

IMPROVING HIERARCHICAL MULTICLASS PERCEPTRONS
FOR ENTITY DETECTION USING
GOOGLE SETS AND PRONOUN
DISAMBIGUATION

by

Michael Quinn

APPROVED BY SUPERVISORY COMMITTEE:

Dr. Vincent Ng, Chair

Dr. Yang Liu

Dr. Latifur Khan

© Copyright 2008
Michael Quinn
All Rights Reserved

To my lovely wife, Brook. I don't know why she puts up with me.

To my enlightening advisor, Dr. Vincent Ng. I don't know why he puts up with me.

And to my cantakerous cat, Spooky. I don't know why I put up with him.

IMPROVING HIERARCHICAL MULTICLASS PERCEPTRONS
FOR ENTITY DETECTION USING
GOOGLE SETS AND PRONOUN
DISAMBIGUATION

by

MICHAEL QUINN, B.S.C.S.

THESIS

Presented to the Faculty of
The University of Texas at Dallas
in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

May, 2008

ACKNOWLEDGEMENTS

I would like to thank my advisor, for if he had never pushed my mental limits in the classroom, I would never have had the wise idea to do a thesis in the first place.

I would like to thank my wife for tolerating me, in general, but especially during the final push to complete this thesis. I know she will look back fondly on these years, thinking about how wonderful it is that I will never have to repeat them.

Finally, I would like to thank my team at Verizon Wireless. Without their flexibility, it really would have taken me five years to finish my Bachelors and that would have destroyed any motivation to continue on with my Masters. Thank you Brady, Gene and Rik for each making special efforts to make sure I able to do what needed to be done for school without leaving work undone for others to pick up.

April, 2008

IMPROVING HIERARCHICAL MULTICLASS PERCEPTRONS
FOR ENTITY DETECTION USING
GOOGLE SETS AND PRONOUN
DISAMBIGUATION

Publication No. _____

Michael Quinn, M.S.C.S.
The University of Texas at Dallas, 2008

Supervising Professor: Dr. Vincent Ng

Entity detection is the process of identifying objects in a body of text and correctly classifying each object into one of a set of predefined types. I propose two extensions to improve the performance of hierarchical multiclass perceptrons when used to solve entity detection. The first extension is to use Google Sets to automatically generate training instances. This will address the problems of too few training instances and high unseen word rates. The second extension is to disambiguate pronouns by identifying the influences on the pronoun classification, and using those influences as features. This will address the problem that pronouns carry less information than nominals and proper nouns. A noticeable improvement in F-measure was seen using the combination of these two approaches.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS.....	v
ABSTRACT.....	vi
LIST OF TABLES.....	ix
LIST OF FIGURES.....	x
CHAPTER 1. INTRODUCTION.....	1
CHAPTER 2. RELATED WORK.....	3
2.1 Named Entities.....	3
2.2 Entity Detection.....	3
2.3 Word Sense Disambiguation.....	3
CHAPTER 3. PROBLEM DESCRIPTION.....	5
CHAPTER 4. APPROACH.....	8
4.1 Introduction.....	8
4.2 Phase 1: Phrase Finding.....	9
4.2.1 Section Overview.....	9
4.2.2 Features.....	10
4.2.3 Training File Creation.....	16
4.3 Phase 2: Primary Type/Subtype Labeling.....	21
4.3.1 Section Overview.....	21
4.3.2 Training Set Transformation.....	22
4.3.3 Hierarchical Multiclass Perceptron.....	23
4.3.4 Using Google Sets to Improve Hierarchical Multiclass Perceptron ...	28
4.4 Phase 3: Secondary Type/Subtype Labeling - Pronoun Disambiguation ...	32
4.4.1 Overview.....	32
4.4.2 Creating Training Instances.....	33
4.4.3 Features.....	33

CHAPTER 5. EVALUATION	35
5.1 Experimental Setup	35
5.2 Evaluation	37
5.2.1 Phase 1: Phrase Identification	38
5.2.2 Phase 2: Primary Type/Subtype Labeling	38
5.2.3 Phase 3: Secondary Type/Subtype Labeling	41
5.3 Overall Error Analysis	43
5.3.1 Fold 2 Error Analysis	43
5.3.2 Type Error Analysis	43
5.3.3 Subtype Error Analysis	44
CHAPTER 6. CONCLUSION	47
REFERENCES	49
VITA	

LIST OF TABLES

3.1	ACE Type and Subtype Information	7
4.1	Bikel Feature	12
4.2	Fold Distribution for the ACE 2005 Corpus	21
5.1	Article Distribution for the ACE 2005 Corpus	35
5.2	Type Statistics for the ACE 2005 Corpus	36
5.3	Subtype Statistics for the ACE 2005 Corpus	36
5.4	Phase 1 Results - Phrase Identification	38
5.5	Detailed Results - Flat Multiclass Perceptron	39
5.6	Detailed Results - Hierarchical Multiclass Perceptron	41
5.7	Detailed Results - Improved Hierarchical Multiclass Perceptron	42
5.8	Detailed Results - Pronoun Disambiguation Applied to Improved Hierarchical Multiclass Perceptron	42
5.9	Summary of Results	43
5.10	Analyzing Type Performance in Relation to Frequency/Phrase and Unseen Phrases	44
5.11	Overall Type Performance using Improved Hierarchical Multiclass Perceptrons and Pronoun Disambiguation	44
5.12	Analyzing Subtype Performance in Relation to Phrase Frequency and Unseen Phrase Rate	45
5.13	Overall Subtype Performance using Improved Hierarchical Multiclass Perceptrons and Pronoun Disambiguation	46

LIST OF FIGURES

3.1	Example of Entity Detection	6
4.1	Example Phrases	10
4.2	WordNet Hierarchy Example	16
4.3	Example Article Segment	17
4.4	Example XML segment	17
4.5	Example Tagged Article Segment for Type	18
4.6	Example Tagged Article Segment for Subtype	18
4.7	Example Training File Segment for Type	19
4.8	Example Training Data	20
4.9	Example Training Data	20
4.10	Example Training Data	23
4.11	Examples of ambiguous words not submitted to Google Sets	30
4.12	Feature weight calculation for word distance	34
5.1	Precision, Recall and F-measure Formulas	37

CHAPTER 1

INTRODUCTION

There is a large body of text that exists without the entities in the text identified and labeled by type and subtype. It is probably safe to say that most text has not been labeled in such a fashion, and since such labeling would be of great assistance to information retrieval, text summarization and coreference resolution it seems worthwhile to attempt to improve upon the existing techniques for identifying and labeling entities with type and subtype information, a task that is commonly known as entity detection.

When developing new techniques to address an existing problem, it makes sense to try to build upon the work that others have done on the same problem or a similar problem. Named entity recognition is a similar problem that attempts to identify the proper names of persons, locations and organizations as well as date/time information and numerical information. Given that the named entity recognition problem is well studied, and that the problem is very similar to labeling entities with type and subtype information, named entity recognition research should provide a solid foundation for work on building a system to solve the entity detection problem.

While building on a strong foundation of similar work, sometimes great ideas come from someone else's approach to solving a problem that is not directly related. This is the case for Ciaramita et al.'s [1] work on word sense disambiguation using hierarchical multiclass perceptrons. While word sense might be a useful feature, it is the approach of exploiting hierarchical relationship between types and subtypes to improve performance while also increasing the training set size by automatically

generating new labeled training instances using data external to the training set that is really interesting.

Progress is made by moving one step forward. Named entity recognition is a good starting point but with the focus on proper nouns, such technique will probably not adequately address pronouns. Hierarchical multiclass perceptrons allows for the inclusion of additional generated training data that might not be as high of quality as the original training data, but has no way to prevent the generated training data from overwhelming the models, diluting the models and decreasing the performance of the models instead of increasing it.

The goal of this thesis is to address the two issues above, namely, the need for more training instances and pronoun ambiguity with regard to pronoun classification. Specifically, we propose to use Google Sets to automatically generate training instances as well as an improved method for incorporating those training instances into a hierarchical multiclass perceptron. In addition, we propose a new method for disambiguating pronouns. Pronouns are very ambiguous when it comes to classification because they rely on the object they reference to determine the class. By labeling pronouns after the classifications of proper and common nouns are complete, these classifications can be used to influence the pronoun classifications.

The remainder of the thesis will follow a path covering previous work, applying the knowledge gained from that work to create a new approach, evaluating that approach and concluding with the discoveries made using the new approach.

CHAPTER 2

RELATED WORK

2.1 Named Entities

Named entities are proper names of persons, locations and organizations. *IdentiFinder* is a HMM based model for solving the named entity recognition problem [2] and is an extension of a previous system, *Nymble* [3]. *IdentiFinder* introduced a highly successful learning algorithm solving named entity recognition into a field previously dominated by rule based systems [2]. While the techniques used to construct the named entity recognition model were not new, the techniques had not previously been applied to named entity recognition.

