Improving Scalability of Support Vector Machines for Biomedical Named Entity Recognition

by

Mona Soliman Habib
B.Sc., Ain Shams University, 1981
M.Sc., Colorado Technical University, 2001

A dissertation submitted to the Graduate Faculty of the University of Colorado at Colorado Springs in partial fulfillment of the requirements for the degree of Doctor of Philosophy
Department of Computer Science
2008
This dissertation for
Doctor of Philosophy Degree by

Mona Soliman Habib

has been approved for the
Department of Computer Science

by

Dr. Jugal K. Kalita, Chair

Dr. Albert T. Chamillard

Dr. Edward C. Chow

Dr. Lisa Hines

Dr. Aleksander Kolcz

Dr. Xiaobo Zhou

Date
This research work aims to explore the scalability problems associated with solving the named entity recognition problem using high-dimensional input space and Support Vector Machines (SVM), and to provide solutions that lower the computational time and memory requirements. The named entity recognition approach used is one that fosters language and domain independence by eliminating the use of prior language and/or domain knowledge. The biomedical domain is chosen as the focus of this work due to its importance and inherent challenges.

To investigate the scalability of the SVM machine learning approach, we conduct a series of baseline experiments using biomedical abstracts. Single class and multi-class classification performance and results are examined. The initial performance results – measured in terms of precision, recall, and F-score – are comparable to those obtained using more complex approaches. These results demonstrate that the simplified architecture used is capable of providing a reasonable solution to the language and domain-independent named entity recognition problem.

To improve the scalability of named entity recognition using SVM, two main factors are tackled: computation time and memory requirements. We develop a new cutting plane algorithm for multi-class classification – \textit{SVM-PerfMulti} – based on a new structural formulation for support vector optimization. The solution is several orders of magnitude faster than the traditional \textit{SVM-Multiclass} machine and provides good out-of-
the-box performance, yet it requires a considerable amount of memory that is linear w.r.t.
the training data size.

We introduce a new database-supported solution embedded in a relational database
server. The database framework supports both the binary and multi-class classification
problems. We explore the trade-off between computation time and online memory
requirement when input feature vectors and trained support vectors are stored and fetched
from the database. Database caching of input vectors has minimal to no impact on the
computation time. On the other hand, constantly fetching support vectors from permanent
storage negatively impacts computation time and can be improved by caching kernel
computations in active memory. The database framework promotes the usability of the
machine learning solution and provides a way to build a growing dictionary of previously
identified named entities.
To the Memory of my Beloved Father

To my Children, Marise and Pierre, the Joy of my Life
Acknowledgments

First, I would like to acknowledge my late father in a special way as this dissertation work nears its end. Dad, this is for you … I miss you very much and I hope that I was able to fulfill your last will despite the many years that went by since you left our world. Without the unconditional love and strong belief in me that you and my mother have constantly given me, I would not have achieved anything in life.

I would like to express my gratitude to my advisor, Dr. Jugal Kalita, for his valuable advice and all of the help and support that he has provided me throughout this entire research process. His continuous encouragement and trust, even when I doubted myself, kept me going and motivated me to complete this dissertation.

My thanks also goes to my advisory committee members: Dr. Tim Chamillard, Dr. Edward Chow, Dr. Xiaobo Zhou, Dr. Lisa Hines, and Dr. Alek Kolcz, for their valuable feedback and directions to make this dissertation possible. Special thanks to Dr. Tim Chamillard for dedicating one of his research machines for my use to run the many experiments needed to complete this work. I am also grateful to Mr. Jim Martin for his support with the research machines, and to Ms. Patricia Rea for her prompt response to many inquiries and her constant willingness to help.

I would also like to thank the following individuals for providing software and tools that proved invaluable in carrying out this work: Dr. Thorsten Joachims at Cornell University for giving me the permission to use SVM-Struct as a basis for my work, as well as his SVM-Light, SVM-Perf, and SVM-Multiclass implementations for experimentation; Mr. Claudio Giuliano at the Dot.Kom Project in Italy for allowing me to use the JFex feature extraction software and providing the initial extraction scripts; and
Mr. Stephen Senator at the US Air Force Academy for giving me access to the USAFA research clusters during the early stages of this research endeavor.

Last, but not least, I extend my warmest expression of gratefulness and love to my family – my husband, Reda, and my children, Marise and Pierre – for believing in me and for standing by me in the midst of all the ups and downs I experienced throughout the research process. I would also like to thank my brother, Aziz, and his family for their love and encouragement. My special gratitude goes to my extended family and friends for their thoughts and prayers. Finally, thanks be to the Lord for getting me to this day and for giving me the strength to accomplish this work.
# Table of Contents

<table>
<thead>
<tr>
<th>Chapter 1</th>
<th>Introduction .................................................................................................................. 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.1 Motivation ........................................................................................................ 1</td>
</tr>
<tr>
<td></td>
<td>1.2 Research Methodology ....................................................................................... 3</td>
</tr>
<tr>
<td></td>
<td>1.3 Dissertation Outline ....................................................................................... 6</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>Named Entity Recognition ....................................................................................... 9</td>
</tr>
<tr>
<td></td>
<td>2.1 Named Entity Recognition Approaches ................................................................ 10</td>
</tr>
<tr>
<td></td>
<td>2.2 Common Machine Learning Architecture .......................................................... 12</td>
</tr>
<tr>
<td></td>
<td>2.3 Performance Measures ....................................................................................... 15</td>
</tr>
<tr>
<td></td>
<td>2.4 Biomedical Named Entity Recognition ............................................................. 16</td>
</tr>
<tr>
<td></td>
<td>2.5 Language-Independent NER ................................................................................. 21</td>
</tr>
<tr>
<td></td>
<td>2.6 Named Entity Recognition Challenges ................................................................ 29</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>Support Vector Machines ....................................................................................... 31</td>
</tr>
<tr>
<td></td>
<td>3.1 Support Vector Machines ................................................................................... 31</td>
</tr>
<tr>
<td></td>
<td>3.2 Binary Support Vector Classification ................................................................ 32</td>
</tr>
<tr>
<td></td>
<td>3.3 Multi-class Support Vector Classification ....................................................... 39</td>
</tr>
<tr>
<td></td>
<td>3.4 Named Entity Recognition Using SVM ............................................................... 43</td>
</tr>
<tr>
<td></td>
<td>3.5 SVM Scalability Challenges ............................................................................... 44</td>
</tr>
<tr>
<td></td>
<td>3.6 Emerging SVM Techniques .................................................................................. 45</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>Identifying Challenges in Large Scale NER ............................................................. 47</td>
</tr>
<tr>
<td></td>
<td>4.1 Baseline Experiment Design ............................................................................... 48</td>
</tr>
<tr>
<td></td>
<td>4.2 Features Selection ............................................................................................... 50</td>
</tr>
<tr>
<td></td>
<td>4.3 Single Class Results ......................................................................................... 52</td>
</tr>
<tr>
<td>Chapter</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>4.4</td>
<td>Multi-class Results</td>
</tr>
<tr>
<td>4.5</td>
<td>Challenges and Problems</td>
</tr>
<tr>
<td>4.6</td>
<td>NER/SVM Scalability Challenges</td>
</tr>
<tr>
<td>4.7</td>
<td>SVM Usability Issues</td>
</tr>
<tr>
<td>5</td>
<td>Improving Multi-Class SVM Training</td>
</tr>
<tr>
<td>5.1</td>
<td>Basic All-Together Multi-Class SVM</td>
</tr>
<tr>
<td>5.2</td>
<td>Improving SVM Training Time</td>
</tr>
<tr>
<td>5.3</td>
<td>$SVM-PerfMulti$: New Multi-Class Instantiation</td>
</tr>
<tr>
<td>5.4</td>
<td>Reducing Memory Requirements</td>
</tr>
<tr>
<td>6</td>
<td>Database Framework for SVM Learning</td>
</tr>
<tr>
<td>6.1</td>
<td>SVM Database Architecture</td>
</tr>
<tr>
<td>6.2</td>
<td>$SVM-MultiDB$ Database Schema</td>
</tr>
<tr>
<td>6.3</td>
<td>Embedded Database Functions</td>
</tr>
<tr>
<td>6.4</td>
<td>$SVM-MultiDB$ Usage Example</td>
</tr>
<tr>
<td>6.5</td>
<td>Improved SVM Solution Usability</td>
</tr>
<tr>
<td>6.6</td>
<td>Tradeoff of Training Time vs. Online Memory Needs</td>
</tr>
<tr>
<td>7</td>
<td>Single-Class Experiments</td>
</tr>
<tr>
<td>7.1</td>
<td>Single-Class Scalability Results</td>
</tr>
<tr>
<td>7.2</td>
<td>Effect of Regularization Factor $C$</td>
</tr>
<tr>
<td>7.3</td>
<td>Comparison of $SVM-Perf$ and $SVM-PerfDB$</td>
</tr>
<tr>
<td>8</td>
<td>Multi-Class Experiments</td>
</tr>
<tr>
<td>8.1</td>
<td>Multi-Class Scalability Results</td>
</tr>
<tr>
<td>8.2</td>
<td>Multi-Class Performance Measures</td>
</tr>
</tbody>
</table>
List of Tables

Table 2.1 - Overview of BioNLP Methods, Features, And External Resources .......... 26
Table 2.2 – Performance Comparison of Systems Using BioNLP Datasets ................. 26
Table 2.3 – Comparison of Systems Using the CoNLL-02 Data ................................. 27
Table 2.4 – Comparison of Systems Using the CoNLL-03 Data ................................. 28
Table 4.1 – Effect of Positive Example Boosting....................................................... 53
Table 4.2 – Performance of BioNLP Systems Using SVM vs. Baseline Results.......... 56
Table 4.3 – Effect of Positive Examples Boosting on Single-Class SVM Results....... 56
Table 4.4 – Summary of Baseline Multi-Class Experiment Results ............................ 57
Table 4.5 – Baseline Multi-Class Experiment Results 1978-1989 Set ............................ 57
Table 4.6 – Baseline Multi-Class Experiment Results 1990-1999 Set ............................ 57
Table 4.7 – Baseline Multi-Class Experiment Results 2000-2001 Set ............................ 57
Table 4.8 – Baseline Multi-Class Experiment Results 1998-2001 Set ............................ 57
Table 4.9 – Single and Multi-class Training Times .................................................... 59
Table 7.1 - Consistent Performance Change with Varying C-Factor .......................... 108
Table 8.1 – Summary of Dataset Contents ................................................................. 114
Table 8.2 - Comparison of $SVM$-$PerfMulti$, $SVM$-$MultiDB$, and $SVM$-$Multiclass$ Training Time vs. Training Data Size .................................................. 119
Table 8.3 – Performance of BioNLP Systems Using SVM vs. $SVM$-$PerfMulti$ Results 124
Table 8.4 – Summary of $SVM$-$PerfMulti$ Experiment Results ................................. 124
Table 8.5 – $SVM$-$PerfMulti$ Results 1978-1989 Set ................................................. 124
Table 8.6 – $SVM$-$PerfMulti$ Results 1990-1999 Set ................................................. 124
Table 8.7 – SVM-PerfMulti Results 2000-2001 Set ........................................................ 125
Table 8.8 – SVM-PerfMulti Results 1998-2001 Set ........................................................ 125
Table 8.9 – SVM-PerfMulti Overall Results .................................................................. 125
Table 8.10 – CoNLL-02 SVM-Multiclass Spanish Performance Results ....................... 128
Table 8.11 – CoNLL-02 SVM-PerfMulti Spanish Performance Results ....................... 129
Table 8.12 – CoNLL-02 SVM-Multiclass Dutch Performance Results ......................... 131
Table 8.13 – CoNLL-02 SVM-PerfMulti Dutch Performance Results .......................... 131
Table 8.14 – CoNLL-02 SVM-PerfMulti Dutch Performance Results ........................... 132
Table 8.15 – Other Experimental Datasets ..................................................................... 133
Table 9.1 – Biomedical JNLPBA-04 Performance Measures Comparison .................... 139
Table 9.2 – Spanish CoNLL-02 Performance Measures Comparison............................ 139
Table 9.3 – Dutch CoNLL-02 Performance Measures Comparison ............................... 139
Table B.1 – Basic Statistics for the JNLPBA-04 Data Sets ............................................. 163
Table B.2 – Absolute and Relative Frequencies for Named Entities Within Each Set .. 163
Table B.3 – Number of Named Entities in the CoNLL-02 Dataset ................................ 164
Table B.4 – Basic Statistics for the CoNLL-03 Dataset ............................................... 166
Table B.5 – Number of Named Entities in the CoNLL-03 Dataset ............................... 166
Table C.1 – The Current GENIA Ontology .................................................................... 168
List of Figures

