myi);
    #reset the list of visited nodes
    %visited = %blank;
    $lastRelPrinted="";
    $totalPrinted = 0;
    #call the recursive versions of the BFS
    bredthfirst_rec($start,$end);
    #fill out the attributes with NONE until we have 40 attributes
    #this is done because WEKA requires all lists have the same number of
    #attributes
    for ($myi=$totalPrinted; $myi<40;$myi++){
        print OUTFILE "NONE,";
    }
}

#recursive bredth first search helper function for bredthfirst
sub bredthfirst_rec{
    my($start,$end) = @_;
    my($bfsString);
    my($myi);
    my($next);
    #set this node to be visited
    $visited{$start} = "true";
    #if we've made it to the end, return true
    if($start == $end){
        return "true";
    }
    @thisarray = @edge[$start];
    #for every node to which there may be an edge
    for($myi = 0;$myi<$wordCount;$myi++){
        #if there is an edge to this node
        if($edge[$start][$myi] ne "" && $visited{$myi} ne "true"){
            #recursively call BFS on that node
            if(bredthfirst_rec($myi,$end) eq "true"){
                #if we found a path to the end via that node, return true
                if($edge[$start][$myi] ne $lastRelPrinted){
                    print OUTFILE "$edge[$start][$myi]," ;
                    $lastRelPrinted="$edge[$start][$myi]";
                    $totalPrinted +=1;
                }
            }
            return "true";
        }
    }
}
return "false";
}
}

```

A.2.3 Weighted Voting & Results Formatting Script

```
#!/usr/local/bin/perl

#####
#Weighted Voting & Results Formatting Script #
#####
#Brian Harrington - July 2005 #
#####
#This script takes the output of 4 WEKA classifiers and for each classification #
#computes the mean probability for each classification type. It then produces #
#output based on those probabilities in a format acceptable by the LLL05 #
#scoring service, which can be found at #
#http://genome.jouy.inra.fr/texte/LLLchallenge/scoringService.php #
#####

#####
#Define I/O files to be used #
#####
#The sentence ID relating to each classification instance, assumed to be one #
#ID and 2 dictionary words (in canonical form) per line
$IDfile = "./brian_testing_IDlines.txt";
#The WEKA output files
$resultsFile1 = "./brian_weka_results1.txt";
$resultsFile2 = "./brian_weka_results2.txt";
$resultsFile3 = "./brian_weka_results3.txt";
$resultsFile4 = "./brian_weka_results4.txt";
#The result file, which can be uploaded to the LLL05 scoring service
$outfile = "./brian_results.txt";

#####
#Open Necessary Files #
#####
open(IDFILE, "< $IDfile") or die "Cannot open $IDfile";
open(RESFILE1, "< $resultsFile1") or die "Cannot open $resultsFile1";
open(RESFILE2, "< $resultsFile2") or die "Cannot open $resultsFile2";
open(RESFILE3, "< $resultsFile3") or die "Cannot open $resultsFile3";
open(RESFILE4, "< $resultsFile4") or die "Cannot open $resultsFile4";

open(OUTFILE, "> $outfile") or die "Cannot open $outfile";

#Print the appropriate header, the coreference distinction can be changed
#based on whether or not we are testing with coreference.
print OUTFILE (<<EOH);
% Participant name: Brian Harrington
% Participant institution: Oxford University
% Participant email address: brian.harrington@stx.ox.ac.uk
% Format checked: YES
% Basic data: NO
% Coreference distinction: WITH AND WITHOUT COREFERENCE
EOH

#The last ID printed, this is used to see if we need to begin a new sentence
#entry, or if we are just continuing to build the sentence entry on which
#we are currently working
$lastID = "";

#For each classification instance
while($idLine = <IDFILE>){
    #Retrieve the the ID number and the two dictionary words being classified
    #from the IDFILE.
```

```

($id,$word1,$word2) = split(/ \| /,$idLine);
#Remove extraneous spaces
chomp($word2);
chop($word2);

#####
#Retrieve probabilities from WEKA output files
#####

#For each weka output file, read in the line pertaining to this classification
#instance, and retrieve the probabilities for each classification type.
$wekaLine1 = <RESFILE1>;
($dummy,$dummyNum,$dummyQMark,$result1,$dummyPlus,$AtoTProb1,$TtoAProb1,$NegProb1)
    = split(/\s+/, $wekaLine1);
$wekaLine2 = <RESFILE2>;
($dummy,$dummyNum,$dummyQMark,$result2,$dummyPlus,$AtoTProb2,$TtoAProb2,$NegProb2)
    = split(/\s+/, $wekaLine2);
$wekaLine3 = <RESFILE3>;
($dummy,$dummyNum,$dummyQMark,$result3,$dummyPlus,$AtoTProb3,$TtoAProb3,$NegProb3)
    = split(/\s+/, $wekaLine3);
$wekaLine4 = <RESFILE4>;
($dummy,$dummyNum,$dummyQMark,$result4,$dummyPlus,$AtoTProb4,$TtoAProb4,$NegProb4)
    = split(/\s+/, $wekaLine4);