2.2 Entity Detection

Entity detection is broader of a problem than named entities, attempting to label more classes, and expanding from just proper names to include nominals and pronouns. Using the ACE 2002 corpus as motivation, Florian et al. developed a general statistical framework [4] that not only handles English text, but can also process Arabic and Chinese text for the entity detection task. Florian et al.'s approach used 10 different feature types spread across 3 different models with each model contributing to the final prediction.

2.3 Word Sense Disambiguation

Word sense disambiguation is the task of identifying the proper sense of a word in a given context that has multiple valid senses. Ciaramita et al. [1] attacked this prob-

lem with a novel approach of creating a more generic training set from WordNet to be used in conjunction with the existing training set already available. This new training data lacked the sense information needed for it to be compatible with the existing training set, but instead contained supersense information. This supersense information acted as parent to the existing sense information. An algorithm was needed that could exploit such a relationship and the hierarchical multiclass perceptron fit the requirement. Ciaramita et al.'s hierarchical multiclass perceptron was based on "Ultraconservative Online Algorithms for Multiclass Problems" [5], which detailed update methods for hierarchical multiclass perceptrons that guarantee convergence.

Word sense disambiguation does not directly apply to the problem of entity detection, but the approach of using hierarchical multiclass perceptrons to incorporate automatically generated training data to improve performance is of use and will be used as the baseline for my proposed extensions.

CHAPTER 3

PROBLEM DESCRIPTION

Entity detection is the problem of identifying objects in a body of text and classifying them into a set of predefined classes. There are two parts to this problem. First the boundaries of what needs to be labeled must be correctly identified. Second, the phrases that have been identified by these boundaries must then be correctly labeled with the right class. Figure 3.1 gives an example of entity detection in the Automatic Content Extraction (ACE) 2005 evaluation¹, which has two levels of classification. The two levels of classification are type and its more specific child, subtype. A more detailed list can be found in Table 3.1. This example was taken from the ACE 2005 Multilingual Training Corpus, which is the corpus that this thesis will focus its entity detection efforts. The details of this corpus are provided below.

The ACE 2005 Multilingual Training Corpus contains English, Arabic and Chinese training data from the 2005 Automatic Content Extraction (ACE) technology evaluation. The English portion of this corpus contains entities labeled with type and subtype information. The problem being studied is identifying the phrases that make up these entities and correctly labeling them with both the type and subtype information.

In Table 3.1 it is shown that the type-subtype relationship is a strict hierarchy with no overlap with the exception of Underspecified, which appears under both VEH and WEA. To enforce a strict hierarchy, these two subtypes are given unique labels.

¹<http://www.nist.gov/speech/tests/ace/2005/>

Time^{*ORG:Media*} is reporting today that a al-Qaeda^{*ORG:Non-Governmental*}
operative^{*PER:Individual*} was captured and is spitting out that
they^{*ORG:Non-Governmental*} have plans in the work to smuggle
nukes^{*WEA:Nuclear*} into the US^{*GPE:Nation*} from
Mexico^{*GPE:Nation*}.

Figure 3.1. Example of Entity Detection

This actually makes the classification problem slightly more difficult, but this also creates the strict hierarchy where each subtype has only one corresponding parent.

The ACE corpus is a multilingual corpus, but only the English portion of the corpus will be used for our purposes.

Table 3.1. ACE Type and Subtype Information

Type	Subtype
FAC (Facility)	Airport, Building-Grounds, Path, Plant, Subarea-Facility
GPE (Geo- Political Entity)	Continent, County-or-District, GPE-Cluster, Nation, Population-Center, Special, State-or-Province
LOC (Location)	Address, Boundary, Celestial, Land-Region-Natural, Region-General, Region-International, Water-Body
ORG (Organization)	Commercial, Educational, Entertainment, Government, Media, Medical-Science, Non-Governmental, Religious, Sports
PER (Person)	Group, Indeterminate, Individual
VEH (Vehicle)	Air, Land, Subarea-Vehicle, Underspecified, Water
WEA (Weapon)	Biological, Blunt, Chemical, Exploding, Nuclear, Projectile, Sharp, Shooting, Underspecified

CHAPTER 4

APPROACH

4.1 Introduction

Two interesting problems arise when considering the type and subtype labeling task with regard to the ACE corpus. First, by using a class size as large as subtypes (45 classes), either the ACE corpus would be huge or many classes will have very few examples to train from. Improving performance when given so many classes to train with few training instances is a worthwhile problem to attempt to overcome. Second, the existing approaches to named entity relations and entity detection rely on a small window of words to determine the class of the word in question. This approach makes sense for most cases, but in the case of pronouns, it seems that the focus is too narrow to include the object that the pronoun is referring to. Considering Florian et al. use a window of 5 words [4], if the object that the pronoun is referring to does not appear in that window, then that object makes no contribution to determining the type of the pronoun. Being able to overcome this problem as well is also an exciting challenge.

Two goals can now be set. First, identify and implement a method for automatically generating training instances that, when included in model training, improves overall performance of the system. Second, develop a method for creating pronoun features that would include the object being referred to by the pronoun.

In this chapter, the three phases to the proposed approach, identifying phrases, labeling those phrases and relabeling the pronouns, will be covered. The features will be given the most attention in during the phase one coverage with the changes and addition to those features covered in phases 2 and 3. The algorithms in phases 1

and 3 are only covered lightly since third party implementations of both algorithms were used, but phase 2 will give considerable focus to hierarchical multiclass perceptrons, which was implemented specifically for this experiment. Google Sets will also be introduced during phase 2, including what it is, how it can be used to generate automated training instances and how to effectively incorporate the additional training instances into hierarchical multiclass perceptrons. Phase 3 will focus on why relabeling pronouns makes sense, and how to extend what was accomplished in phase 2 to accomplish phase 3. The overall goal of this chapter is to cover the preparation of the experimental setup of each of these 3 phases in enough detail that the results could be easily duplicated.

4.2 Phase 1: Phrase Finding

4.2.1 Section Overview

The first phase of the experiment seems to be a simple, well understood task, but the performance achieved at this phase sets the upper limit on all the future phases. It is not possible to correctly label a phrase when the latter phrases do not know the phrase exists.

The first phase of labeling types and subtype is to identify the phrases in the training data. Conditional random fields (CRFs) are probabilistic models suited for labeling a sequence of data [6]. CRFs outperform hidden Markov models and maximum entropy Markov models [7] for several real world labeling tasks as well as not requiring a strict independence assumption on features. Since phrase identification is a word based sequence labeling problem, and CRFs are good at labeling word sequences, the two were brought together and a phrase finder was born. In the rest of this section, we will describe how to generate the instances for training the phrase finder, as well as the features used to represent each instance.

Time^{ORG:Media} is reporting today that a al-Qaeda^{ORG:Non-Governmental}
operative^{PER:Individual} was captured and is spitting out that
they^{ORG:Non-Governmental} have plans in the work to smuggle
nukes^{WEA:Nuclear} into the US^{GPE:Nation} from
Mexico^{GPE:Nation}.

Figure 4.1. Example Phrases

The term phrase will be used throughout the paper to indicate any collection of words that has been tagged by ACE. These phrases are not bound to follow any guidelines other than being deemed worthy of a single tag by the ACE tagging committee. An example of a sentence tagged with type and subtype information from the ACE corpus is shown in Figure 4.1

4.2.2 Features

The following list of feature types is derived from previous work on entity detection by Florian et al. [4]. As mentioned before, since CRF is a word based model, the features described below are all word based features.

Shallow Parsing Information

Shallow parsing contributes 2 features. The shallow parse information is generated by YamCha¹, yet another multipurpose chunk annotator, that was trained to identify

¹<http://chasen.org/~taku/software/YamCha/>

grammatical phrases, such as noun phrases and verb phrases. YamCha returns a single result per word that is broken into two features. For example, the beginning of a noun phrase would be labeled as B-NP by YamCha. Since the phrase boundaries are of great importance, even if the phrase type is wrong, the output from YamCha was split into two parts.

Parts of Speech Tags

Parts of speech tags contribute a single feature. Parts of speech tags are provided by MXPOST².

Prefixes and Suffixes up to a Length of 4

Prefixes and suffixes contribute 8 features.

Bikel

Bikel is a feature that mimics the feature used in Identifinder [2]. This feature is derived from a simple set of regular expressions run against each token in the training set. This feature is an order list, and the first match is what is used as the feature. This feature will be referred to as the Bikel feature going forward and the details of how to construct the feature can be found in Table 4.1.

Gazetteer Information

Gazetteer information provides 14 features. Six of the features are binary features that are dependent on a word lookup in a word based dictionary. The remaining 8 features are all based on phrase lookups.