Figure 2.1 - Commonly Used Architecture ................................................................. 15

Figure 3.1 – SVM Linearly Separable Case ............................................................... 33

Figure 3.2 – SVM Non-Linearly Separable Case ....................................................... 36

Figure 3.3 – SVM Mapping to Higher Dimension Feature Space .............................. 37

Figure 3.4 – Comparison of Multi-Class Boundaries ............................................... 39

Figure 3.5 – Half-Against-Half Multi-Class SVM ...................................................... 42

Figure 4.1 – Baseline Experiments Architecture ....................................................... 49

Figure 5.1 – Multi-Class SVM Error Representation ............................................... 69

Figure 6.1 – Embedded Database Architecture ........................................................ 87

Figure 6.2 - SVM Learning Data Model ........................................................................ 90

Figure 6.3 - SVM-Perf and SVM-PerfMulti Memory Usage vs. Data Size ................... 97

Figure 6.4 - SVM-PerfDB Training Time vs. Examples Cache Size ............................. 98

Figure 6.5 - SVM-MultiDB Training Size vs. Examples Cache Size ............................. 99

Figure 7.1 - SVM-Light, SVM-Perf, and SVM-PerfDB Training Time vs. Training Data Size ......................................................................................................................... 105

Figure 7.2 - SVM-Light Number of Support Vectors vs. Training Data Size ............. 106

Figure 7.3 - SVM-Perf Number of Support Vectors vs. Training Data Size .............. 106

Figure 7.4 - SVM-Perf Training Time vs. Variation of C-Factor ................................... 107

Figure 7.5 - SVM-Perf Performance vs. Regularization Factor C .............................. 109

Figure 7.6 – Single Class Protein Performance (Recall/Precision/F-Score) – C=1.0 .... 110

Figure 7.7 – Modified SVM-Perf Memory vs. Varying C ........................................ 111
Figure 7.8 – Modified SVM-Perf vs. SVM-PerfDB Training Time: $C=0.01$ .................. 112
Figure 7.9 – Modified SVM-Perf vs. SVM-PerfDB Training Time: $C=1.0$ ................. 112
Figure 7.10 – Original SVM-Perf vs. SVM-PerfDB Training Time: $C=0.01$ .......... 113
Figure 7.11 – Original SVM-Perf vs. SVM-PerfDB Training Time: $C=1.0$ .......... 113
Figure 8.1 - SVM-Multiclass Number of Support Vectors vs. Training Data Size ........ 118
Figure 8.2 - SVM-PerfMulti Number of Support Vectors vs. Training Data Size ......... 118
Figure 8.3 - SVM-Multiclass, SVM-PerfMulti, and SVM-MultiDB Training Time vs. Training Data Size ................................................................................................................. 120
Figure 8.4 - SVM-MultiDB and SVM-PerfMulti Training Time vs. Training Data Size 120
Figure 8.5 – Comparison of Single-Class and Multi-Class Memory Usage vs. Training Data Size ............................................................................................................................................. 122
Figure 8.6 - SVM-PerfMulti Overall Performance vs. Training Data Size ................. 123
Figure 8.7 - SVM-PerfMulti Protein Performance vs. Training Data Size ................. 123
Figure 8.8 – SVM-PerfMulti Training Time vs. Acceleration Factor ......................... 127
Figure 8.9 – SVM-PerfMulti Overall Performance vs. Acceleration Factor ............... 127
Figure 8.10 – SVM-PerfMulti Protein Performance vs. Acceleration Factor ............... 127
Figure 8.11 – CoNLL-02 Spanish SVM-PerfMulti vs. SVM-Multiclass Training Time 129
Figure 8.12 – CoNLL-02 Spanish SVM-PerfMulti Overall Performance .................. 129
Figure 8.13 – CoNLL-02 Dutch SVM-PerfMulti vs. SVM-Multiclass Training Time ... 130
Figure 8.14 – CoNLL-02 Dutch SVM-PerfMulti Overall Performance .................... 131
Figure 9.1 – Biomedical JNLPBA-04 Training Time Comparison .............................. 136
Figure 9.2 – Biomedical JNLPBA-04 Memory Comparison ................................. 136
Figure 9.3 – Spanish CoNLL-02 Training Time Comparison ................................. 137
Figure 9.4 – Spanish CoNLL-02 Memory Comparison ................................................... 137
Figure 9.5 – Dutch CoNLL-02 Training Time Comparison ........................................... 137
Figure 9.6 – Dutch CoNLL-02 Memory Comparison .................................................... 138
Figure 9.7 – Effect of Regularization Parameter $C$ on Training Time ....................... 140
Figure 9.8 – Effect of Regularization Parameter $C$ on Performance ......................... 141
Figure 10.1 – Recommended Service-Oriented Architecture .................................... 152
Chapter 1
Introduction

This research work aims to explore the scalability problems associated with solving the Named Entity Recognition (NER) problem using high-dimensional input space and Support Vector Machines (SVM), and to present two implementations addressing these issues. The NER domain chosen as the focus of this work is the biomedical publications domain, especially selected due to its importance and inherent challenges. In the following sections, we express our motivation for this work and briefly describe the research methodology, followed by an outline of the dissertation contents.

1.1 Motivation

Named entity recognition (NER) is one of the important tasks in information extraction, which involves the identification and classification of words or sequences of words denoting a concept or entity. Examples of such information units are names of persons, organizations, or locations in the general context of newswires. Domain-specific named entities are those terms or phrases that denote concepts relevant to one particular domain. For example, protein and gene names are named entities which are of interest to the domain of molecular biology and medicine. The massive growth of textual information available in the literature and on the Web necessitates the automation of identification and management of named entities in text.

The task of identifying named entities in a particular language is often accomplished by incorporating knowledge about the language taxonomy in the method used. In the
English language, such knowledge may include capitalization of proper names, known titles, common prefixes or suffixes, part of speech tagging, and/or identification of noun phrases in text. Techniques that rely on language-specific knowledge may not be suitable for porting to other languages. With the extension of named entity recognition to new information areas, the task of identifying meaningful entities has become more complex as categories are more specific to a given domain. NER solutions that achieve a high level of accuracy in some language or domain may perform much poorly in a different context.

Different approaches are used for carry out the identification and classification of entities. Statistical, probabilistic, rule-based, memory-based, and machine learning methods are developed. The extension of NER to specialized domains raise the importance of devising solutions that require less human intervention in the annotation of examples or the development of specific rules. Machine learning techniques are therefore experiencing an increased adoption and much research activity is taking place in order to make such solutions more feasible. Support Vector Machine (SVM) is rapidly emerging as a promising pattern recognition methodology due to its generalization capability and its ability to handle high-dimensional input. However, SVM is known to suffer from slow training especially with large input data size.

In this research work, we explore a solution that eliminates language and domain-specific knowledge from the named entity recognition process when applied to the English biomedical entity recognition task, as a baseline for other languages and domains. The biomedical field NER remains a challenging task due to growing nomenclature, ambiguity in the left boundary of entities caused by descriptive naming,
difficulty of manually annotating large sets of training data, strong overlap among
different entities, to cite a few of the NER challenges in this domain. The proposed
solution capitalizes on SVM’s ability to handle high-dimensional input space. In this
dissertation, we explore the scalability issues for named entity recognition using high-
dimensional features and support vector machines and investigate ways to address them.

1.2 Research Methodology

We propose an integrated machine learning NER solution using support vector
machines that attempts to improve the scalability of NER/SVM with large datasets. This
research will focus on All-Together multi-class training algorithms for their superior
performance and challenging scalability. This multi-class approach builds one SVM
model that maximizes all separating hyperplanes at the same time, which is a challenging
optimization problem.

We envision an SVM solution assisted by a special database schema and embedded
database modules. The database schema design incorporates input data, evolving training
model(s), pre-computed kernel outputs and dot products, and classification output data. In
order to reduce the communication overhead with the database backend, we propose
extending the database server with embedded modules. This should also provide a better
integration of all components. Database triggers may be used for frequently updated
fields to improve the potential parallelization of the learning processes.

Building a growing list of previously identified and annotated named entities will be
made possible by the database repository, which would provide a valuable resource to
constantly improve the classification performance. The evolving gazetteer list can be
used during preprocessing or post-processing to annotate and/or correct the classification of newly discovered named entities thereby boosting the overall performance. While this extension will not be implemented as part of this work, it can be easily incorporated at a future time.

Following an investigation and literature review stage, this work proceeds in three phases with different objectives:

- The first phase addresses the named entity recognition problem using a language and domain-independent approach. It focuses on the feature selection criteria and the basic architecture design that eliminates language or domain-specific pre- and post-processing. This phase also assesses the baseline performance using multi-class support vector machines, and the potential challenges and problems using this machine learning solution.

- The second phase aims to tackle the slow training time of multi-class support vector machines and to develop algorithm(s) to accelerate learning as this issue constitutes the main obstacle when dealing with large real-world datasets. A secondary objective is to identify ways to lower the memory required by the potential solution especially with large input sizes.

- The objective of the final phase is to design and implement a database framework for SVM learning that promotes the usability of machine learning and integrates training and testing example datasets, trained models, and classification outputs in a common environment. The database framework is to incorporate both the binary and multi-class support vector machines and provide complementary solutions to lower online memory needs.
During phase one of this research work, we realized the need for a novel machine learning framework that promotes reusability, expandability, and maintainability of the solution and provides an architecture that encourages future work. Initial thoughts about a service-oriented architecture for machine learning are presented in Chapter 10. Given the scope and challenge level of the proposed research plan, we decided to focus on tackling NER/SVM scalability issues and not pursue the development of the recommended service-oriented architecture for the time being.

In this research work, we focus on the main SVM learning and classification modules, where both of the binary and multi-class cases co-exist and are accessible through a user machine type selection. Feature extraction and classification performance evaluation remain external to the database embedded solution, with provisions made to import the input data vectors and to export the classified output for evaluation. We use the JFex software (Giuliano et al. 2005) for feature extraction. The database modules include an input data loading module in order to streamline the inclusion of an embedded feature extraction module in the future.

The evaluation of this research will concentrate on the biomedical literature domain using the JNLPBA-04 (BioNLP) datasets and evaluation tools. We include evaluation data in the database schema design and provide a module to export the classified data to be evaluated by the external tools. This again should make the inclusion of embedded evaluation modules easier in the future.

In order to validate the applicability of the NER/SVM solution to other cases, we conduct experiments using the CoNLL-02 Spanish and Dutch datasets and evaluate these experiments using the corresponding evaluation scripts for the CoNLL challenge task.
1.3 Dissertation Outline

This thesis is organized as follows. In Chapter 2, we present the named entity recognition problem and the current state of the art solutions for it. We then explore the language-independent NER research activities as well as those specific to recognizing named entities in biomedical abstracts as an example of a specialized domain. The methods and features used by many systems using the same sets of training and testing data from three NER challenge tasks are summarized followed by a discussion of the named entity recognition scalability challenges.

The mathematical foundation of Support Vector Machine (SVM) is briefly introduced in Chapter 3. The binary classification for linearly separable and non-linearly separable data is presented, followed by the different approaches used to classify data with several classes. We conclude the chapter with a discussion of SVM scalability issues and briefly introduce how they are addressed in the literature.

In order to investigate the potential problems associated with named entity recognition using support vector machines, we performed a series of baseline single class and multi-class experiments using datasets from the biomedical domain. The approach used is one that eliminates the use of prior language or domain-specific knowledge. The detailed architecture and methods used as well as the experiments’ results are presented in Chapter 4. We compare the results to other systems using the same datasets and demonstrate that the simplified NER process is capable of achieving performance measures that are comparable to published results. We discuss the challenges and problems encountered and summarize the NER/SVM scalability issues. The usability aspects of SVM machine learning are also examined.
To address the scalability issues of the NER/SVM solution using high-dimensional input space, we begin by tackling the training computational time component. In Chapter 5, we explore the mathematical foundation of multi-class support vector machines and the evolution of the problem formulation aiming to reduce the training time by improving the optimization algorithms. We then introduce a new cutting plane algorithm that approximates the optimization problem while achieving good out-of-the-box performance in linear time. The cutting plane algorithm is first implemented as a standalone C executable in order to study its effectiveness in reducing the computational time.

Having improved the training time for the multi-class classification problem, we then present a database-supported SVM framework that seeks to promote the usability of the SVM solution and the reusability of its classification results, as well as improve scalability by lowering the online memory requirements. The database framework addresses both the single class and multi-class classification problems. The database architecture, schema design, and a description of the embedded SVM modules are described in Chapter 6.