#Strip of astrixs from chosen options in output
if($AtoTProb1 =~ /\*(.*)/){
    $AtoTProb1 = $1;
}
if($AtoTProb2 =~ /\*(.*)/){
    $AtoTProb2 = $1;
}
if($AtoTProb3 =~ /\*(.*)/){
    $AtoTProb3 = $1;
}
if($AtoTProb4 =~ /\*(.*)/){
    $AtoTProb4 = $1;
}
if($TtoAProb1 =~ /\*(.*)/){
    $TtoAProb1 = $1;
}
if($TtoAProb2 =~ /\*(.*)/){
    $TtoAProb2 = $1;
}
if($TtoAProb3 =~ /\*(.*)/){
    $TtoAProb3 = $1;
}
if($TtoAProb4 =~ /\*(.*)/){
    $TtoAProb4 = $1;
}
if($NegProb1 =~ /\*(.*)/){
    $NegProb1 = $1;
}
if($NegProb2 =~ /\*(.*)/){
    $NegProb2 = $1;
}
if($NegProb3 =~ /\*(.*)/){
    $NegProb3 = $1;
}
if($NegProb4 =~ /\*(.*)/){
    $NegProb4 = $1;
}
}

```

```

#####
#Compute average probabilities and choose maximum #
#####

#Compute the average probability for each of the classification types
$AtoTProb = $AtoTProb1 + $AtoTProb2 + $AtoTProb3 + $AtoTProb4;
$AtoTProb = $AtoTProb/4;
$TtoAProb = $TtoAProb1 + $TtoAProb2 + $TtoAProb3 + $TtoAProb4;
$TtoAProb = $TtoAProb/4;
$NegProb = $NegProb1 + $NegProb2 + $NegProb3 + $NegProb4;
$NegProb = $NegProb/4;
#Square the probability for the NEG option. Since all probabilities are
#less than or equal to 1, this decreases the odds of having a result of NEG
#and will therefore increase recall of the overall program.
$NegProb = $NegProb * $NegProb

#Choose the classification of highest probability
if($NegProb > $AtoTProb && $NegProb > $TtoAProb){
    $result = "NEG";
}elsif($TtoAProb > $NegProb && $TtoAProb > $AtoTProb){
    $result = "TtoA";
}else{
    $result = "AtoT";
}

#####
#Format Output #
#####

#If the result is negative, do nothing
if($result eq "NEG" || $word1 eq $word2){
    print "";
}
#If the result is AtoT, use the first entity as the agent, and the second
#as the target.
}elsif($result eq "AtoT"){
    #If we are continuing to build on the output of the same sentence as
    #the last instance, just add to the agent, target, and genic_interaction
    #lines as appropriate.
    if($lastID eq $id){
        $agentsLine = $agentsLine." agent(\'$word1\)\'";
        $targetsLine = $targetsLine." target(\'$word2\)\'";
        $intsLine = $intsLine." genic_interaction(\'$word1\','\'$word2\)\'";
    }
    #If we're starting on a new sentence, print out the formatted output for
    #the last sentence, and then start on a formatted output for this
    #sentence
    else{
        print OUTFILE $lastID."\n";
        print OUTFILE $agentsLine."\n";
        print OUTFILE $targetsLine."\n";
        print OUTFILE $intsLine."\n\n";
        $lastID = $id;
        $agentsLine = "agents agent(\'$word1\)\'";
        $targetsLine = "targets target(\'$word2\)\'";
        $intsLine = "genic_interactions genic_interaction(\'$word1\','\'$word2\)\'";
    }
}
#If the result is TtoA, use the first entity as the target, and the second
#as the agent.
}elsif($result eq "TtoA"){
    #If we are continuing to build on the output of the same sentence as
    #the last instance, just add to the agent, target, and genic_interaction

```

```

#lines as appropriate.
if($lastID eq $id){
    $agentsLine = $agentsLine." agent(\'$word2\)\'";
    $targetsLine = $targetsLine." target(\'$word1\)\'";
    $intsLine = $intsLine." genic_interaction(\'$word2\','\'$word1\)\'";
}
#If we're starting on a new sentence, print out the formatted output for
#the last sentence, and then start on a formatted output for this
#sentence
else{
    print OUTFILE $lastID."\n";
    print OUTFILE $agentsLine."\n";
    print OUTFILE $targetsLine."\n";
    print OUTFILE $intsLine."\n\n";
    $lastID = $id;
    $agentsLine = "agents agent(\'$word2\)\'";
    $targetsLine = "targets target(\'$word1\)\'";
    $intsLine = "genic_interactions genic_interaction(\'$word2\','\'$word1\)\'";
}
}
#If we reach this point, there is an error: print an error message to the screen and exit
else{
    print "ERROR $result is not an expected result";
    exit(1);
}
}
#Close all open files
close(IDFILE);
close(RESFILE1);
close(RESFILE2);
close(RESFILE3);
close(OUTFILE);

```