²<http://www.inf.ed.ac.uk/resources/nlp/local.doc/MXPOST.html>

Table 4.1. Bikel Feature

Word Feature	Example Text	Intuition
twoDigitNum	90	Two digit year
fourDigitNum	1990	Four digit year
containsDigitAndAlpha	A8956-67	Product code
containsDigitAndDash	09-96	Date
containsDigitAndSlash	11/9/89	Date
containsDigitAndComma	23,000.00	Monetary amount
otherNum	456789	Other number
allCaps	BBN	Organization
capPeriod	M.	Person name initial
firstWord	sentence start	No useful capitalization information
initCap	Sally	Capitalized word
lowerCase	can	Uncapitalized word
other	,	Punctuation marks, all other words

Two of the binary features utilize the Unix words³ file available on most Unix systems. The first feature, the uncommon word feature, is true if the word does not appear in the words file or it appears in the words file but does not start with a lowercase letter. The logic behind this is to create a feature to aid with unseen words that may be proper names, so this feature should be true for both seen proper names (likely in the words file) and unseen proper names (not likely in the words file). The second feature is true if the word exists in the words file and that word starts with a capital letter or a number. This feature is not expected to aid in classification, but in phrase finding, since many common proper names are included in the words file.

One binary feature makes use of an attempt to create a more complete dictionary of person names. The source of these names was three pages from wikitionary, an online open dictionary project that contains appendices for surnames, female names

³The default words file from Fedora Core 6 was used for this experiment.

and male names⁴. Using this source, a dictionary of 41,784 unique names was created. This feature is a simple lookup, and true if the word appears in the dictionary.

The final three binary features were an attempt to create dictionaries to aid specific types. These dictionaries were sourced from the CIA World Factbook⁵ to aid in identifying geo-political entities, locations and organizations. For each of these three dictionaries, the common words were thrown out, using the uncommon words filter described previously, and a dictionary was created from the remaining words. If a word appears in the dictionary, the feature is true.

The 8 phrase based dictionaries were an attempt to increase the performance of phrase boundary identification and aid type and subtype classification. Moving from word based lookups to phrase based lookups takes a little work. The typical dictionary is a list of phrases, where each phrase is one or more words. To facilitate the lookups, the dictionaries are converted to trees. For example, United Bank and United Airlines would both share United at the root and the United subtree would have two leaf nodes, one for Bank and one for Airlines. Given this tree structure, each sentence is processed one word at a time recursively, with the function returning the depth of the last successful match. If there is a successful match with a depth n greater than one, the first word under consideration has a feature of B, the next $n-1$ words have a feature of I, and the word pointer moves forward n words in the sentence. When there is a successful match with a depth of 1, the word is looked up in common words, and if it is a common word, the feature is O, otherwise the feature is B and in both cases the word pointer moves one word forward in the sentence.

⁴<http://en.wiktionary.org/wiki/Appendix:Surnames>,
http://en.wiktionary.org/wiki/Appendix:Names_female,
http://en.wiktionary.org/wiki/Appendix:Names_male

⁵<https://www.cia.gov/library/publications/the-world-factbook/appendix/appendix-f.html>

Finally, if the success depth returns 0, then the word pointer moves forward one word. Processing continues until the end of the sentence is reached. Using this methodology, the standard BIO notation for labeling phrases is used as the feature value. Also note that partial matches are valid, as long as the match is greater than one word, or a one word match is not considered a common word.

The dictionary trees are independent, and although this requires a tree traversal for each phrase based feature, it does not require assigning the precedence of dictionaries.

The 8 phrase dictionaries came from a variety of sources. The SEC companies dictionary came from the list of companies required to file with the SEC⁶ and this source was chosen since it includes international businesses that do business in the United States as well as domestic corporations which should aid labeling ORG Commercial. Four dictionaries were provided for the CoNLL task, LOC, PER, ORG and MISC⁷ which should help label the types they are named after. The person names dictionary described in the binary features section was also used to build phrases of names, replacing the tree with a single word dictionary lookup in the person names dictionary aiding type PER. A list of titles was sourced from wikipedia⁸ which should also aid type PER. The finally phrase dictionary is sourced from wikipedia to aid with the type VEH⁹.

⁶<http://www.sec.gov/edgar/indices/fullindex.htm>

⁷<http://www.cnts.ua.ac.be/conll2003/ner/lists/>

⁸<http://en.wikipedia.org/wiki/Title>

⁹<http://en.wikipedia.org/wiki/Category:Vehicles>

WordNet Information

WordNet¹⁰ contributes 4 features. WordNet has the concept of synset, or synonym set, where each word in that set has a sense that is synonymous to all the words in that set. By identifying what set a word belongs to, that set should provide a more generic representation of that word. Each word in WordNet has one or more senses and these senses are ordered from statistically most likely to statistically least likely. The first two features are the synsets of the first to senses of a word.

WordNet also has a concept of hyponym and hypernym. The hyponym/ hypernym relationship forms a hierarchical relationship. For example, red, green and blue are all hyponyms of color, and color is the hypernym of red. Using this structure, one should be able to go from a more specific term to a more general term, for example from crimson to red to color. Using this pattern, it should be possible to determine if a specific word belongs to one of the ACE types. The final two features for WordNet information are the results of these lookups for the first two senses of the word. The results are not binary, but the actual synset of the ACE type if one is found.

Figure 4.2 is an example of a WordNet hierarchy for the first sense of the word "coach". Using this example, the first WordNet feature would be "coach, manager, handler" uniquely represented as a number and the third WordNet feature would be "person, individual, someone, somebody, mortal, soul" also numerically represented. The second and fourth features use the same structure but for the second sense of the word coach.

¹⁰<http://wordnet.princeton.edu/>

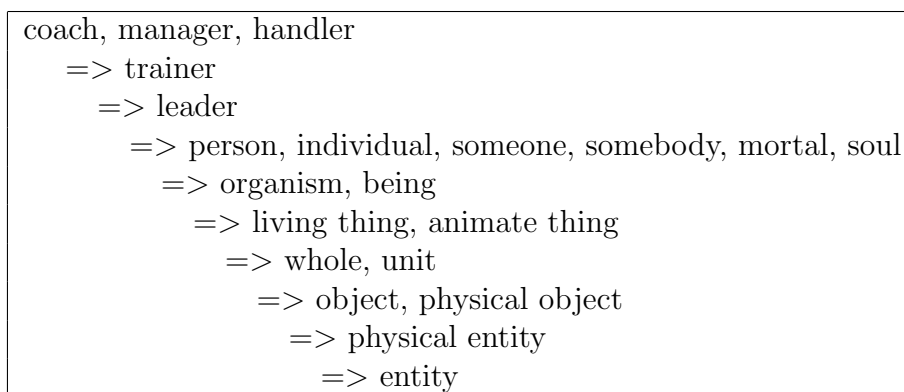


Figure 4.2. WordNet Hierarchy Example

NER Tags

NER tags provide a single feature. The named entity recognition tagger, *IdentiFinder* [2], discussed in Section 2.1, provides the NER tags. The seven NER tags are person, location, organization, percent, money, date and time.

4.2.3 Training File Creation

Since CRF is a word based model, one training instance is created for each word in a training text. Details of how to create the training files in a format that will be easily ingestible by CRF++ are given below.

Preprocessing the ACE Document

The ACE corpus is broken up into two files per article of interest. These files are the article text marked up in SGML, and entity information in a separate XML file. The XML file includes the start and end character index of the each entity, but this index is not simply a character depth into the article file, but the character depth after all the tags have been removed. While stripping the tags from the file prior to labeling simplifies the process, valuable information is lost. Many of the tags in the article file

The Davao Medical Center, a regional government hospital, recorded 19 \wedge index = 493

Figure 4.3. Example Article Segment

```

<entity ID="..." TYPE="ORG" SUBTYPE="Medical-Science">
  :
<entity_mention ID="..."
<extent>
<charseq START="493" END="516">The Davao Medical Center</charseq>
</extent>
<head>
<charseq START="497" END="516">Davao Medical Center</charseq>
</head>
</entity_mention>
  :
</entity>

```

Figure 4.4. Example XML segment

indicate text that has not been labeled but is counted when calculating the positions of the labels. This tag information needs to be retained to build the best training instances.

The first step in computing the type and subtype information of each word is combining the two files into one file. Using the indexes of the start and ends of each label (see Figure 4.4), the article file (see Figure 4.3) can be modified to include in line tags identifying the class values (see Figure 4.5). This is a simple process of inserting tags from the highest index to the lowest index, into the article file which prevents the need to track shifts in the indexes if the tags were inserted in order from lowest to the highest index.

After the two files have been combined into a single file, this new tagged file can now be processed in ways that would have previously caused problems maintaining

The <ORG>Davao Medical Center</ORG>, a regional government hospital, recorded 19

Figure 4.5. Example Tagged Article Segment for Type

The <Medical-Science>Davao Medical Center</Medical-Science>, a regional government hospital, recorded 19

Figure 4.6. Example Tagged Article Segment for Subtype

the label positions. Several of our features are generated by external application that prefers their text to be tokenized and work best on a sentence of data at a time. To accommodate these programs the Treebank tokenizer was used. The next step after tokenization is sentence boundary identification, and for this MXTerminate [8] was used. With the preparation finally completed, the addition of derived features can begin.