In Chapter 7 and 8, we report the results of a series of scalability experiments using existing SVM implementations as well as our newly developed standalone and database-supported solution. Chapter 7 examines the results of single class experiments classifying protein named entities in the JNLPBA-04 biomedical data. In Chapter 8, we compare the outcomes of multi-class experiments using the same biomedical datasets where the following five named entity types are discovered: protein, DNA, RNA, cell line, and cell type. In addition, we apply the existing and new multi-class solutions to two additional
datasets, namely the CoNLL-02 Spanish and Dutch challenge data aiming to classify general named entities such as a person, location, or organization.

At the final stages of this work, we conducted a final literature review where we came across a new yet-to-be-published paper that analyzes some cutting plane algorithms using the latest 1-slack SVM formulation that we use in conjunction with our new multi-class cutting plane algorithm. This paper briefly mentions a multi-class implementation and upon verification of the source code, we realized that it is a newer implementation than the previous one used for our baseline experiments. In Chapter 9, we compare our cutting plane algorithm results with those achieved using the newly available multi-class version. We report on the differences in training time, memory, and performance in Chapter 9.

Finally, we discuss some of the known concerns about our proposed research work and how we addressed them as part of the solution, as well as a self-evaluation of the contributions and success criteria in Chapter 10. We also present some research directions that we contemplate for future research. These directions include a recommendation for a dynamic service-oriented machine learning architecture, incremental learning, feature selection, and unsupervised learning. We briefly brainstorm on some ideas in each research direction.
Chapter 2
Named Entity Recognition

Named entity recognition (NER) is one of the important tasks in information extraction, which involves the identification and classification of words or sequences of words denoting a concept or entity. Examples of named entities in general text are names of persons, locations, or organizations. Domain-specific named entities are those terms or phrases that denote concepts relevant to one particular domain. For example, protein and gene names are named entities which are of interest to the domain of molecular biology and medicine. The massive growth of textual information available in the literature and on the Web necessitates the automation of identification and management of named entities in text.

The task of identifying named entities in a particular language is often accomplished by incorporating knowledge about the language taxonomy in the method used. In the English language, such knowledge may include capitalization of proper names, known titles, common prefixes or suffixes, part of speech tagging, and/or identification of noun phrases in text. Techniques that rely on language-specific knowledge may not be suitable for porting to other languages. For example, the Arabic language does not use capitalization to identify proper names, and word variations are based on the use of infixes in addition to prefixes and suffixes. Moreover, the composition of named entities in literature pertaining to specific domains follows different rules in each, which may or may not benefit from those relevant to general NER.
This chapter presents an overview of the research activities in the area of named entity recognition in several directions related to the focus of this work:

- Named entity recognition approaches
- Common machine learning architecture for NER
- Named entity recognition in the biomedical literature context
- Language-independent named entity recognition
- Named entity recognition challenges

### 2.1 Named Entity Recognition Approaches

Named entity recognition activities began in the late 1990’s with limited number of general categories such as names of persons, organizations, and locations (Sekine 2004; Nadeau and Sekine 2007). Early systems were based on the use of dictionaries and rules built by hand, and few used supervised machine learning techniques. The CoNLL-02 (Tjong Kim Sang 2002a) and CoNLL-03 (Tjong Kim Sang and De Meulder 2003) discussed later in this chapter provided valuable NE evaluation tasks for four languages: English, German, Spanish, and Dutch. With the extension of named entity recognition activities to new languages and domains, more entity categories are introduced which made the methods relying on manually built dictionaries and rules much more difficult to adopt, if at all feasible.

The extended applicability of NER in new domains led to more adoption of supervised machine learning techniques which include:

- Decision Trees (Paliouras et al. 2000; Black and Vasilakopoulos 2002)
- AdaBoost (Carreras et al. 2002, 2003b; Wu et al. 2003; Tsukamoto et al. 2002)
• Maximum Entropy Model (ME) (Kazama et al. 2001; Bender et al. 2003; Chieu and Ng 2003; Curran and Clark 2003; Lin et al. 2004; Nissim et al. 2004)
• Boosting and voted perceptrons (Carreras et al. 2003a; Dong and Han 2005; Wu et al. 2002; Wu et al. 2003)
• Recurrent Neural Networks (RNN) (Hammerton 2003)
• Conditional Random Fields (CRF) (McCallum and Li 2003; Settles 2004; Song et al. 2004; Talukdar et al. 2006)

Memory-based (De Meulder and Daelemans 2003; Hendrickx and Bosch 2003; Tjong Kim Sang 2002b) and transformation-based (Black and Vasilakopoulos 2002; Florian 2002; Florian et al. 2003) techniques have been successfully used for recognizing general named entities where the number of categories is limited, but are less adopted in more complex NER tasks such as the biomedical domain (Finkel et al. 2004). Recent NER systems also combined several classifiers using different machine learning techniques in order to achieve better performance results (Florian et al. 2003; Klein et al. 2003; Mayfield et al. 2003; Song et al. 2004; Rössler 2004; Zhou 2004).

With the growing adoption of machine learning techniques for NER, especially for specialized domains, the need for developing semi-supervised or unsupervised solutions increases. Supervised learning methods rely on the existence of manually annotated
training data, which is very expensive in terms of labor and time and a hindering factor for many complex domains with growing nomenclature. However, using unannotated training data or a mixture of labeled and unlabeled data requires the development of new NER machine learning solutions based on clustering and inference techniques. Named entity recognition systems that attempted to use unannotated training data include (Cucerzan and Yarowsky 1999; Riloff and Jones 1999; De Meulder and Daelemans 2003; Yangarber and Grishman 2001; Hendrickx and Bosch 2003; Goutte et al. 2004; Yangarber et al. 2002; Zeng et al. 2003; Bodenreider et al. 2002; Collins and Singer 1999).

Comparing the relative performance of the various NER approaches is a nontrivial task. The performance of earlier systems that relied on manually built dictionaries and rules depends in the first place on the quality of the rules and dictionaries used. Systems based on statistical approaches and machine learning techniques, whether they use just one method or a combination of several techniques, require the use of annotated training data and extraction of many orthographic, contextual, linguistic, and domain-specific features in addition to possibly using external resources such as dictionaries, gazetteers, or even the World Wide Web. Therefore, judging the performance of a given system cannot be made solely based on the choice of a machine learning approach but rather on the overall solution design and final performance results. This observation makes the use of a machine learning technique for NER an art more than a science.

2.2 Common Machine Learning Architecture

Constructing a named entity recognition solution using a machine learning approach requires many computational steps including preprocessing, learning, classification, and
post-processing. The specific components included in a given solution vary but they may be viewed as making part of the following groups summarized in Figure 2.1.

2.2.1 Preprocessing Modules

Using a supervised machine learning technique relies on the existence of annotated training data. Such data is usually created manually by humans or experts in the relevant field. The training data needs to be put in a format that is suitable to the solution of choice. New data to be classified also requires the same formatting. Depending on the needs of the solution, the textual data may need to be tokenized, normalized, scaled, mapped to numeric classes, prior to being fed to a feature extraction module. To reduce the training time with large training data, some techniques such as chunking or instance pruning (filtering) may need to be applied.

2.2.2 Feature Extraction

In the feature extraction phase, training and new data is processed by one or more pieces of software in order to extract the descriptive information about it. The choice of feature extraction modules depends on the solution design and may include the extraction of orthographic and morphological features, contextual information about how tokens appear in the documents, linguistic information such as part-of-speech or syntactic indicators, and domain-specific knowledge such as the inclusion of specialized dictionaries or gazetteers (reference lists). Some types of information may require the use of other machine learning steps to generate it, for example, part-of-speech tagging is usually performed by a separate machine learning and classification software which may or may not exist for a particular language.
Preparing the data for use by the feature extractor may require special formatting to suit the input format of the software. Also, depending on the choice of machine learning software, one may need to reformat the output of the feature extraction to be compatible with what’s expected by the machine learning module(s). Due to the lack of standardization in this area and because no integrated solutions exist for named entity recognition, several incompatibilities exist between the many tools one may use to build the overall architecture. In addition, one may also need to build customized interfacing modules to fit all the pieces of the solution together.

### 2.2.3 Machine Learning and Classification

Most of the publicly available machine learning software use a two-phased approach where learning is first performed to generate a trained machine followed by a classification step. The trained model for a given problem can be reused for many classifications as long as there is no need to change the learning parameters or the training data.

### 2.2.4 Post-Processing Modules

The post-processing phase prepares the classified output for use by other applications and/or for evaluation. The classified output may need to be reformatted, regrouped into one large chunk if the input data was broken down into smaller pieces prior to being processed, remapped to reflect the string class names, and tested for accuracy by evaluation tools. The final collection of annotated documents may be reviewed by human experts prior to being used for other needs.
2.3 Performance Measures

The performance measures used to evaluate the named entity recognition systems participating in the CoNLL-02, CoNLL-03 and JNLPBA-04 challenge tasks are precision, recall, and the weighted mean $F_{\beta=1}$-score. Precision is the percentage of named entities found by the learning system that are correct. Recall is the percentage of named entities present in the corpus that are found by the system. A named entity is correct only if it is an exact match of the corresponding entity in the data file, i.e., the complete named entity is correctly identified. Definitions of the performance measures used are summarized below. The same performance measures are used to evaluate the results of
the experiments performed during the course of this work. It is worth noting that since named entities are frequently composed of several words, the definition of the performance measures usually refers to complete matches of multi-word named entities, and not to individual word classifications.

\[
\text{Recall} = \frac{\text{# of correctly classified entities}}{\text{# of entities in the corpus}} = \frac{tp}{tp + fn}
\]

\[
\text{Precision} = \frac{\text{# of correctly classified entities}}{\text{# of entities found by algorithm}} = \frac{tp}{tp + fp}
\]

\[
\text{Weighted Mean } F_\beta = \frac{(1 + \beta^2) \times (\text{precision} \times \text{recall})}{\beta^2 \times \text{precision} + \text{recall}}
\]

With \( \beta = 1 \), \( F_{\beta=1} = \frac{2 \times (\text{precision} \times \text{recall})}{(\text{precision} + \text{recall})} \)

### 2.4 Biomedical Named Entity Recognition

Biomedical entity recognition aims to identify and classify technical terms in the domain of molecular biology that are of interest to biologists and scientists. Example of such entities are protein and gene names, cell types, virus name, DNA sequence, and others. The U.S. National Library of Medicine maintains a large collection of controlled vocabulary, MeSH (NLM 2007b), used for indexing articles for MEDLINE/PubMed (NLM 2007a) and to provide a consistent way to retrieve information that may use different terminology for the same concepts. PubMed’s constantly growing collection of articles raises the need for automated tools to extract new entities appearing in the literature.

The biomedical named entity recognition remains a challenging task as compared to general NER. Systems that achieve high accuracy in recognizing general names in the
newswires (Tjong Kim Sang and De Meulder 2003) have not performed as well in the biomedical NER with an accuracy of 20 or 30 points difference in their F-score measure. The biomedical field NER presents many challenges due to growing nomenclature, ambiguity in the left boundary of entities caused by descriptive naming, shortened forms due to abbreviation and aliasing, difficulty to create consistently annotated training data with large number of classes, etc. (Kim et al. 2004). Definitions of some biomedical named entities and sample annotated biomedical abstracts are provided in Appendix C.

Biomedical named entity recognition systems make use of publicly or privately available corpora to train and test their systems. The quality of the corpus used impacts the output performance as one would expect. Cohen et al. (Cohen et al. 2005) compare the quality of six publicly available corpora and evaluate them in terms of age, size, design, format, structural and linguistic annotation, semantic annotation, curation, and maintenance. The six public corpora are the Protein Design Group (PDG) corpus (Blaschke et al. 1999), the University of Wisconsin corpus (Craven and Kumleim 1999), Medstract (Pustejovsky et al. 2002), the Yapex corpus (Franzen et al. 2002; Eriksson et al. 2002), the GENIA corpus (Ohta et al. 2002; Kim et al. 2003), and the BioCreative task dataset (GENETAG) (Tanabe et al. 2005). These corpora are widely used in the biomedical named entity recognition research community and serve as basis of performance comparison. Cohen et al. (Cohen et al. 2005) conclude that the GENIA corpus’ quality has improved over the years most probably due to its continued development and maintenance. The GENIA corpus is the source of the JNLPBA-04 challenge task datasets described in Appendix B, often referred to as the BioNLP data. We used the JNLPBA-04 (BioNLP) datasets for the biomedical NER experiments
performed during the course of this work. Evaluation of this research is primarily using the JNLPBA-04 datasets as they represent a challenging NER domain. To demonstrate the applicability of potential solutions to other languages and/or domains, we include experiments using the CoNLL-02 Spanish and Dutch datasets. When evaluating the classification performance results of the three main tasks, we use the evaluation scripts belonging to each challenge task in order to provide a common basis of comparison of the results to published results using the same datasets.

The availability of common experimentation data and evaluation tools provides a great opportunity for researchers to compare their performance results against other systems. The BioNLP tools and data are used by many systems with published results.

Table 2.1 summarizes the methods used by several systems performing biomedical named entity recognition using the JNLPBA-04 (BioNLP) datasets as well as the features and any external resources they used. Systems performing biomedical NER used Support Vector Machine (SVM), Hidden Markov Model (HMM), Maximum Entropy Markov Model (MEMM), or Conditional Random Fields (CRF) as their classification methods either combined or in isolation. The features used by the different systems are listed in abbreviated form in Table 2.1 and include some or all of the following:

- lex: lexical features beyond simple orthographic features
- ort: orthographic information
- aff: affix information (character n-grams)
- ws: word shapes
- gen: gene sequences (ATCG sequences)
- wv: word variations
• len: word length
• gaz: gazetteers (reference lists of known named entities)
• pos: part-of-speech tags
• np: noun phrase tags
• syn: syntactic tags
• tri: word triggers
• ab: abbreviations
• cas: cascaded entities
• doc: global document information
• par: parentheses handling information
• pre: previously predicted entity tags
• External resources: B: British National Corpus (Oxford 2005); M: MEDLINE corpus (NLM 2007a); P: Penn Treebank II corpus (The Penn Treebank Project 2002); W: World Wide Web; V: virtually generated corpus; Y: Yapex (Eriksson et al. 2002) (Y); G: GAPSCORE (Chang et al. 2004)

In the following paragraphs we will highlight the top performing systems (Zhou and Su 2004; Finkel et al. 2004; Settles 2004; Giuliano et al. 2005). Two of the top performing systems used SVM, where one combined it with HMM (Zhou and Su 2004) and the other used it as the only classification model (Giuliano et al. 2005). The second best system used a maximum entropy approach (Finkel et al. 2004) and the system that ranked third used conditional random fields (Settles 2004).

Zhou and Su (Zhou and Su 2004) combined a Support Vector Machine (SVM) with a sigmoid kernel and a Hidden Markov Model (HMM). The system explored deep
knowledge in external resources and specialized dictionaries in order to derive alias information, cascaded entities, and abbreviation resolution. The authors made use of existing gene and protein name databases such as SwissProt in addition to a large number of orthographic and language-specific features such as part-of-speech tagging and head noun triggers. The system achieved the highest performance with an F-score of 72.6% with post-processing using the external resources. However, the F-score achieved using machine learning alone, i.e., prior to post-processing, is 60.3%. Given the complex system design and the large number of preprocessing and post-processing steps undertaken in order to correctly identify the named entities, it is difficult to judge the impact of the machine learning approach alone when comparing this system to others. The performance gain is mostly due to the heavy use of specialized dictionaries, gazetteer lists, and previously identified entities in order to flag known named entities.

Finkel et al. (Finkel et al. 2004; Dingare et al. 2005) achieved F-score of 70.1% on BioNLP data using a maximum entropy-based system previously used for the language-independent task CoNLL-03. The system used a rich set of local features and several external sources of information such as parsing and searching the web, domain-specific gazetteers, and compared NE’s appearance in different parts of a document. Post-processing attempted to correct multi-word entity boundaries and combined results from two classifiers trained forward and backward. (Dingare et al. 2005) confirmed the known challenge in identifying named entities in biomedical documents, which causes the performance results to lag behind those achieved in general NER tasks such as CoNLL-03 by 18% or more.
Settles (Settles 2004) used Conditional Random Fields (CRM) and a rich set of orthographic and semantic features to extract the named entities. The system also made use of external resources, gazetteer lists, and previously identified entities in order to flag known named entities.

Giuliano et al. (Giuliano et al. 2005) used a Support Vector Machine (SVM) with a large number of orthographic and contextual features extracted using the jFex software. The system incorporated part-of-speech tagging and word features of tokens surrounding each analyzed token in addition to features similar to those used in this experiment. In addition, Giuliano et al. (Giuliano et al. 2005) pruned the data instances in order to reduce the dataset size by filtering out frequent words from the corpora because they are less likely to be relevant than rare words.

The comparison of the top performing systems does not single out a particular machine learning methodology in being more efficient than others. From observation of the rich feature sets used by these systems, which include language and domain-specific knowledge, we may conclude that the combination of machine learning with prior domain knowledge and a large set of linguistic features is what led to the superior performance of these systems as compared to others that used the same machine learning choice with a different set of features.

2.5 Language-Independent NER

Table 2.3 and Table 2.4 summarize the methods and features used by several systems performing language-independent named entity recognition using CoNLL-02 datasets and CoNLL-03 datasets respectively. The composition of these datasets is described in details in Appendix B. CoNLL-02 concerned general NER in Spanish and Dutch
languages, while CoNLL-03 focused on English and German languages. Systems performing language-independent NER used Support Vector Machine (SVM), Hidden Markov Model (HMM), Maximum Entropy Markov Model (MEMM), Conditional Random Fields (CRF), Conditional Markov Model (CMM), Robust Risk Minimization (RMM), Voted Perceptrons (PER), Recurrent Neural Networks (RNN), AdaBoost, memory-based techniques (MEM), or transformation-based techniques (TRAN) as their classification methods either combined or in isolation. The features used by the different systems are listed in abbreviated form in Table 2.3 and Table 2.4 and include some or all of the following:

- **lex**: lexical features beyond simple orthographic features
- **ort**: orthographic information
- **aff**: affix information (character n-grams)
- **ws**: word shapes
- **wv**: word variations
- **gaz**: gazetteers
- **pos**: part-of-speech tags
- **tri**: word triggers
- **cs**: global case information
- **doc**: global document information
- **par**: parentheses handling
- **pre**: previously predicted entity tags
- **quo**: word appears between quotes
- **bag**: bag of words
CoNLL-02 top performing systems used Adaboost (Carreras et al. 2002; Wu et al. 2002), character-based tries (Cucerzan and Yarowsky 2002), and stacked transformation-based classifiers (Florian 2002) learning techniques. (Carreras et al. 2002) combined boosting and fixed depth decision trees to identify the beginning and end of an entity and whether a token is within a named entity. (Florian 2002) stacked several transformation-based classifiers and Snow (Sparse Network of Winnows) – “an architecture for error-driven machine learning, consisting of a sparse network of linear separator units over a common predefined or incrementally learned feature space” – in order to boost the system’s performance. The output of one classifier serves as input to the next one. (Cucerzan and Yarowsky 2002) used a semi-supervised approach using a small number of seeds in a boosting manner based on character-based tries. (Wu et al. 2002) combined boosting and one-level decision trees as classifiers.

The F-score values achieved by the top performing systems in CoNLL-02 were in the high 70’s to low 80’s for Spanish, and slightly less for Dutch. All systems used a rich set of various features. It is interesting to note that the systems’ performance ranking, summarized in Table 2.3, differed slightly between the two languages. In 2005, Giuliano (Giuliano et al. 2005) classified the Dutch data using SVM and a rich set of orthographic and contextual features in addition to part-of-speech tagging. Giuliano achieved an F-score of 75.60 which ranks second as compared to the CoNLL-02 top performing systems.

The CoNLL-03 task required the inclusion of a machine learning technique and the incorporation of additional resources such as gazetteers and/or unannotated data in
addition to the training data (Tjong Kim Sang and De Meulder 2003). All participating systems used more complex combinations of learning tools and features than the CoNLL-02 systems. A summary of the methods and features as well as the performance results of CoNLL-03 systems is presented in Table 2.4. The three top performing systems used Maximum Entropy Models (MEM). (Florian et al. 2003) combined MEM with a Hidden Markov Model (HMM), Robust Risk Minimization, a transformation-based technique, and a large set of orthographic and linguistic features. (Klein et al. 2003) used an HMM and a Conditional Markov Model in addition to Maximum Entropy. (Chieu and Ng 2003) did not combine MEM with other techniques.

An interesting observation is that the systems that combined Maximum Entropy with other techniques performed consistently in the two languages, while (Chieu and Ng 2003) ranked second on the English data and #12 on the German data. (Wong and Ng 2007) used a Maximum Entropy approach with the CoNLL-03 data and a collection of unannotated data and achieved an F-score of 87.13 on the English data. The system made use of “a novel yet simple method of exploiting this empirical property of one class per named entity to further improve the NER accuracy” (Wong and Ng 2007). Compared to the CoNLL-03 systems, (Wong and Ng 2007) ranks third on the English data experiments. No results were reported using the German data.

Another important note is that the CoNLL-03 systems achieved much higher accuracy levels on the English data with F-scores reaching in the high 80’s, while F-score levels using the German data remained comparable to CoNLL-02 results going as high as 72.41 for the best score. This observation leads us to confirm our earlier conclusion that judging the performance of a machine learning technique based on NER performance
would not be necessarily accurate, as the same system achieves inconsistent performance levels when used with different languages. Our conclusion remains that the performance of a given NER system is a result of the combination of classification techniques, features, and external resources used, and no one component may be deemed responsible for the outcome separately. The quality and complexity of the training and test data is also a major contributing factor in reaching a certain performance.

Poibeau (Poibeau et al. 2003) discusses the issues of multilingualism and proposes an open framework to support multiple languages which include: Arabic, Chinese, English, French, German, Japanese, Finnish, Malagasy, Persian, Polish, Russian, Spanish and Swedish. The project maintains resources for all supported languages using the Unicode character set and apply almost the same architecture to all languages. Interestingly, the multilingual system capitalizes on commonality across Indo-European languages and shares external resources such as dictionaries by different languages.

In conclusion, the review of the NER approaches and how they are applied in different languages and domains demonstrates that the performance of an NER system in a given context depends on the overall solution including the classification technique(s), features, and external resources used. The quality and complexity of the training and test data also impacts the final accuracy level. Many researchers who worked on biomedical NER noted that the same systems that achieved high performance measures on general or multilingual NER failed to achieve similar results when used in the biomedical domain. This highlights the complex nature of biomedical NER and the need for special approaches to deal with its inherent challenges.
Table 2.1 - Overview of BioNLP Methods, Features, And External Resources

<table>
<thead>
<tr>
<th>System</th>
<th>Methods/Features/External Resources</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhou (Zhou and Su 2004)</td>
<td>SVM + HMM + aff + ort + gen + gaz + pos + tri + ab + cas + pre</td>
<td>Combined models + cascaded entities + previous predictions + language &amp; domain resources</td>
</tr>
<tr>
<td>Finkel (Finkel et al. 2004; Dingare et al. 2005)</td>
<td>MEMM + lex + aff + ws + gaz + pos + syn + ab + doc + par + pre + B + W</td>
<td>Language &amp; domain resources + lexical features + previous predictions + boundary adjustment + global document information + Web</td>
</tr>
<tr>
<td>Settles (Settles 2004)</td>
<td>CRF + lex + aff + ort + ws + gaz + tri + pre + W</td>
<td>Domain resources + Web + previous predictions</td>
</tr>
<tr>
<td>Song (Song et al. 2004)</td>
<td>SVM + CRF + lex + aff + ort + pos + np + pre + V</td>
<td>Combined models + language &amp; domain resources + previous predictions</td>
</tr>
<tr>
<td>Zhao (Zhao 2004)</td>
<td>HMM + lex + pre + M</td>
<td>Lexical features + domain resources + previous predictions + Medline</td>
</tr>
<tr>
<td>Rössler (Rössler 2004)</td>
<td>SVM + HMM + aff + ort + gen + len + pre + M</td>
<td>Combined models + domain resources + previous predictions + Medline</td>
</tr>
<tr>
<td>Park (Park et al. 2004)</td>
<td>SVM + aff + ort + ws + gen + wv + pos + np + tri + M + P</td>
<td>Language &amp; domain resources + Medline + Penn Treebank corpus</td>
</tr>
<tr>
<td>Lee (Lee, Hwang et al. 2004)</td>
<td>SVM + lex + aff + pos + Y + G</td>
<td>Lexical features + language resources + Yapex corpus + Gapscore corpus</td>
</tr>
<tr>
<td>Giuliano (Giuliano et al. 2005)</td>
<td>SVM + lex + ort + pos + ws + wv</td>
<td>Lexical features + language resources + collocation + instance pruning</td>
</tr>
</tbody>
</table>

SVM: Support Vector Machine; HMM: Hidden Markov Model; MEMM: Maximum Entropy Markov Model; CRF: Conditional Random Fields; lex: lexical features; aff: affix information (character n-grams); ort: orthographic information; ws: word shapes; gen: gene sequences (ATCG sequences); wv: word variations; len: word length; gaz: gazetteers; pos: part-of-speech tags; np: noun phrase tags; syn: syntactic tags; tri: word triggers; ab: abbreviations; cas: cascaded entities; doc: global document information; par: parentheses handling; pre: previously predicted entity tags; External resources (ext): B: British National Corpus; M: MEDLINE corpus; P: Penn Treebank II corpus; W: world wide web; V: virtually generated corpus; Y: Yapex; G: GAPSCORE. Legend Source: (Kim et al. 2004)