Creating Training Instances

CRF is a word based model requiring the generation of word based training instances, and these training instances need to be grouped into sentences. The initial preparation needed to compute the class values left one file per article, with one sentence per line and words are space delimited. See Figure 4.7 for an illustration on transforming tagged words into training instances. As can be seen, each training instance is labeled with one of three labels: B (the beginning of a phrase), I (the continuation of a phrase), and O (a word that is not part of a phrase). For each new line seen in the tagged article file, an extra new line is inserted into the training file which will separate sentences by a blank line.

This provides the base training file with no features. CRF++ includes a configuration file to specify which features to include, so including the type as show

The <ORG> Davao Medical Center </ORG> ,
becomes

WORD	TYPE	PHRASE BOUNDARY
The	O	O
Davao	ORG	B
Medical	ORG	I
Center	ORG	I
,	O	O

Figure 4.7. Example Training File Segment for Type

in Figure 4.7 does not require using it. The type and subtype will be added to the training files, but ignored, since these training files will be used to build the training files needed for phases 2 and 3, it is useful to have the labels needed for the latter phases available.

The features that only require lookups will be added to the training file a word at a time or a sentence at a time as described in the features section. This leaves the addition of the features generated by the external programs MXPOST, YamCha and Identifinder. To add the parts of speech feature to the training set, a sentence worth of words is feed into MXPOST all on one line, and MXPOST returns one line with the same number of words, with the tags appended to them.¹¹ The line is split up on spaces, the tags are removed from the words and added to the training set. YamCha is even simpler, since it takes the same format as CRF++ and needs parts of speech tags to do shallow parsing. The output from YamCha when run against the current training file results in the next iteration of the training file. Finally, Identifinder tags sentences in line, just like was used when combining the original files as shown in Figures 4.3, 4.4 and 4.5. The same process will be used to extract the tags from the Identifinder, and the ordered list of tags will be added to the training file.

¹¹MXPOST converts *word* into *word_label*

The	O	O	N	X	<< Current word
Davao	ORG	B	Y	Z	
Medical	ORG	I	Y	R	
Center	ORG	I	Y	Z	
,	1	2	3	4	

Figure 4.8. Example Training Data

Template	Expanded feature
U01:%x[0,0]	U01:Medical
U05:%x[-1,0]/%x[0,0]	U05:Davao/Medical
U105:%x[0,0]/%x[1,0]/%x[2,2]	U105:Medical/Center
U15:%x[1,1]	U15:ORG

Figure 4.9. Example Training Data

This concludes the 31 features that need to be added to the training files. The remaining features used for phase 1 are n-gram features controlled by the CRF++ configuration file.

CRF++ Configuration

CRF++ makes use of a configuration file that not only allows the inclusion and exclusion of features for a given word, features from other words relatively close to the word can be added as well. The best way to explain this is with an example. In Figure 4.8 there is an example training file and Figure 4.9 shows how a line in the configuration file would be converted into a feature. Using the power of the configuration file, n-gram features can easily be created. To easily explain the n-gram features, let's use the notation that 0 is the current word, -1 is the previous word and 1 is the next word. Using that notation, there are 3 1-gram features, (-1), (0) and (1), 2 2-gram features, (-1,0) and (0,-1) and 3 3-gram features (-2,-1,0), (-1,0,1) and (0,1,2).

Table 4.2. Fold Distribution for the ACE 2005 Corpus

Article Class	Total Articles	Fold 0	Fold 1	Fold 2	Fold 3	Fold 4
Broadcast Conversation	52	11	10	11	10	10
Conversational Telephone Speech	34	8	6	7	6	7
Broadcast News	34	44	43	44	43	43
News Magazine	81	17	16	16	16	16
Newsgroups	37	8	7	8	7	7
Weblogs	114	24	22	23	22	23

Training and Testing

With the training files complete and the configuration file complete, training and test can proceed. The training/ test split follows a standard 5 fold cross validation. Details of the evaluation procedure can be found in Chapter 5.

4.3 Phase 2: Primary Type/Subtype Labeling

4.3.1 Section Overview

The second phase of this type/subtype labeling approach uses hierarchical multiclass perceptrons to label all the phrases identified by phase 1. Since the phrase based hierarchical multiclass perceptrons will be using essentially the same set of features that were used to train the phrase finder, the rest of this section will begin by describing how the word based features are transformed into phrase based features for training the perceptrons. After that, an overview of hierarchical multiclass perceptrons will be provided. Finally, the first extension to the hierarchical multiclass perceptron approach to entity detection — using Google set to automatically generate training instances — will be presented.

4.3.2 Training Set Transformation

Phase 2 attempts to label all the phrases identified in phase 1. Now that the problem is phrase based instead of word based, the training setup will need to be transformed to reflect that. A transformation is all that is needed, since the set of general features used in phrase one is maintained for phase 2.

A good understanding of what is covered in Figure 4.9 will be needed to understand the transformation process. The training file format for the perceptrons and the maximum entropy program used in phase 3 are the same. Each phrase has a training instance and there is one training instance per line in the training files. The line is space delimited, with the first column in the file being the class and all remaining columns being features. The features are unordered and variable length, making the training file very easy to construct correctly.

The transformation from CRF++ to the common format uses the exact transformation diagrammed in Figure 4.9. To identify phrases to convert and add to the common format training file, the CRF++ training files is processed line by line. When a training instance is found with a B label, all the features where first index of the %x are less then or equal to 0 are included. For example, the rules U01:%x[0,0] and U12:%[-2,0] would be included but U22:%x[2,0] would not. All the forward rules, rules where the first index of the %x is greater than 0, are held in memory. If the label after the B label is another B or an O, the features held in memory are written out to the existing training instance, and that training instance is ended, otherwise the features in memory are discarded and the training instance is continued. When the label is I, the features are generated where the %x index equals 0 are added to the training instance, and the features where the index is greater than 0 are held in memory. Following these rules, the features of the first word in a phrase that look

ORG U01:Medical U01:Davao U01:Center U05:The/Davao U105:Center/, U15:ORG U15:ORG U15:1

Figure 4.10. Example Training Data

back before the phrase will be given to the phrase, the features of the last word in the phrase that look after the phrase will be given to the phrase, and all interior words will only contribute features that attributable that that specific word.

Again referencing Figure 4.9, the transformation rules would result in Figure 4.10. Repeated features (i.e., the same feature that appears more than once in a single training instance) will be treated as a single feature, since the features are considered binary by both algorithms. Both the perceptron models used in Phase 2 and the maximum entropy models used in phase 3 take the same input format, so the phase 3 will start with the phase 2 training instances and add features.

4.3.3 Hierarchical Multiclass Perceptron

As mentioned previously, hierarchical multiclass perceptrons will be used to label the automatically identified phrases with type and subtype information. Hierarchical multiclass perceptrons were chosen over other learning algorithms for their ability to incorporate additional automatically generated training instances that only incorporated a subset of the feature set of the manually labeled training instances provided by the ACE corpus. For this reason, hierarchical multiclass perceptrons will be considered the baseline approach.

Before describing the hierarchical multiclass perceptron learning algorithm, the more basic perceptrons will be covered briefly to provide the grounding necessary to fully understand the hierarchical multiclass perceptron.

Binary Perceptron

A binary perceptron [9] takes an instance of an object and the set of features related to that instance and then uses that information to determine if the instance is or is not a member of a specific class. The basic structure of a perceptron is made up of two parts, a weight vector that includes a weight for all possible features, and an activation function that returns a true/false value for membership in the class. The simplest activation function returns true if the dot product of the feature vector and the weight vector is greater than 0.

Training a binary perceptron requires a learning rate η , the perceptron weight vector W , and a labeled training set. The training set is made up of training instances, and each instance has a discrete value of $(1, -1)$ that defines the instances inclusion or exclusion from the class, and a set of features in a feature vector X .

The learning rate η effects how quickly the weight vector changes as errors are discovered during training. A η that is too large will cause rapid learning, but at the possibility that the perceptron will not converge but instead oscillate around the optimal weight vector, while η that is too small will take a long time to converge.

The binary perceptron is trained by iterating over each training instance and updating the weight w_i in the weight vector W for each corresponding feature in the training instances feature vector when an error is detected. When no error is detected, no updates occur and training proceeds with the next training instance. One iteration over every training instance in the training set is referred to as an epoch. Training a perceptron can be thought to occur in an infinite loop that is broken at the end of an epoch when an end of training condition is met. Linearly separable problems or training sets that appear to describe linearly separable problems, will terminate after one epoch without any errors being detected [10]. Problems that do not meet

the first condition will need an implementation specific solution, such as termination when oscillation is detected or termination when training error ceases to improve.