Table 2.2 – Performance Comparison of Systems Using BioNLP Datasets

(Recall / Precision/ \(F_{\beta=1}\))

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhou (Zhou and Su 2004), with post-processing</td>
<td>75.3 / 69.5 / 72.3</td>
<td>77.1 / 69.2 / 72.9</td>
<td>75.6 / 71.3 / 73.8</td>
<td>75.8 / 69.5 / 72.5</td>
<td>76.0 / 69.4 / 72.6</td>
</tr>
<tr>
<td>Finkel (Finkel et al. 2004)</td>
<td>66.9 / 70.4 / 68.6</td>
<td>73.8 / 69.4 / 71.5</td>
<td>72.6 / 69.3 / 70.9</td>
<td>71.8 / 67.5 / 69.6</td>
<td>71.6 / 68.6 / 70.1</td>
</tr>
<tr>
<td>Settles (Settles 2004)</td>
<td>63.6 / 71.4 / 67.3</td>
<td>72.2 / 68.7 / 70.4</td>
<td>71.3 / 69.6 / 70.5</td>
<td>71.3 / 68.8 / 70.1</td>
<td>70.3 / 69.3 / 69.8</td>
</tr>
<tr>
<td>Giuliano (Giuliano et al. 2005)</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>64.4 / 69.8 / 67.0</td>
</tr>
<tr>
<td>Song (Song et al. 2004)</td>
<td>60.3 / 66.2 / 63.1</td>
<td>71.2 / 65.6 / 68.2</td>
<td>69.5 / 65.8 / 67.6</td>
<td>68.3 / 64.0 / 66.1</td>
<td>67.8 / 64.8 / 66.3</td>
</tr>
<tr>
<td>Zhao (Zhao 2004)</td>
<td>63.2 / 60.4 / 61.8</td>
<td>72.5 / 62.6 / 67.2</td>
<td>69.1 / 60.2 / 64.7</td>
<td>69.2 / 60.3 / 64.4</td>
<td>69.1 / 61.0 / 64.8</td>
</tr>
<tr>
<td>Rössler (Rössler 2004)</td>
<td>59.2 / 60.3 / 59.8</td>
<td>70.3 / 61.8 / 65.8</td>
<td>68.4 / 61.5 / 64.8</td>
<td>68.3 / 60.4 / 64.1</td>
<td>67.4 / 61.0 / 64.0</td>
</tr>
<tr>
<td>Park (Park et al. 2004)</td>
<td>62.8 / 55.9 / 59.2</td>
<td>70.3 / 61.4 / 65.6</td>
<td>65.1 / 60.4 / 62.7</td>
<td>65.9 / 59.7 / 62.7</td>
<td>66.5 / 59.8 / 63.0</td>
</tr>
<tr>
<td>Lee (Lee, Hwang et al. 2004)</td>
<td>42.5 / 42.0 / 42.2</td>
<td>52.5 / 49.1 / 50.8</td>
<td>53.8 / 50.9 / 52.3</td>
<td>52.3 / 48.1 / 50.1</td>
<td>50.8 / 47.6 / 49.1</td>
</tr>
<tr>
<td>Baseline (Kim et al. 2004)</td>
<td>47.1 / 33.9 / 39.4</td>
<td>56.8 / 45.5 / 50.5</td>
<td>51.7 / 46.3 / 48.8</td>
<td>52.6 / 46.0 / 49.1</td>
<td>52.6 / 43.6 / 47.7</td>
</tr>
<tr>
<td>System</td>
<td>Methods/Features</td>
<td>Spanish Results (Rec/Prec/ $F_{\beta=1}$)</td>
<td>Dutch Results (Rec/Prec/ $F_{\beta=1}$)</td>
<td>$F_{\beta=1}$ Rank</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------</td>
<td>-------------------------------------------</td>
<td>------------------------------------------</td>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td>Carreras (Carreras et al. 2002)</td>
<td>ADA + decision trees + lex + pre + pos + gaz + ort + ws</td>
<td>81.40 / 81.38 / 81.39 1</td>
<td>76.29 / 77.83 / 77.05 1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Florian (Florian 2002)</td>
<td>Stacked TRAN + Snow + forward-backward</td>
<td>79.40 / 78.70 / 79.05 2</td>
<td>74.89 / 75.10 / 74.99 3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Cucerzan (Cucerzan and Yarowsky 2002)</td>
<td>Character-based tries + pos + aff + lex + gaz</td>
<td>76.14 / 78.19 / 77.15 3</td>
<td>71.62 / 73.03 / 72.31 5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Wu (Wu et al. 2002)</td>
<td>ADA + decision tree + lex + pos + gaz + cs + pre</td>
<td>77.38 / 75.85 / 76.61 4</td>
<td>73.83 / 76.95 / 75.36 2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Burger (Burger et al. 2002)</td>
<td>HMM + gaz + pre</td>
<td>77.44 / 74.19 / 75.78 5</td>
<td>72.45 / 72.69 / 72.57 4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Tjong Kim Sang (Tjong Kim Sang 2002b)</td>
<td>MEM + stacking + combination</td>
<td>75.55 / 76.00 / 75.78 6</td>
<td>68.88 / 72.56 / 70.67 7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Patrick (Patrick et al. 2002)</td>
<td>Six stages using compiled lists and n-grams + context</td>
<td>73.52 / 74.32 / 73.92 7</td>
<td>68.90 / 74.01 / 71.36 6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Jansche (Jansche 2002)</td>
<td>CMM + ws +cs + collocation</td>
<td>73.76 / 74.03 / 73.89 8</td>
<td>69.26 / 70.11 / 69.68 8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Malouf (Malouf 2002)</td>
<td>MEMM (also tried HMM) + pre + boundary detection</td>
<td>73.39 / 73.93 / 73.66 9</td>
<td>65.50 / 70.88 / 68.08 9</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Tsukamoto (Tsukamoto et al. 2002)</td>
<td>ADA (five cascaded classifiers)</td>
<td>74.12 / 69.04 / 71.49 10</td>
<td>65.02 / 57.33 / 60.93 10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Black (Black and Vasilakopoulos 2002)</td>
<td>TRAN (also tried decision trees)</td>
<td>66.24 / 68.78 / 67.49 11</td>
<td>51.69 / 62.12 / 56.43 12</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>McNamee (McNamee and Mayfield 2002)</td>
<td>SVM (two cascaded) + 9000 binary features</td>
<td>66.51 / 56.28 / 60.97 12</td>
<td>63.24 / 56.22 / 59.52 11</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Baseline (Tjong Kim Sang 2002a)</td>
<td>TR</td>
<td>56.48 / 26.27 / 35.86 13</td>
<td>45.19 / 64.38 / 53.10 13</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Giuliano (Giuliano et al. 2005)</td>
<td>SVM + lex + ort + pos + ws + wv</td>
<td>not used</td>
<td>69.60 / 82.80 / 75.60 * 2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

SVM: Support Vector Machine; HMM: Hidden Markov Model; MEMM: Maximum Entropy Markov Model; CRF: Conditional Random Fields; CMM: Conditional Markov Model; RRM: Robust Risk Minimization; PER: Voted Perceptrons; RNN: Neural Networks; ADA: AdaBoost; MEM: Memory-Based; TRAN: Transformation-Based; lex: lexical features; aff: affix information (character n-grams); ort: orthographic information; ws: word shapes; wv: word variations; gaz: gazetteers; pos: part-of-speech tags; tri: word triggers; cs: global case information; doc: global document information; par: parentheses handling; pre: previously predicted entity tags; quo: between quotes; bag: bag of words; chu: chunk tags
Table 2.4 – Comparison of Systems Using the CoNLL-03 Data

<table>
<thead>
<tr>
<th>System</th>
<th>English Results</th>
<th>German Results</th>
<th>SVM: Support Vector Machine; HMM: Hidden Markov Model; MEMM: Maximum Entropy Markov Model; CRF: Conditional Random Fields; CMM: Conditional Markov Model; RRM: Robust Risk Minimization; PER: Voted Perceptrons; RNN: Neural Networks; ADA: AdaBoost; MEM: Memory-Based; TRAN: Transformation-Based; lex: lexical features; aff: affix information (character n-grams); ort: orthographic information; ws: word shapes; wv: word variations; gaz: gazetteers; pos: part-of-speech tags; tri: word triggers; cs: global case information; doc: global document information; par: parentheses handling; pre: previously predicted entity tags; quo: between quotes; bag: bag of words; chu: chunk tags</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System</strong></td>
<td><strong>Methods/Features</strong></td>
<td><strong>Performance</strong></td>
<td><strong>Performance</strong></td>
</tr>
<tr>
<td><strong>English Results</strong></td>
<td><strong>(Rec/Prec/ Fβ=1)</strong></td>
<td><strong>Fβ=1</strong> Rank</td>
<td><strong>(Rec/Prec/ Fβ=1)</strong></td>
</tr>
<tr>
<td><strong>German Results</strong></td>
<td><strong>(Rec/Prec/ Fβ=1)</strong></td>
<td></td>
<td><strong>(Rec/Prec/ Fβ=1)</strong></td>
</tr>
<tr>
<td><strong>Table 2.4 – Comparison of Systems Using the CoNLL-03 Data</strong></td>
<td><strong>English Results</strong></td>
<td><strong>German Results</strong></td>
<td><strong>SVM: Support Vector Machine; HMM: Hidden Markov Model; MEMM: Maximum Entropy Markov Model; CRF: Conditional Random Fields; CMM: Conditional Markov Model; RRM: Robust Risk Minimization; PER: Voted Perceptrons; RNN: Neural Networks; ADA: AdaBoost; MEM: Memory-Based; TRAN: Transformation-Based; lex: lexical features; aff: affix information (character n-grams); ort: orthographic information; ws: word shapes; wv: word variations; gaz: gazetteers; pos: part-of-speech tags; tri: word triggers; cs: global case information; doc: global document information; par: parentheses handling; pre: previously predicted entity tags; quo: between quotes; bag: bag of words; chu: chunk tags</strong></td>
</tr>
<tr>
<td><strong>Florian (Florian et al. 2003)</strong></td>
<td>MEMM + HMM + RRM + TRAN + lex + pos + aff + pre + ort + gaz + chu + cs</td>
<td>88.54 / 88.99 / 88.76</td>
<td>1</td>
</tr>
<tr>
<td><strong>Chieu (Chieu and Ng 2003)</strong></td>
<td>MEMM + lex + pos + aff + pre + ort + gaz + tri + quo + doc</td>
<td>88.51 / 88.12 / 88.31</td>
<td>2</td>
</tr>
<tr>
<td><strong>Klein (Klein et al. 2003)</strong></td>
<td>MEMM + HMM + CMM + lex + pos + aff + pre</td>
<td>86.21 / 85.93 / 86.07</td>
<td>3</td>
</tr>
<tr>
<td><strong>Zhang (Zhang and Johnson 2003)</strong></td>
<td>RRM + lex + pos + aff + pre + ort + gaz + chu + tri</td>
<td>84.88 / 86.13 / 85.50</td>
<td>4</td>
</tr>
<tr>
<td><strong>Curran (Curran and Clark 2003)</strong></td>
<td>MEMM + lex + pos + aff + pre + ort + gaz + ws + cs</td>
<td>85.50 / 84.29 / 84.89</td>
<td>6</td>
</tr>
<tr>
<td><strong>Mayfield (Mayfield et al. 2003)</strong></td>
<td>SVM + HMM + lex + pos + aff + pre + ort + chu + ws + quo</td>
<td>84.90 / 84.45 / 84.67</td>
<td>7</td>
</tr>
<tr>
<td><strong>Carreras (Carreras et al. 2003a)</strong></td>
<td>PER + lex + pos + aff + pre + ort + gaz + chu + ws + tri + bag</td>
<td>82.84 / 85.81 / 84.30</td>
<td>8</td>
</tr>
<tr>
<td><strong>McCallum (McCallum and Li 2003)</strong></td>
<td>CRF + lex + ort + gaz + ws</td>
<td>83.55 / 84.52 / 84.04</td>
<td>9</td>
</tr>
<tr>
<td><strong>Bender (Bender et al. 2003)</strong></td>
<td>MEMM + lex + pos + pre + ort + gaz + chu</td>
<td>83.18 / 84.68 / 83.92</td>
<td>10</td>
</tr>
<tr>
<td><strong>Munro (Munro et al. 2003)</strong></td>
<td>Voting + Bagging + lex + pos + aff + chu + cs + tri bag</td>
<td>84.21 / 80.87 / 82.50</td>
<td>11</td>
</tr>
<tr>
<td><strong>Wu (Wu et al. 2003)</strong></td>
<td>ADA (stacked 3 learners) + lex + pos + aff + pre + ort + gaz</td>
<td>81.39 / 82.02 / 81.70</td>
<td>12</td>
</tr>
<tr>
<td><strong>Whitelaw (Whitelaw and Patrick 2003)</strong></td>
<td>HMM + aff + pre + cs</td>
<td>78.05 / 81.60 / 79.78</td>
<td>13</td>
</tr>
<tr>
<td><strong>Hendricks (Hendrickx and Bosch 2003)</strong></td>
<td>MEM + lex + pos + aff + pre + ort + gaz + chu</td>
<td>80.17 / 76.33 / 78.20</td>
<td>14</td>
</tr>
<tr>
<td><strong>De Meulder (De Meulder and Daelemans 2003)</strong></td>
<td>MEM + lex + pos + aff + ort + gaz + chu + cs</td>
<td>78.13 / 75.84 / 76.97</td>
<td>15</td>
</tr>
<tr>
<td><strong>Hammerton (Hammerton 2003)</strong></td>
<td>RNN + lex + pos + gaz + chu</td>
<td>53.26 / 69.09 / 60.15</td>
<td>16</td>
</tr>
<tr>
<td><strong>Baseline</strong></td>
<td>50.90 / 71.91 / 59.61</td>
<td>17</td>
<td>31.86 / 28.89 / 30.30</td>
</tr>
<tr>
<td><strong>Giuliano (Giuliano et al. 2005)</strong></td>
<td>SVM + lex + ort + pos + ws + wv</td>
<td>76.70 / 90.50 / 83.10</td>
<td>* 11</td>
</tr>
<tr>
<td><strong>Talukdar (Talukdar et al. 2006)</strong></td>
<td>CRF + Three NE lists + Context pattern induction + tri + pruning</td>
<td>F-score = 84.52</td>
<td>* 8</td>
</tr>
<tr>
<td><strong>Wong (Wong and Ng 2007)</strong></td>
<td>MEMM + 300 million unlabeled tokens + lex + aff + pre + ort + cs</td>
<td>F-score = 87.13</td>
<td>* 3</td>
</tr>
</tbody>
</table>
2.6 Named Entity Recognition Challenges

In this section we summarize the named entity recognition challenges in different domains:

- The explosion of information raises the need for automated tools to extract meaningful entities and concepts from unstructured text in various domains.
- An entity that is relevant in one domain may not be irrelevant in another.
- Named entities may take any shape, often composed of multiple words. This raises more challenges in correctly identifying the beginning and the end of a multi-word NE.
- NER solutions are not easily portable across languages and domains, and the same system performs inconsistently in different contexts.
- General NER systems that have an F-score in the high 80’s and higher do not perform as well in the biomedical context, with F-score values lagging behind by 15 to 30 points.
- Manually annotating training data to be used with machine learning techniques is a labor expensive, time consuming, and error prone task.
- The extension of NER to challenging domains with multiple NE classes makes manual annotation very difficult to accomplish, especially with growing nomenclature.
- There is a growing need for systems that use semi-supervised or unsupervised machine learning technique in order to use mostly unannotated training data.
- Due to the large size of potential NER datasets in real-world applications, classification techniques need to be highly scalable.
• The quality of the annotated training data, the features and external resources used impact the overall recognition performance and accuracy.

• Extraction of language and domain-specific features requires additional processes.

• The effect of part-of-speech (POS) tagging on performance may be questionable. (Collier and Takeuchi 2004) note that simple orthographic features have consistently been proven to be more valuable than POS. This observation has been confirmed during phase One of this work (presented in Chapter 4).

• It is difficult to judge the efficacy of a given technique because of the different components used to construct the total solution. There is no consistent way to conclude whether a particular machine learning or other approach are best suited for NER. The quality of the recognition can only be seen as a whole.

In the following chapter, we briefly introduce the theory of support vector machines as our choice of machine learning method for the biomedical named entity recognition. Given the unique challenges in recognizing biomedical entities discussed earlier in this chapter, we decided to select a classification model that is capable of handling a high number of features and of discovering patterns in a large input space with irregular representation of classes. Support vector machines promise to handle both questions but not without challenges of their own.
Chapter 3

Support Vector Machines

In this chapter we present a brief summary of the Support Vector Machine (SVM) theory and its application in the area of named entity recognition. An introduction to the mathematical foundation of support vector machines for binary classification is presented, followed by an overview of the different approaches used for multi-class problems. We then discuss the scalability issues of support vector machines and how they have been addressed in the literature.

3.1 Support Vector Machines

The Support Vector Machine (SVM) is a powerful machine learning tool based on firm statistical and mathematical foundations concerning generalization and optimization theory. It offers a robust technique for many aspects of data mining including classification, regression, and outlier detection. SVM was first suggested by Vapnik in the early 1970’s but it began to gain popularity in the mid-1990’s. SVM is based on Vapnik’s statistical learning theory (Vapnik 1998) and falls at the intersection of kernel methods and maximum margin classifiers. Support vector machines have been successfully applied to many real-world problems such as face detection, intrusion detection, handwriting recognition, information extraction, and others.

Support Vector Machine is an attractive method due to its high generalization capability and its ability to handle high-dimensional input data. Compared to neural networks or decision trees, SVM does not suffer from the local minima problem, it has fewer learning parameters to select, and it produces stable and reproducible results. If two
SVMs are trained on the same data with the same learning parameters, they produce the same results independent of the optimization algorithm they use. However, SVMs suffer from slow training especially with non-linear kernels and with large input data size.

Support vector machines are primarily binary classifiers. Extensions to multi-class problems are most often done by combining several binary machines in order to produce the final multi-classification results. The more difficult problem of training one SVM to classify all classes uses much more complex optimization algorithms and are much slower to train than binary classifiers.

In the following sections, we present the SVM mathematical foundation for the binary classification case, then discuss the different approaches applied for multi-classification.

### 3.2 Binary Support Vector Classification

Binary classification is the task of classifying the members of a given set of objects into two groups on the basis of whether they have some property or not. Many applications take advantage of binary classification tasks, where the answer to some question is either a yes or no. For example, product quality control, automated medical diagnosis, face detection, intrusion detection, or finding matches to a specific class of objects.

The mathematical foundation of Support Vector Machines and the underlying Vapnik-Chervonenkis dimension (VC Dimension) is described in details in the literature covering the statistical learning theory (Vapnik 1998; Abe 2005; Müller et al. 2001; Kecman 2001; Joachims 2002; Alpaydin 2004) and many other sources. In this section we briefly introduce the mathematical background of SVMs in the linearly separable and
non-linearly separable cases. One of the attractive properties of support vector machines is the geometric intuition of its principles where one may relate the mathematical interpretation to simpler geometric analogies.

### 3.2.1 Linearly Separable Case

In the linearly separable case, there exists one or more hyperplanes that may separate the two classes represented by the training data with 100% accuracy. Figure 3.1(a) shows many separating hyperplanes (in the case of a two-dimensional input the hyperplane is simply a line). The main question is how to find the optimal hyperplane that would maximize the accuracy on the test data. The intuitive solution is to maximize the gap or margin separating the positive and negative examples in the training data. The optimal hyperplane is then the one that evenly splits the margin between the two classes, as shown in Figure 3.1(b).

![Figure 3.1 – SVM Linearly Separable Case](image)
In Figure 3.1(b), the data points that are closest to the separating hyperplane are called support vectors. In mathematical terms, the problem is to find $f(x) = (w^T x_i + b)$ with maximal margin, such that:

\[
\begin{align*}
    w^T x_i + b &= 1 \quad \text{for data points that are support vectors} \\
    w^T x_i + b &> 1 \quad \text{for other data points}
\end{align*}
\]

Assuming a linearly separable dataset, the task of learning coefficients $w$ and $b$ of support vector machine $f(x) = (w^T x_i + b)$ reduces to solving the following constrained optimization problem:

find $w$ and $b$ that minimize:

\[
\frac{1}{2} ||w||^2 
\]

subject to:

\[
y_i(w^T x_i + b) \geq 1, \quad \forall i
\]

Note that minimizing the inverse of the weights vector is equivalent to maximizing $f(x)$.

This optimization problem can be solved by using the unconstrained Lagrange function defined as:

\[
Q(w, b, \alpha) = \frac{1}{2} w^T w - \sum_{i=1}^{N} \alpha_i [y_i (w^T x_i + b) - 1] \quad \text{s.t. } \alpha_i \geq 0, \forall i
\]

where $\alpha_1, \alpha_2, \ldots, \alpha_N$ are Lagrange multipliers and $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_N]^T$.

The support vectors are those data points $x_i$ with $\alpha_i > 0$, i.e., the data points within each class that are the closest to the separation margin.

Solving for the necessary optimization conditions results in

\[
w = \sum_{i=1}^{N} \alpha_i y_i x_i \quad \text{where, } \sum_{i=1}^{N} \alpha_i y_i = 0
\]
By replacing $w = \sum_{i=1}^{N} \alpha_i y_i x_i$ into the Lagrange function and by using $\sum_{i=1}^{N} \alpha_i y_i = 0$ as a new constraint, the original optimization problem can be rewritten as its equivalent unconstrained dual problem as follows:

Find $\alpha$ that maximizes

$$\sum_{i} \alpha_i - \frac{1}{2} \sum_{i} \sum_{j} \alpha_i \alpha_j y_i y_j x_i^T x_j \quad \ldots \quad \ldots \quad (5)$$

subject to

$$\sum_{i=1}^{N} \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad \forall i \quad \ldots \quad \ldots \quad (6)$$

The optimization problem is therefore a convex quadratic programming problem which has global minimum. This characteristic is a major advantage of support vector machines as compared to neural networks or decision trees. The optimization problem can be solved in $O(N^3)$ time, where $N$ is the number of input data points.

3.2.2 Non-Linearly Separable Case

In the non-linearly separable case, it is not possible to find a linear hyperplane that separates all positive and negative examples. To solve this case, the margin maximization technique may be relaxed by allowing some data points to fall on the wrong side of the margin, i.e., to allow a degree of error in the separation. Slack Variables $\xi_i$ are introduced to represent the error degree for each input data point. Figure 3.2 demonstrates the non-linearly separable case where data points may fall into one of three possibilities:

1. Points falling outside the margin that are correctly classified, with $\xi_i = 0$
2. Points falling inside the margin that are still correctly classified, with $0 < \xi_i < 1$
3. Points falling outside the margin and are incorrectly classified, with $\xi_i = 1$
Figure 3.2 – SVM Non-Linearily Separable Case

If all slack variables have a value of zero, the data is linearly separable. For the non-linearily separable case, some slack variables have nonzero values. The optimization goal in this case is to maximize the margin while minimizing the points with $\xi_i \neq 0$, i.e., to minimize the margin error.

In mathematical terms, the optimization goal becomes:

find $w$ and $b$ that minimize: $\frac{1}{2}||w||^2 + C \sum_i \xi_i^2$ ........ (7)

s.t. $y_i (w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i$ ........ (8)

where $C$ is an user-defined parameter to enforce that all slack variables are as close to zero as possible. Finding the most appropriate choice for $C$ will depend on the input data set in use and requires tuning.

As in the linearly separable problem, this optimization problem can be converted to its dual problem:

find $\alpha$ that maximizes $\sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i^T x_j$ ........ (9)
In order to solve the non-linearly separable case, SVM introduces the use of a mapping function \( \Phi: \mathbb{R}^M \rightarrow F \) to translate the non-linear input space into a higher dimension feature space where the data is linearly separable. Figure 3.3 presents an example of the effect of mapping the nonlinear input space into a higher dimension linear feature space.

The dual problem is solved in feature space where its aim becomes to:

find \( \alpha \) that maximizes
\[
\sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \Phi(x_i)^T \Phi(x_j) \quad \ldots \ldots \quad (11)
\]

s.t.
\[
\sum_{i=1}^{N} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad \forall i \quad \ldots \ldots \quad (12)
\]

the resulting SVM is of the form:
\[
f(x) = w^T \Phi(x) + b = \sum_{i=1}^{N} \alpha_i y_i \Phi(x_i)^T \Phi(x) + b
\]
3.2.3 The Kernel “Trick”

Mapping the input space into a higher dimension feature space transforms the nonlinear classification problem into a linear one that is more likely to be solved. However, the problem is more likely to face the curse of dimensionality. The kernel “trick” allows the computation of the vector product $\Phi(x_i)^T\Phi(x_j)$ in the lower dimension input space.