The problem of determining type and subtype has been found to not be a linearly separable problem while researching this paper and it has also been found with the given training set that over fitting the training data is common. To address these two points, the labeled data was divided into 5 folds, 3 folds were used for training, 1 fold was heldout and 1 fold was test data. Two rounds of training were preformed. The first round of training was performed for 10 epochs, at which point it was the training results and heldout results had significantly diverged. After collecting the results after each epoch, the best heldout results were used to determine when to terminate training and the perceptron was trained a second time for optimal number of epochs. This final trained perceptron was used to collect test results, and this same methodology was used for all variations of perceptrons.

Multiclass Perceptron

The multiclass perceptron builds upon the binary perceptron model to create a new model that allows for an instance to be identified as one of many possible classifications. The multiclass perceptron maintains a weight vector W for each class that is to be identified by the multiclass perceptron. Given a feature vector x , the weighted sum is calculated for each class, and the multiclass perceptron returns the class with the max weighted sum.

Training a multiclass perceptron requires a training set where each instance of the training set includes a class y_i and a feature set for that instance x_i . Like the binary perceptron, the multiclass perceptron is trained one epoch at a time, until an exit condition is met. Also like the binary perceptron, during training, if the predicted class matches the target class, then no updates occur and training proceeds to the

next training instance. The removal of the activation function and the addition of a weight vector for each class do require changes to the update function.

An update occurs during training, for any instance where the multiclass perceptron predicts a class different from the target class. Each weight in the target class weight vector is increased by 1 if the corresponding feature is active in the training instance. Next define a set E to be the classes that had weighted sums greater than the target class. The weight vector for each class in E will need to have weights corresponding to active features in the training instance decreased by a fraction of 1, so that the sum of all the fractions used by each class total 1. The most straight forward case is the case where each class is decreased by $\frac{1}{|E|}$. This method of updating the weight vectors has been show by Crammer and Singer [5] to converge to the optimal weight vector.

Hierarchical Multiclass Perceptron

Hierarchical multiclass perceptrons address problems where more than one level of information is available, and the upper levels should have influence on the lower levels. The ACE corpus provides such a problem with the classification of entities into both a type and a subtype, where the subtypes can be related back to a single parent type. The correct identification of the type should strongly influence the identification of the subtype.

An additional benefit can be derived from the separation between the hierarchies in cases where additional training data may be available, but only appropriate to one level of the hierarchy. Prior work on more general problems, such as previous ACE corpora that use a subset of the 2005 types and do not provide subtypes at all, could be incorporated into the development of a better solution to problems requiring more specific answers.

To exploit the relationship between levels in a hierarchical multiclass perceptron, the multiclass perceptrons in the lower levels need to know what the relationship is with the higher level. If a two level hierarchy exists, with the top level being people and vehicles and the bottom level being individuals, groups, planes, cars and boats. The bottom level needs to be aware not only of its multiclass perceptron, but also that individuals and groups are people and planes, cars and boats are vehicles.

Continuing with this example and applying hierarchical multiclass perceptrons to it, the top level is a normal multiclass perceptron. The bottom level contains a normal multiclass perceptron but will make use of both the classes and the weighted sum from the top level to aid the bottom level decision. The bottom level takes the weighted sum for each bottom level class and adds that with the weighted sum of the corresponding parent top level class. For example, the weighted sum of individual would be added with the weighted sum of people and compared with the weighted sum of car and added to the weighted sum of vehicles to determine if the bottom class was an individual or a car.

Training a hierarchical multiclass perceptron has a training phase for each level. Using a two level hierarchical multiclass perceptron, training proceeds as follows. First the top level is trained as a normal multiclass perceptron for one epoch on a training set that is specific to the top level. Once the top level has been trained, training can begin on the bottom level. If the instance is classified correctly at the bottom level, no updates on either level occur and the training proceeds to the next training instance. When an error occurs, the bottom multiclass perceptron is updated normally and the top level multiclass perceptron is notified of the error. If the top level perceptron also has an error classifying the type using the same training

instance, the top level also does an update to its weights. If the top level makes the correct prediction, then no update occurs at the top level.

Determining the Number of Training Iterations

Because of the total number of possible features in relation to the size of the training set, it was expected that any of the perceptron models would tend to overfit. While the simplest solution might be to use held out data to determine the best number of epochs to train for, a better solution seemed likely. To retain the 5 fold cross validation structure for all three phases, while still managing the problem of overfitting a twist on held out data was needed. Instead of designating a fold for held out data, and losing that data for the next phase as well as losing the ability to use it for training on the current phase, a 3 fold training, 1 fold held out, 1 fold test plan was arrived at. Executing this plan required training on all combinations of 3 folds of test, 1 fold of held out and 1 fold of test. For each test set, this would yield 4 held out results sets. Taking the best epoch for each of these 4 test sets and averaging provides an average best epoch for a given test set. This average best epoch then in theory should balance undertraining and overfitting when used to train on all four folds for a given test set. Using this methodology, although considerably more work, does not sacrifice any data to a held out set, but still provides a solution to the overfitting problem.

4.3.4 Using Google Sets to Improve Hierarchical Multiclass Perceptron

The addition of Google Sets is the first proposed addition to the baseline approach of using hierarchical multiclass perceptrons to label phrases. Google Sets is a service provided by Google that allows a user to provide a list of phrases and Google Sets will attempt to return a larger set of related phrases. The quality of the returned sets varies considerably, and for many phrases, an empty set is returned. Even with

these limitations, Google Sets is a valuable resource for expanding the number of phrases available for training. Details of how to utilize information from Google Sets are discussed below.

Google Sets Data Acquisition

The creation of the Google Sets training data was done by fold, so that each fold of the original training data had a corresponding fold of Google Sets training data. The phrases in each fold of training data were used as seed phrases to submit to Google Sets. The seed phrases submitted to Google Sets are returned by Google Sets in the return set unless the empty set is returned. Given this knowledge, if the Google Sets training data was not separated by fold, then the Google Sets training data would likely provide all the unknown phrases when performing five fold cross validation. While this would lead to impressive performance improvements, it is not good research. Thus, following the pattern of five fold cross validation where one fold is used for testing, and four folds are used for training, the Google Sets data leaves the fold that corresponds to the test fold out, and includes the other four folds in as training data.

As mentioned before, Google Sets is used to automatically generate training instances. Given the original training data, all the phrases are extracted and labeled with their corresponding subtype. The list of subtypes uniquely identifies the type, so type information is not also needed for this step. The phrase list is then pruned of all phrases that are ambiguously labeled, leaving a list of phrases that have only a single corresponding subtype. It can be seen from Figure 4.11 that removing ambiguous words from the submission list removes most of the pronouns which would not benefit from the Google Sets treatment. This list is then submitted to Google Sets one phrase at a time and the returned phrase list is assigned to the original phrase. After collecting

Word	List of types making the word ambiguous
it	30 of the 45 possible subtypes
they	26 of the 45 possible subtypes
he	Air, Commercial, Group, Indeterminate, Individual

Figure 4.11. Examples of ambiguous words not submitted to Google Sets

all the needed new phrase lists from Google Sets, each fold of the original training data is processed again. During this processing, the phrase in the original training data is used as a key to look up the corresponding phrase list returned from Google Sets. If the key exists for the lookup, then the phrase list is added to the Google Sets training data for that fold. If the key does not exist, the phrase is one of the ambiguous phrases, and nothing new is added to the Google Sets phrase list. No attempt is made to filter the new Google Sets training data for quality or ambiguity. The new Google Sets training data would probably benefit greatly from a quality filter, but we will leave this for future research.

Handling ambiguously labeled return phrases. Ambiguity could easily be handled for the resultant phrases, but ambiguous phrases in the phrases returned from Google Sets should be helpful. If Google Sets returns the same phrase for two different submitted phrases where the submitted phrases had different subtypes, that phrase will be labeled with two different subtypes in the training data, and potentially two different types as well. If this phrase happens to be an unseen phrase in the test set, the ambiguity has decreased the search space for a possible label, unless the phrase is so ambiguous that it was labeled as all the different subtypes via different phrase submission to Google Sets. By including phrases that are ambiguously labeled via this process, in the worst case, no additional information has been gained from excluding the phrases. Given the best case, the ambiguous word is unambiguous with regard

to the type and is only ambiguous between two subtypes. Such occurrences would be highly beneficial for unseen phrases, and thus there is no reason to exclude ambiguous returned values. A similar case could be made for not filtering the phrases from the original training set. The original training set was filtered as a simple way to remove the pronouns, many of which are highly ambiguous, and none of which would benefit much from Google Sets returning more pronouns. More advanced filtering methods when generating Google Sets training data by yield improved performance, but the justification for the filtering methods used in this case should be logically sound.

Building Google Sets Training Instances

Google Sets provides phrases only. This means all the features derived from the surrounding words in the sentence do not apply to the Google Sets training instances. This also makes generating Google Sets training instances very easy. During the collection phase, all the phrases were given labels, and the label is the first entry needed in the training file. The only features that apply without sentence information are the gazetteer information features, which are dictionary lookups. Since it is known what CRF rule name maps to which dictionary lookup, the training file can be generated directly with going through the transformation that the ACE data did. The phrase is submitted to all the gazetteer information functions, with false and O being discarded. Once these 14 features are added, the training instance is complete and ready for use.