From Mercer’s theorem, there is a class of mappings $\Phi$ such that $\Phi(x)^T\Phi(y) = K(x,y)$, where $K$ is a corresponding kernel function. Being able to compute the vector products in the lower dimension input space while solving the classification problem in the linearly separable feature space is a major advantage of SVMs using a kernel function. The dual problem then becomes to:

$$\text{find } \alpha \text{ that maximizes } \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i, x_j) \quad \ldots \quad \ldots \quad (13)$$

subject to

$$\sum_{i=1}^{N} \alpha_i y_i = 0,$$
$$0 \leq \alpha_i \leq C, \quad \forall i \quad \ldots \quad \ldots \quad (14)$$

and the resulting SVM takes the form:

$$f(x) = w^T \Phi(x_i) + b = \sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + b \quad \ldots \quad \ldots \quad (15)$$

Examples of kernel functions:

- Linear kernel (identity kernel): $K(x, y) = (x^T y)$
- Polynomial kernel with degree $d$: $K(x, y) = (x^T y + 1)^d$
- Radial basis kernel with width $\sigma$: $K(x, y) = e^{\frac{|x-y|^2}{\sigma^2}}$
• Sigmoid kernel with parameter $K$ and $\Theta$: $K(x, y) = \tanh(Kx^Ty + \Theta)$

• It’s also possible to use other kernel functions to solve specific problems

### 3.3 Multi-class Support Vector Classification

For classification problems with multiple classes, different approaches are developed in order to decide whether a given data point belongs to one of the classes or not. The most common approaches are those that combine *several* binary classifiers and use a voting technique to make the final classification decision. These include: One-Against-All (Vapnik 1998), One-Against-One (Kreßel 1999), Directed Acyclic Graph (DAG) (Platt et al. 2000), and Half-against-half method (Lei and Govindaraju 2005). A more complex approach is one that attempts to build *one* Support Vector Machine that separates all classes at the same time. In this section we will briefly introduce these multi-class SVM approaches. Figure 3.4 compares the decision boundaries for three classes using a One-Against-All SVM, a One-Against-One SVM, and an All-Together SVM. The interpretation of these decision boundaries will be discussed as we define the training and classification techniques using each approach.

![Figure 3.4 – Comparison of Multi-Class Boundaries](image)
3.3.1 One-Against-All Multi-Class SVM

One-Against-All (Vapnik 1998) is the earliest and simplest multi-class SVM. For a $K$-class problem, it constructs $K$ binary SVMs. The $i$th SVM is trained with all the samples from the $i$th class against all the samples from the other classes. To classify a sample $x$, $x$ is evaluated by all of the $K$ SVMs and the label of the class that has the largest value of the decision function is selected.

For a $K$-class problem, One-Against-One maximizes $K$ hyperplanes separating each class from all the rest. Since all other classes are considered negative examples during training of each binary classifier, the hyperplane is optimized for one class only. As illustrated in Figure 3.4, unclassifiable regions exist when more than one classifier returns a positive classification for an example $x$ or when all classifiers evaluate $x$ as negative (Abe 2005).

3.3.2 One-Against-One or Pairwise SVM

One-Against-One (Kreßel 1999) constructs one binary machine between pairs of classes. For a $K$-class problem, it constructs $K(K-1)/2$ binary classifiers. To classify a sample $x$, each of $K(K-1)/2$ machines evaluate $x$ and casts a vote. Finally, the class with the most votes is chosen.

Since One-Against-One separates two classes at a time, the separating hyperplanes identified by this approach are tuned better than those found with One-Against-All (Figure 3.4). Unclassifiable regions exist only when all classifiers evaluate a sample $x$ as negative.
3.3.3 Directed Acyclic Graph SVM

Similar to One-Against-One SVM, Directly Acyclic Graph (DAG) (Platt et al. 2000) trains $K(K-1)/2$ binary classifiers for pairwise classes. To evaluate a sample $x$, this technique builds a DAG ordering the classes 1 through $K$ to make a decision. The sample $x$ is first evaluated by the first and the last classifier on the DAG and eliminates the lower vote from the DAG. The process is repeated until only one class remains and its label is chosen. Therefore, a decision is reached after $(K - 1)$ binary SVM evaluations. Unclassifiable regions are eliminated by excluding one class at a time.

3.3.4 Half-Against-Half SVM

Half-Against-Half multi-class SVM (Lei and Govindaraju 2005) is useful for problems where there is a close similarity between groups of classes. Figure 3.5 illustrates an example with six classes where a linear separation exist between a group of three classes and another group of three classes. Using Half-Against-Half SVM, a binary classifier is built that evaluates one group of classes against another group. The trained model consists of at most $2^{\lceil \log_2 K \rceil}$ binary SVMs.

To classify a sample $x$, this technique identifies the group of classes where the sample $x$ belongs, than continues to evaluate $x$ with a subgroup, and so on, until the final class label is found. The classification process is similar to a decision tree that requires $\lceil \log_2 K \rceil$ evaluations at most. Figure 3.5 illustrates a possible decision tree for the six classes.
3.3.5 All-Together or All-At-Once SVM

An All-Together multi-classification approach is computationally more expensive yet usually more accurate than all other multi-classification methods. Hsu and Lin (Hsu and Lin 2002) note that “as it is computationally more expensive to solve multi-class problems, comparisons of these methods using large-scale problems have not been seriously conducted.

This approach builds one SVM that maximizes all separating hyperplanes at the same time. Training data representing all classes is used to generate the trained model. With this approach, there are no unclassifiable regions as each data point belongs to some class represented in the training dataset. Figure 3.4 illustrates the elimination of unclassifiable regions in this case.

The All-Together multi-class SVM poses a complex optimization problem as it maximizes all decision functions at the same time (Crammer and Singer 2001).

---

Algorithms to decompose the problem (Hsu and Lin 2002) and to solve the optimization problem (Tsochantaridis et al. 2004) have been developed, however, the All-Together multi-class SVM approach remains a daunting task. The training time is very slow which makes the approach so far unusable for real-world problems with a large data set and/or a high number of classes.

3.4 Named Entity Recognition Using SVM

The extension of named entity recognition to new domains with high ambiguity and growing nomenclature raises concerns about the machine learning methodology that is most suitable for such domains. Support Vector Machine (SVM) is a promising technique due to its ability to handle a high number of features and to discover patterns in a large input space with irregular representation of classes. SVM is gaining popularity as the approach of choice for challenging data such as the biomedical NER data. Many systems that participated in the JNLPBA-04 challenge task to identify named entities in the GENIA biomedical corpus were able to achieve reasonable performance results using SVM either separately or in conjunction with other machine learning approaches.

Evidence exists that using high-dimensional orthographic and contextual features leads to good NER classification. In order to classify entities with varying shapes and ambiguous boundaries, input data vectors are usually sparse and high-dimensional. Due to its high generalization ability, SVM is capable of discovering patterns under these circumstances.

Multi-class problems remain a daunting task for SVM, especially with large data size and a high number of classes. For multi-word named entities, one needs to design a learning solution that identifies the beginning, middle, and end of a named entity with
accuracy. There is no known best approach to handle this situation. One possibility is to label all parts of a named entity with the same label in order to minimize the number of classes. Another approach is to train machines to identify either the beginning or the end then combine all classified outputs in order to label the whole sequence of words in the entity. The number of classes in the problem doubles using this method. If an All-Together multi-class SVM is used, more classes lead to a much more complex optimization problem and much longer training time.

Integration of prior domain knowledge into SVM training is currently achieved outside of learning and classification process. SVM handles input vectors and is not yet capable of handling general information types.

### 3.5 SVM Scalability Challenges

(Bennett and Campbell 2000) discuss the common usability and scalability issues of support vector machines. In this section we summarize the SVM scalability challenges noted in the literature, which include:

- Optimization requires $O(n^3)$ time and $O(n^2)$ memory for single class training, where $n$ is input size (depends on algorithm used). To address this issue, new optimization algorithms continue to be introduced (Joachims 2006; Serafini and Zanni 2005; Collobert et al. 2006b, 2006a).

- Multi-class training time is much higher, especially for All-Together optimization. Performance depends on approach used and deteriorates with more classes.

- Slow training, especially with non-linear kernels, may be addressed by:
  - Reducing the input data size: pruning, chunking, clustering
Reducing the number of support vectors: model decomposition and shrinking (Joachims 1999)

Reducing feature dimensionality: using a priori clustering (Barros de Almeida et al. 2000) or adaptive clustering (Boley and Cao 2004)

- SVM is impractical for large input datasets, especially with non-linear kernel functions.

In addition to the scalability issues, SVM shares similar usability challenges as neural networks and other machine learning techniques related to the selection of model parameters. Model parameters are often selected using a grid search, cross-validation, or heuristic-based methods. Selection of a suitable kernel function for the problem at hand is another designer-determined factor, which is also tackled using cross-validation, decision trees, or heuristics. While these issues are concerning SVM usability, as discussed in Chapter 4, using grid search and cross-validation to select the best model may be unfeasible with large datasets.

### 3.6 Emerging SVM Techniques

Recent developments in the area of support vector machines attempt to use SVM for unsupervised learning in order to answer the need for new techniques to deal with unannotated data. Support Vector Clustering (SVC) (Ben-Hur et al. 2001) maps the input data into a higher dimension feature space where it tries to find clusters of data by minimizing a sphere in the higher dimension space. SVC introduced a novel idea that takes advantage of SVM’s mapping, however, this technique remains applicable only to toy data or very small datasets and cannot be used with real-world problems. Attempts to improve and/or provide alternative methods for clustering using support vectors followed
(Lee and Daniels 2006; Lee 2006). This area of research would certainly benefit from improved SVM scalability options.

Other research activities to accelerate SVM training in binary classification (Tsang et al. 2005) and to introduce new kernels for classification (Sullivan and Luke 2007) are also introduced.

In the next chapter, we describe the NER/SVM experimentation architecture and report the results and observations we reached by constructing a set of biomedical named entity recognition experiments using support vector machines. These experiments were instrumental in providing good insight into the scalability and usability challenges when using SVM with large scale from the unique biomedical domain. Our research work is influenced by reviewing the literature and the problems identified during the exploration phase presented in the following chapter.
Chapter 4
Identifying Challenges in Large Scale NER

In this chapter we report on the investigative phase of this work, which consists of building the infrastructure necessary to conduct the research and of designing and implementing a series of experiments to investigate the potential challenges of large scale named entity recognition using support vector machines. The main motivation for conducting these baseline experiments is to identify the challenges in large scale named entity recognition problems, to assess the feasibility of the language and domain independent NER approach, and to obtain a set of baseline performance measures.

The baseline experiments attempt to eliminate language and domain-specific knowledge from the named entity recognition process when applied to the English biomedical entity recognition task, as a baseline for other languages and domains. The biomedical field NER remains a challenging task due to growing nomenclature, ambiguity in the left boundary of entities caused by descriptive naming, difficulty of manually annotating large sets of training data, strong overlap among different entities, to cite a few of the NER challenges in this domain.

In the following sections, we present the architecture and methods used to conduct the experiments. We then report the experiments results both in the case of single-class (protein) classification and for the multi-class classification results, and compare the results to those reported by other systems using the same challenge data. We conclude the chapter with a discussion of the challenges and problems faced while conducting these experiments which motivate our research work on scalability (and usability) issues.
4.1 Baseline Experiment Design

The baseline experiment aims to identify biomedical named entities using a supervised learning approach. The training and testing data use the JNLPBA-04 (Kim et al. 2004) challenge task data, where the names of proteins, cell lines, cell types, DNA and RNA entities are previously labeled. The approach employed in this experiment is the supervised machine learning using Support Vector Machines (SVM) (Vapnik 1998), due to their ability to handle high-dimensional feature and input space.

The JNLPBA-04 shared task (Kim et al. 2004) is an open challenge task proposed at the “International Joint Workshop on Natural Language Processing in Biomedicine and its Application”. The task uses the GENIA corpus (Kim et al. 2003) described above. The systems participating in this task employ a variety of machine learning techniques such as Support Vector Machines (SVM), Hidden Markov Models (HMM), Maximum Entropy Markov Models (MEMMs) and Conditional Random Fields (CRFs). Five systems adopted SVMs either in isolation (Park et al. 2004; Lee, Hwang et al. 2004), or in conjunction with other model (Zhou and Su 2004; Song et al. 2004; Rössler 2004). The results of the experiment presented in this paper are compared to the five JNLPBA-04 task participating systems listed above, in addition to the results reported in (Giuliano et al. 2005). Table 4.2 summarizes the performance comparison results.

This experiment is composed of two main parts – the first identifies protein named entities only, while the second locates and classifies all five named entities (protein, DNA, RNA, cell type, and cell line). The SVM-Light software by T. Joachims (Joachims 2002) is used for the single-class part of the experiment, and the SVM-Multiclass
software – also by T. Joachims – is used for the multi-class experiments. Table 4.3 and Table 4.4 summarize the experiment results of both parts.