Adding Google Sets Training Instances

Adding the Google Sets training data to the ACE training data only requires combining the same Google Sets folds and the ACE folds that will be used for training. Since

the two sets of training instances are in the same format, the hierarchical multiclass perceptron can be run on the combined training set without modification.

Since the Google Sets training instances are noisily labeled, it is conceivable that, to most effectively utilize Google Sets, the Google Sets data and the ACE data needed to be treated differently. Consequently, regulating control over the influence Google Sets had on the Hierarchical Multiclass Perceptron was done using a multiclass perceptron instead of just including the additional training instances with the ACE training instances. This multiclass perceptron's results were then feed into the hierarchical multiclass perceptron as a feature, allowing the Hierarchical Multiclass Perceptron to regulate the influence the Google Sets data would play. Since there exists two Google Sets training set, one for type and one for subtype, two multiclass perceptrons were created and added as a feature to the appropriate levels of the perceptron.

4.4 Phase 3: Secondary Type/Subtype Labeling - Pronoun Disambiguation

4.4.1 Overview

Pronoun disambiguation is the second proposed extension to the baseline entity detection algorithm. Pronouns differ from the other phrases being labeled in that they derive their labels from the objects they reference. Knowing this, it makes sense that pronouns should be addressed after all other phrases have been labeled, so the labels of their objects that the pronouns are actually references are available to help determine the how the pronouns themselves should be labeled.

Since the relabeling (henceforth secondary labeling) is limited to pronouns the feature set for this problem is limited in scope to the phrases in the same article. Since the number of phrases in an article is unknown, it would be ideal to have an

algorithm with the flexibility to accommodate any number of features. Maximum entropy allows for unrestricted number of features [8], and the features can be added without concern with how they will be used. The extent at which a feature contributes to a target probability is handled by the algorithm. The simplicity of use makes this algorithm an ideal choice for statistical classification problems where it is desired to focus on feature selection. The additional features used by the secondary labeler will be covered below. The secondary labeling actually replaces the label from phase 2 with the label for phase 3, without considering its value or attempting to use it as a feature for phase 3.

4.4.2 Creating Training Instances

In creation of the training instances, one training instance will be generated for each pronoun that appears in the source text. This collection of training instances will be labeled for both type and subtype information, creating two training sets, allowing for two independent models to be trained. The training instance from phase 2 is the basis from the training instance in phase 3.

4.4.3 Features

In addition to those features from phase 2, additional features will be generated based on the labels of the non-pronominal phrases from phase 2. Specifically, there are two feature types used to relate the labels from phase 2 to the pronouns. The first feature type is a same sentence feature type. For each class c , if a phrase labeled with c appears in the same sentence as the pronoun, then a feature is added to that pronoun's feature set with the feature value 1. The second feature type is based on distance. If the number of words between the nearest example of a class and the pronoun is d , and the total number of words in an article is T , the feature value is

$$\text{Feature weight} = 1 - \frac{\text{distance in words between pronoun and class}}{\text{Total words in the article}}$$

Figure 4.12. Feature weight calculation for word distance

calculated as $1 - \frac{d}{T}$. The closer a class is to the pronoun, the closer the feature value is to 1. If a class exists in the same article as a pronoun, that pronoun will have a feature for that class with the feature value calculated using the formula in Figure 4.12. Since the number of new features is dependent on the number of classes in an article, there can be 7 new class features for the distance based feature and 7 new class features for the same sentence feature, for a maximum of 14 new features for a type training instance. Using the same logic, there is a max of 90 additional features that could be added to a subtype training instance.

Phase 3 has to treat the training and the test instance generation separately. The training instances for phase 3 use the correct labels, while the test instances use the labels generated by phase 2. The output for phase 3 is a list of labels, which are in the same order as the test instances. Substituting the phase 2 result for the first column of the corresponding test instance will create a file that has the same format as the phase 2 training files, but includes the phase 2 predictions instead of the correct labels. These files can then be used to generate the phase 3 test files in the same manner as the training files were generated.

CHAPTER 5

EVALUATION

5.1 Experimental Setup

Evaluation corpus. To evaluate the performance of the proposed approach, the ACE 2005 Multilingual Training Corpus was used. The ACE corpus provides training data in English, Arabic and Chinese, but the English data will be the only data utilized for our experiment. Six different data sources provide the variety of articles in the corpus. These data sources are broadcast conversation, conversational telephone speech, broadcast news, news magazine, newsgroups and weblogs with the article distribution and word distribution shown in Table 5.1. The ACE corpus provides labeled data for both the type and subtype classifications. In Table 5.2 the break down of total phrases and total unique phrases can be seen for each type, and in Table 5.3 the same break down can be seen for the subtypes.

Evaluation methodology. As mentioned before, evaluating the performance of the approach will be done using five fold cross validation. Each fold will be created

Table 5.1. Article Distribution for the ACE 2005 Corpus

Article Class	Number of Articles	Number of words
Broadcast Conversation	52	33,874
Conversational Telephone Speech	34	34,868
Broadcast News	34	52,444
News Magazine	81	33,459
Newsgroups	37	26,371
Weblogs	114	35,529

Table 5.2. Type Statistics for the ACE 2005 Corpus

Class	Total Phrases	Total Unique Phrases
ORG	5560	992
GPE	7442	830
VEH	844	202
WEA	883	128
PER	24726	2283
LOC	1110	251
FAC	1405	282

Table 5.3. Subtype Statistics for the ACE 2005 Corpus

Class	Total Phrases	Unique Phrases	Class Class	Total Phrases	Phrases Unique
Commercial	1437	287	Media	1047	147
Plant	23	6	Bldg-Grnds	884	208
Biological	37	10	Airport	153	33
Nuclear	130	12	GPE-Cluster	111	26
Individual	15125	1823	Water-Body	92	34
Region-Int.	84	32	Special	162	19
Group	7923	649	WEAUnderspecified	207	39
State	701	131	Nation	4729	291
Med.-Sci.	48	29	Region-Gen.	568	136
Path	185	53	Sports	470	105
Shooting	181	33	Address	7	5
Continent	83	15	Subarea	31	17
Religious	36	15	County	42	18
Educational	122	41	Land	304	73
Projectile	124	35	Region-Natural	93	33
Pop.-Center	1614	318	Subarea	160	56
Government	1306	231	VEHUnderspecified	15	5
Exploding	149	37	Non-Gov	1037	225
Chemical	36	4	Sharp	12	5
Entertainment	57	26	Boundary	37	4
Air	290	72	Celestial	229	20
Water	204	40	Blunt	7	3
Indeterminate	1678	69			

$$\begin{aligned}
\text{Precision} &= \frac{|correct \cap predicted|}{|predicted|} \\
\text{Recall} &= \frac{|correct \cap predicted|}{|correct|} \\
\text{F-measure} &= \frac{2 * (Precision * Recall)}{(Precision + Recall)}
\end{aligned}$$

Figure 5.1. Precision, Recall and F-measure Formulas

by sorting each class of article by the publication date, and then evenly dividing the sorted list into the five folds.

Evaluation metric. Performance evaluation will be determined using the CoNLL evaluation script ¹. This script provides three metrics of interest, precision, recall and F-measure that are defined in Figure 5.1.

Phase evaluation. Each of the three phases will use the same methodology to evaluate performance. Each phase will use five fold cross validation with consistent folds between each phase. Performance results for each fold and the overall performance will be calculated with the CoNLL evaluation script. Maintaining this consistency will allow for comparisons to be made between test results from two different phases.

5.2 Evaluation

Overview of Experiment Setup Recall that a three phase approach is used to accomplish the type/subtype classification of the identified entities. The first phase uses CRF to identify the phrases in the corpus that need to be labeled. The second phase uses hierarchical multiclass perceptrons to label the phrases for both type and subtype. Finally, the third phase uses maximum entropy to attempt to disambiguate the pronouns by making use of features not appropriate to other phrases being labeled.

¹<http://www.cnts.ua.ac.be/conll2000/chunking/conlleva1.txt>

Table 5.4. Phase 1 Results - Phrase Identification

Fold	Precision	Recall	F-measure ($\beta = 1$)
Fold 0	92.97	87.84	90.33
Fold 1	93.34	88.22	90.71
Fold 2	92.42	87.79	90.05
Fold 3	92.71	87.04	89.78
Fold 4	92.24	88.03	90.08
Overall	92.75	87.79	90.20

5.2.1 Phase 1: Phrase Identification

The phrase identification is the first phase in identifying and classifying named entities with type and subtype information.

As can be seen in Table 5.4, the maximum F-measure is set to approximately 90.2. The later phases all operate on these phrases, and no additional method attempts to boost phrase results. Therefore, the maximum score from the latter phases have been limited by the phase 1 results.

5.2.2 Phase 2: Primary Type/Subtype Labeling

Multiclass Perceptron Results

Flat perceptrons do not generally make interesting baselines, thus the hierarchical multiclass perceptron is used as the baseline. During the experiments, flat perceptron results were collected as a check against the other results, and those results were interesting. The flat perceptron performance was better than the several of the other hierarchical multiclass perceptron experiments.

Two experiments were run for the flat multiclass perceptron, the first just trained perceptrons on the ACE training data as seen in Table 5.5. The second set of experiments was run against a combined ACE and Google Sets training data

Table 5.5. Detailed Results - Flat Multiclass Perceptron

		type			subtype		
Augmentation		precision	recall	FB1	precision	recall	FB1
none	Fold 0	75.99	71.79	73.83	70.53	66.64	68.53
	Fold 1	78.70	74.39	76.49	68.89	65.11	66.95
	Fold 2	58.37	55.44	56.86	47.18	44.82	45.97
	Fold 3	76.20	71.54	73.80	68.80	64.59	66.63
	Fold 4	76.59	73.09	74.80	66.25	63.23	64.70
	Overall	73.17	69.25	70.86	64.33	60.88	62.35
Google Sets	Fold 0	73.47	69.41	71.38	65.58	61.96	63.72
	Fold 1	72.71	68.72	70.66	67.89	64.17	65.98
	Fold 2	53.45	50.77	52.08	45.88	43.58	44.70
	Fold 3	71.18	66.83	68.94	66.22	62.17	64.13
	Fold 4	62.44	59.59	60.98	55.94	53.39	54.64
	Overall	66.65	63.06	64.80	60.30	57.05	58.52

also in Table 5.5. The performance for both the subtype and type results dropped significantly between the training data without the Google Sets data added in.

Hierarchical Multiclass Perceptron Results

Hierarchical multiclass perceptrons have been shown to improve the performance of classification tasks lower (more detailed) in the hierarchy by utilizing both the high level classification results as well as additional training data [1]. Comparing Table 5.5 and Table 5.6, two interesting things are worth noting. Hierarchical multiclass perceptrons had a small but noticeable improvement over flat multiclass perceptrons. This was the expected and desired results of this experiment. On the other hand, the Google Sets experiment not only did worse than the unaugmented training set, but it actually performed worse than Google Sets with flat multiclass perceptrons.

Why was performance decreasing? The first thing to consider was the Google Sets data. The data is quite dirty, but the perceptron should filter the one offs fairly well by giving them every little weight. The data set is also very large. The data set is

approximately 50,000 unique phrases compared to ACE's approximately 6,000 unique phrases. Both training sets, taken separately, have repeats in phrases. The Google Sets training set has a repetition factor of about 8, while ACE has a repetition factor of 30, with repetition factor being defined as the average number of times the same phrase is seen across the entire corpus. In terms of total number of training phrases, the Google Sets training data is about 2.5 times as large as the ACE data. The number of words cannot be compared, because the ACE corpus is made up complete or partial articles or their equivalent. The Google Sets training data only includes phrases. This final difference means that the features that appear in the Google Sets data are going to be over represented compared to features based on surrounding words that are only available in the ACE corpus.

Given these differences, it seems likely that the Google Sets data is over powering the ACE data. It could still be a possibility that the Google Sets data is worthless as training data for this problem. If that were the case, why would it be in the thesis title?

Improved Hierarchical Multiclass Perceptron Results

If the Google Sets data is over powering the ACE data, and the Google Sets data does have value to contribute to the classification of types and subtypes, there should be a solution to scale back the Google Sets data in relation to the ACE data. There are many possibilities to accomplish this, but an experimentally verified solution is taking the Google Sets data and training a flat multiclass perceptron on that training set and then using that perceptron as an additional feature into the hierarchical multiclass perceptron. This solution separates the two feature sets, preventing Google Sets specific features from over power the ACE features when calculating the classifications, and it additionally adds a weight to each of the classifications coming out of

Table 5.6. Detailed Results - Hierarchical Multiclass Perceptron

		type			subtype		
Augmentation		precision	recall	FB1	precision	recall	FB1
none	Fold 0	77.94	70.23	73.88	68.31	63.15	65.63
	Fold 1	79.99	71.99	75.78	70.79	65.63	68.12
	Fold 2	64.93	58.48	61.54	53.67	51.26	52.44
	Fold 3	78.17	69.67	73.68	66.89	61.24	63.94
	Fold 4	76.52	68.81	72.46	64.25	60.71	62.43
	Overall	75.71	67.83	71.30	64.72	60.40	62.35
Google Sets	Fold 0	68.55	64.76	66.60	63.22	59.72	61.42
	Fold 1	69.79	65.97	67.82	65.39	61.81	63.55
	Fold 2	55.54	52.76	54.11	46.63	44.29	45.43
	Fold 3	68.57	64.38	66.41	62.14	58.34	60.18
	Fold 4	65.84	62.84	64.30	58.94	56.25	57.56
	Overall	65.66	62.14	63.71	59.26	56.10	57.48

the Google Sets perceptron. Since the Google Sets data has no quality control, these weights can act as a sort of quality filter.

As can be seen from Table 5.7 and more easily from Table 5.9, the Google Sets data does add value to the test performance.

Since the perceptron is not really part of the hierarchy of the hierarchical multiclass perceptron, updates were not attempted inline. Thus, determining the number of iterations to train the Google Sets perceptron requires another round of held out runs.

5.2.3 Phase 3: Secondary Type/Subtype Labeling

Results of phase 3 are shown in Table 5.8. In comparison to the phase 2 results (see Table 5.7), the overall performance improves. In addition, it is worth noting that this pronoun disambiguation method works very well for written text, but not well for transcribed spoken text with speaker tags identifying the speakers. Specifically,

Table 5.7. Detailed Results - Improved Hierarchical Multiclass Perceptron

		type			subtype		
Augmentation		precision	recall	FB1	precision	recall	FB1
Google Sets ²	Fold 0	78.92	74.56	76.68	70.77	66.86	68.76
	Fold 1	80.69	76.27	78.42	72.91	68.91	70.85
	Fold 2	63.67	60.47	62.03	50.10	47.59	48.81
	Fold 3	77.59	72.84	75.14	72.46	68.02	70.17
	Fold 4	74.69	71.28	72.94	68.06	64.96	66.47
	Overall	74.98	70.97	72.92	66.54	62.99	64.71

Table 5.8. Detailed Results - Pronoun Disambiguation Applied to Improved Hierarchical Multiclass Perceptron

		type			subtype		
Augmentation		precision	recall	FB1	precision	recall	FB1
Google Sets ³	Fold 0	78.92	74.56	76.68	69.59	66.00	67.75
	Fold 1	82.10	78.31	80.16	74.75	70.84	72.74
	Fold 2	66.13	64.12	65.11	55.93	53.70	54.79
	Fold 3	77.90	74.63	76.23	71.65	68.49	70.04
	Fold 4	75.18	73.33	74.24	61.73	60.37	61.04
	Overall	75.90	72.84	74.33	66.58	63.74	65.12

the ACE corpus includes several of these transcribed spoken text files, and by the method that the folds were created, all the transcribed files ended up in the same fold, fold4. These files include tags that identify the speaker, but the speakers are not labeled. When attempting to find the nearest label to analyze, there actual speaker information is not available, and thus no information to indicate if the PER is an Individual, Group or Indeterminate. As can be seen from the results, this lack of information did not prevent the type score from improving, but with such a large concentration of transcribed files making up this fold, the subtype score was hurt considerably.

Table 5.9. Summary of Results

		type FB1	subtype FB1
Flat	none	70.86	62.35
	Google Sets	64.80	58.52
Hierarchical	none	71.30	62.42
	Google Sets	63.71	57.48
Improved Hierarchical	Google Sets	72.92	64.71
Pronouns Disambiguation	Google Sets	74.33	65.12

5.3 Overall Error Analysis

5.3.1 Fold 2 Error Analysis

When referring to any fold based performance tables, it is difficult not to notice that fold 2 underperforms all other folds after phrase finding. This difference in performance is caused by two characteristics of fold2. First, the fold0 has the second highest number of pronouns, which brings down the score, but this only makes up for a small part of the dip. The larger portion stems from the fact that the pronoun distribution in the other folds is more heavily weighted in personal pronouns, which tend to be easier to classify. To summarize, fold 2 has more of the pronouns that are the most difficult to classify than the other folds.

5.3.2 Type Error Analysis

The PER type had the highest success, as seen in Table 5.11, which can be attributed to its large number of examples in the corpus, signified by the Total column in Table 5.10. Performance drops off considerably between PER and GPE, which can be attributed to the drop of in training examples. The remaining types, except WEA, follow the same pattern, with training examples dropping off, the percent of unseen phrases in the test set increasing and performance decreasing. WEA is a good ex-

Table 5.10. Analyzing Type Performance in Relation to Frequency/Phrase and Unseen Phrases

Class	Total	Unique Phrases	frequency/phrase	Unseen count	% Unseen
PER	24726	2283	10.8305	2454	9.92%
GPE	7442	830	8.9663	684	9.19%
ORG	5560	992	5.6048	1025	18.44%
FAC	1405	282	4.9823	303	21.57%
LOC	1110	251	4.4223	227	20.45%
WEA	883	128	6.8984	130	14.72%
VEH	844	202	4.1782	195	23.10%

Table 5.11. Overall Type Performance using Improved Hierarchical Multiclass Perceptrons and Pronoun Disambiguation

	Precision	Recall	$F_{\beta=1}$
PER	82.97%	84.53%	83.74
GPE	65.59%	74.67%	69.84
WEA	66.86%	53.00%	59.13
ORG	62.21%	43.15%	50.96
VEH	83.05%	34.24%	48.49
LOC	56.36%	41.53%	47.82
FAC	55.26%	35.16%	42.98
Overall	75.90%	72.84%	74.33

ample of how important decreasing unseen test phrases are. Even with WEA small training set, performance was significantly higher than other types with more training instances that still suffered from higher unseen phrase rates.

5.3.3 Subtype Error Analysis

A detailed analysis of each subtype is not worthwhile. Reviewing Table 5.12 and Table 5.13 the analysis of Types is further exemplified. It is clear that a high unseen rate leads to a low F-measure, and the approach used to automatically generate training instances is well founded. This approach needs to be taken further too either provide

Table 5.12. Analyzing Subtype Performance in Relation to Phrase Frequency and Unseen Phrase Rate

Class	Avg Freq	% Unseen	Class	Avg Freq	% Unseen
Commercial	5.0070	23.52%	Media	7.1224	16.52%
Plant	3.8333	21.74%	Bldg-Grnds	4.2500	20.59%
Biological	3.7000	27.03%	Airport	4.6364	24.18%
Nuclear	10.8333	6.92%	GPE-Cluster	4.2692	26.13%
Individual	8.2968	12.38%	Water-Body	2.7059	41.30%
Region-Int.	2.6250	36.90%	Special	8.5263	11.11%
Group	12.2080	8.32%	WEAUnderspecified	5.3077	21.26%
State	5.3511	15.83%	Nation	16.2509	4.86%
Med-Sci	1.6552	64.58%	Region-Gen	4.1765	24.12%
Path	3.4906	31.89%	Sports	4.4762	24.68%
Shooting	5.4848	17.13%	Address	1.4000	100.00%
Continent	5.5333	16.87%	Subarea	1.8235	58.06%
Religious	2.4000	47.22%	County	2.3333	64.29%
Educational	2.9756	45.08%	Land	4.1644	23.68%
Projectile	3.5429	30.65%	Region-Natural	2.8182	44.09%
Pop.-Center	5.0755	21.19%	Subarea	2.8571	38.12%
Government	5.6537	15.24%	VEHUnderspecified	3.0000	20.00%
Exploding	4.0270	23.49%	Non-Gov	4.6089	20.73%
Chemical	9.0000	11.11%	Sharp	2.4000	33.33%
Entertainment	2.1923	52.63%	Boundary	9.2500	10.81%
Air	4.0278	27.59%	Celestial	11.4500	6.55%
Water	5.1000	21.08%	Blunt	2.3333	42.86%
Indeterminate	24.3188	3.58%			

greater coverage or high quality test instances.

Table 5.13. Overall Subtype Performance using Improved Hierarchical Multiclass Perceptrons and Pronoun Disambiguation

	$F_{\beta=1}$		$F_{\beta=1}$
Commercial	38.38	Media	42.52
Plant	34.48	Bldg-Grounds	37.57
Biological	45.83	Airport	47.75
Nuclear	43.04	GPE-Cluster	29.70
Individual	83.47	Water-Body	46.04
Region-Int	0.00	Special	60.31
Group	63.95	WEAUnderspecified	36.78
State	50.76	Nation	69.88
Med-Sci	14.04	Region-Gen	35.01
Path	35.04	Sports	16.13
Shooting	51.69	Address	0.00
Continent	51.13	Subarea	23.65
Religious	36.36	County	9.92
Educational	38.30	Land	41.95
Projectile	45.45	Region-Natural	17.27
Pop.-Center	54.02	Subarea	31.58
Government	43.15	VEHUnderspecified	25.81
Exploding	38.05	Non-Gov	33.94
Chemical	48.00	Sharp	14.29
Entertainment	0.00	Boundary	79.37
Air	47.69	Celestial	70.87
Water	54.00	Blunt	0.00
Indeterminate	60.86		
Overall	66.58%	63.74%	65.12

CHAPTER 6

CONCLUSION

Two methods were developed that enhance the performance of hierarchical multiclass perceptrons when solving the classification task of identifying and labeling entities with their type and subtype. Utilizing Google Sets to generate new training data is an approach that hopefully merits further research.

The first proposed extension takes advantage of automatically generated training instances using Google Sets as a data source. The key to making use of such uncontrolled source data was to limit its influence on the system. The initial results using Google Sets to automatically generate training instances were promising and further study should yield considerable improvements.

The second extension, pronoun disambiguation, used the surrounding labels to reclassify the pronouns and was a quick and effective way to produce noticeably improved results. In addition to the results already achieved, there are some obvious future enhancements that might further improve performance, or at least give the algorithm visibility to its own weaknesses. Tag information from the original articles was not included as a feature, but that feature would likely have been of assistance in the pronoun disambiguation. Article type might also make a useful feature for future enhancements, but with the consolidation of the transcribed files to a single fold, this feature might be unexploitable.

In the end, the approaches worked well together, and as the performance of the primary labeler improves; the performance of the secondary labeler, for pronoun disambiguation, should also improve. Until the primary labeler is extremely good, and

can successfully handle pronouns, it is likely that the post processing step of pronoun disambiguation will continue to have value. As the primary labeler's performance increases, the pronoun disambiguation has more and more accurate labels to work with for the surrounding phrases, which should continue to improve performance.

REFERENCES

- [1] Massimiliano Ciaramita, Thomas Hofmann, and Mark Johnson. Hierarchical semantic classification: Word sense disambiguation with world. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, 2003.
- [2] Daniel M. Bikel, Richard L. Schwartz, and Ralph M. Weischedel. An algorithm that learns what's in a name. *Machine Learning*, 34(1-3):211–231, 1999.
- [3] Daniel M. Bikel, Scott Miller, Richard Schwartz, and Ralph Weischedel. Nymble: a high-performance learning name-finder. In *Proceedings of the fifth conference on Applied natural language processing*, pages 194–201, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.
- [4] R Florian, H Hassan, A Ittycheriah, H Jing, N Kambhatla, X Luo, N Nicolov, and S Roukos. A statistical model for multilingual entity detection and tracking. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 1–8, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.
- [5] Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. In *14th Annual Conference on Computational Learning Theory, COLT 2001 and 5th European Conference on Computational Learning Theory, EuroCOLT 2001, Amsterdam, The Netherlands, July 2001, Proceedings*, volume 2111, pages 99–115. Springer, Berlin, 2001.
- [6] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.
- [7] Hanna M. Wallach. Conditional random fields: An introduction, 2004.
- [8] A. Ratnaparkhi. A simple introduction to maximum entropy models for natural language processing. Technical report, Institute for Research in Cognitive Science, University of Pennsylvania, 1997.
- [9] Frank Rosenblatt. *Principles of Neurodynamics*. Spartan Books, New York, 1962.

- [10] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, Mass., 1969.
- [11] Adam L. Berger, Stephen Della Pietra, and Vincent J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

VITA

Michael Quinn was born in Hurst, Texas on March 14th, 1977. Michael spent his formative years playing in the dirt with construction toys, but changed his focus from dirt to water in his mid-teens, and spent his high school years swimming endless laps, staring at the bottom of the pool. It was during these 5 second breaks between laps that Michael got to know his much later to be wife, Brook. Also at this time Michael developed, in his water logged brain, the idea to go to school at Carnegie Mellon University in Pittsburgh and majoring in Electrical and Computer Engineering. Two years later, after much enjoyment of the Pittsburgh weather and the exciting lifestyle of an impoverished student, CMU was ditched, and Michael left his beloved school-work for a real job. Out in the real world, working at different real jobs, eating food that required more than 10 cents and hot water to make, people would occasionally ask Michael if he had a piece of paper. During this same time, his girlfriend, then fiancée, then wife managed to make a transition from Ms. to Dr. It was obvious that Michael didn't have a piece of paper, and maybe he should get one or two of his own. In August of 2005, a five year plan was begun that would hopefully end in Michael's first piece of paper. In August of 2007 Michael received his first piece of paper in the mail. In April of 2008 he needed to write a vita.

Permanent address: 811 Pebble Ridge Dr.
Lewisville, Texas 75067
U.S.A.