The training and test data pre-processing involves morphological and contextual features extraction only. In order to estimate a worse-case scenario of the approach used, no instance pruning or filtering is performed prior to learning and classification, thereby leaving the scarcity nature of the data intact. No language-specific pre-processing such as part-of-speech or noun phrases tagging is used. No dictionaries, gazetteers (seed words), or other domain-specific knowledge are used. Figure 4.1 presents the architecture used to conduct the experiments. The common machine learning architecture used for NER is

Figure 4.1 – Baseline Experiments Architecture
simplified by limiting the pre-processing stage to feature extraction and reducing post-processing to class labeling and performance evaluation.

The initial results are promising and prove that the approach used is capable of recognizing and identifying the named entities successfully without making use of any language-specific or domain-specific prior knowledge. Performance measures of the experiments are reported in terms of recall, precision, and $F_{\beta=1}$-score.

### 4.2 Features Selection

The training and testing data is preprocessed using the JFEX software (Giuliano et al. 2005) in order to extract morphological and contextual features that do not use language-specific knowledge such as part-of-speech or noun phrase tagging. The generated feature space is very large, including about a million different features. The features extracted are described below. Since words appearing separately or within windows of other words each constitutes a feature in the lexicon, the potential number of possibilities is very high. Including character n-grams describing prefixes, infixes, and suffixes would further increase the number of features in the lexicon. The feature extraction process is intentionally designed that way in order to test the scalability of the approach used and to allow the experiments to proceed in a language-independent and domain-independent fashion. All features are binary, i.e., each feature denotes whether the current token possesses this feature (one) or not (zero). Character n-grams were not included in the baseline experiment data due to memory limitations encountered during the feature extraction process. The morphological features extracted are:

- Capitalization: token begins with a capital letter.
- Numeric: token is a numeric value.
• Punctuation: token is a punctuation.
• Uppercase: token is all in uppercase.
• Lowercase: token is all in lowercase.
• Single character: token length is equal to one.
• Symbol: token is a special character.
• Includes hyphen: one of the characters is a hyphen.
• Includes slash: one of the characters is a slash.
• Letters and Digits: token is alphanumeric.
• Capitals and digits: token contains caps and digits.
• Includes caps: some characters are in uppercase.
• General regular expression summarizing the word shape, for e.g., \(Xx^+X-x^+\) describes a word starting with one capital letter followed by a number of lowercase letter, then a hyphen, one capital letter, another hyphen, and ending with a number of lowercase letters.

The morphological features extracted examine a token as a whole and do not include character n-grams features that detect in-word characteristics such as prefixes, suffixes, or infixes. Morphological feature extraction is applied to the three tokens preceding and following the token being examined in addition to the token itself.

Each word appearing in the training text is considered its own feature. In addition, a consecutive collocation of tokens active over three positions around the token itself is used in order to provide a moving window of consecutive tokens which describes the context of the token relative to its surrounding.
Since biomedical named entities often are composed of more than one token, special labeling for the beginning, the middle, and the ending of a named entity sequence is often used. However, in the initial experiments, all tokens within a sequence use the same label during the SVM training phase. The output of the SVM classification phase is post-processed in order to label the beginning, middle, and end part of a sequence differently, as required by the JNLPBA-04 task evaluation scripts. The post-processing labeling scripts did not attempt to correct classification errors that may arise while identifying the beginning of a sequence, however, the overall performance results are still indicative of the feasibility of the approach used.

4.3 Single Class Results

Using SVM-Light (Joachims 2002), a single-class support vector machine is trained to recognize protein name sequences. The trained machine is then used to classify proteins in the test data. Performance results of the protein classification task are summarized in Table 4.3. The hardware configuration used for single-class classification experiments is a single-processor Windows-based Pentium IV machine with 1 GB of RAM. This configuration was enough to complete each single-class experiment successfully within 20-30 minutes.

Since no pre-processing was performed on the training and testing data besides features extraction, the positive examples in the data sets remained scarce. As a result, we consider the performance results reported in Table 4.3 to represent a worse-case indication of the potential performance.

SVM-Light (Joachims 2002) offers the option of “boosting” the weight of the positive examples relative to the negative ones. We experimented with boosting factors of 2, 4,
and 8 in order to counter the effect of positive data scarcity. The relative performance results are reported in Table 4.1 and Table 4.3. The overall recall, precision, and $F_{\beta=1}$-score measures achieved with the different boosting factors is as follows:

<table>
<thead>
<tr>
<th>Positive Boosting Factor</th>
<th>Recall</th>
<th>Precision</th>
<th>F$\beta=1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>No positive boosting</td>
<td>68.06</td>
<td>59.29</td>
<td>63.37</td>
</tr>
<tr>
<td>Positive boosting factor = 2</td>
<td>75.54</td>
<td>58.82</td>
<td>66.14</td>
</tr>
<tr>
<td>Positive boosting factor = 4</td>
<td>78.70</td>
<td>57.30</td>
<td>66.32</td>
</tr>
<tr>
<td>Positive boosting factor = 8</td>
<td>78.49</td>
<td>54.84</td>
<td>64.56</td>
</tr>
</tbody>
</table>

As expected, increasing the positive weight boosting factor led to an improved recall measure at the expense of the precision measure. The resulting $F_{\beta=1}$-score improved with a boosting factor of two and four relative to the experiment without any positive weight boosting. However, a further increase of the positive boosting factor to eight led to a decrease of the overall $F_{\beta=1}$-score, as a result of the decreasing precision measure. A careful balance of the recall and precision results is required in order to maintain an overall $F_{\beta=1}$-score, if such a measure is deemed to be the final performance indicator.

### 4.4 Multi-class Results

The $SVM-Multiclass$ implementation by T. Joachims is based on (Crammer and Singer 2001) and uses a different quadratic optimization algorithm described in (Tsochantaridis et al. 2004). The $SVM-Multiclass$ implementation uses an “All-Together” multi-classification approach, which is computationally more expensive yet usually more accurate than “One-Against-All” or “one-against-one” multi-classification methods. Hsu and Lin (Hsu and Lin 2002) note that “as it is computationally more expensive to solve multi-class problems, comparisons of these methods using large-scale problems have not been seriously conducted. Especially for methods solving multi-class SVM in one step, a much larger optimization problem is required so up to now
experiments are limited to small data sets.” The multi-class experiment presented in this paper is one such attempt at solving a large-scale problem using an “All-Together” classification method.

Initial experiments for multi-class classification were unsuccessful, mostly due to hitting the processing power limits of the single processor machines. The same experiments were attempted on different machine configurations, and unreasonably long processing time was needed to finally complete one such experiment. The first successful experiment required a total learning and classification time of 17 days in order to complete using a serial algorithm on a quad-processor Pentium IV machine. The same experiment was repeated on a Xeon quad-processor 3.6 GHz Linux machine with four gigabytes of main memory and completed in 97 hours or four days and one hour.

The multi-class performance results are summarized in Table 4.4. Detailed multi-class results are presented in Table 4.5, Table 4.6, Table 4.7, and Table 4.8. The overall recall measure achieved is 62.43%, with a precision of 64.50%, and a final F-score of 63.45%. These results are compared to those obtained by the five JNLPBA-04 participating systems which used support vector machines either in isolation or in combination with other models, as well as the results reported by (Giuliano et al. 2005) using the same task data. The performance comparison results are reported in Table 4.2. The language-independent approach used in this experiment performed very close to (Park et al. 2004) and better than (Lee, Hwang et al. 2004) which both used SVM as the only learning model. Park et al. (Park et al. 2004) used character n-grams, orthographic information, word shapes, gene sequences prior knowledge, word variations, part-of-speech tags, noun phrase tags, and word triggers. Lee et al. (Lee, Hwang et al. 2004)
used lexical features, character n-grams, and part-of-speech tags in a two-phased model based on SVMs.

Rössler (Rössler 2004) adapted a NER-system for German to the biomedical field. The system used character n-grams, orthographic information, gene sequences prior knowledge, and word length as features. The overall performance of (Rössler 2004) is very close to that achieved in this experiment. The approach used by (Rössler 2004) is particularly interesting in demonstrating the applicability of some NER-system from one language to another by not incorporating language-specific features. However, Rössler (Rössler 2004) made use of domain-specific knowledge while applying the system to the biomedical domain.

Zhou and Su (Zhou and Su 2004) developed the system that performed best in the JNLPBA-04 task. Their system performance reached an overall recall/precision/F-score of 76.0%, 69.4%, and 72.6% respectively. Zhou and Su (Zhou and Su 2004) used SVM in conjunction with Hidden Markov Models in a more complex learning method. The systems also made use of many language-specific and domain-specific knowledge such as character n-grams, orthographic information, gene sequences, gazetteers, part-of-speech tags, word triggers, abbreviations, and cascaded entities. While this system performed better than the current multi-class experiment, its heavy use of language and domain-specific prior knowledge contradicts the promise of the approach presented in this research work. The baseline performance of (Zhou and Su 2004) without post-processing was 60.3%.

Song et al. (Song et al. 2004) used SVM in combination with Conditional Random Fields (CRFs) and included character n-grams, orthographic information, and other
lexical features in addition to part-of-speech and noun phrase tagging. The overall performance of this system is comparable to that of the multi-class experiment results hereby presented. Giuliano et al. (Giuliano et al. 2005) also incorporated part-of-speech tagging and word features of tokens surrounding each analyzed token in addition to features similar to those used in this experiment. In addition, Giuliano et al. (Giuliano et al. 2005) pruned the data instances in order to reduce the dataset size by filtering out frequent words from the corpora because they are less likely to be relevant than rare words.

Table 4.2 – Performance of BioNLP Systems Using SVM vs. Baseline Results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhou (Zhou and Su 2004)</td>
<td>75.3 / 69.5 / 72.3</td>
<td>77.1 / 69.2 / 72.9</td>
<td>75.6 / 71.3 / 73.8</td>
<td>75.8 / 69.5 / 72.5</td>
<td>76.0 / 69.4 / 72.6</td>
</tr>
<tr>
<td>Giuliano (Giuliano et al. 2005)</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>64.4 / 69.8 / 67.0</td>
</tr>
<tr>
<td>Song (Song et al. 2004)</td>
<td>60.3 / 66.2 / 63.1</td>
<td>71.2 / 65.6 / 68.2</td>
<td>69.5 / 65.8 / 67.6</td>
<td>68.3 / 64.0 / 66.1</td>
<td>67.8 / 64.8 / 66.3</td>
</tr>
<tr>
<td>Rössler (Rössler 2004)</td>
<td>59.2 / 60.3 / 59.8</td>
<td>70.3 / 61.8 / 65.8</td>
<td>68.4 / 61.5 / 64.8</td>
<td>68.3 / 60.4 / 64.1</td>
<td>67.4 / 61.0 / 64.0</td>
</tr>
<tr>
<td>Habib (baseline)</td>
<td>53.2 / 70.8 / 60.7</td>
<td>63.7 / 63.6 / 63.7</td>
<td>64.2 / 65.4 / 64.8</td>
<td>63.0 / 63.2 / 63.1</td>
<td>62.3 / 64.5 / 63.4</td>
</tr>
<tr>
<td>Park (Park et al. 2004)</td>
<td>62.8 / 55.9 / 59.2</td>
<td>70.3 / 61.4 / 65.6</td>
<td>65.1 / 60.4 / 62.7</td>
<td>65.9 / 59.7 / 62.7</td>
<td>66.5 / 59.8 / 63.0</td>
</tr>
<tr>
<td>Lee (Lee, Hwang et al. 2004)</td>
<td>42.5 / 42.0 / 42.2</td>
<td>52.5 / 49.1 / 50.8</td>
<td>53.8 / 50.9 / 52.3</td>
<td>52.3 / 48.1 / 50.1</td>
<td>50.8 / 47.6 / 49.1</td>
</tr>
<tr>
<td>Baseline (Kim et al. 2004)</td>
<td>47.1 / 33.9 / 39.4</td>
<td>56.8 / 45.5 / 50.5</td>
<td>51.7 / 46.3 / 48.8</td>
<td>52.6 / 46.0 / 49.1</td>
<td>52.6 / 43.6 / 47.7</td>
</tr>
</tbody>
</table>

Table 4.3 – Effect of Positive Examples Boosting on Single-Class SVM Results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No Boosting</td>
<td>57.47 / 53.35 / 55.34</td>
<td>71.69 / 62.76 / 66.93</td>
<td>68.35 / 59.60 / 63.68</td>
<td>68.27 / 58.63 / 63.08</td>
<td>68.06 / 59.29 / 63.37</td>
</tr>
<tr>
<td>Complete</td>
<td>Right</td>
<td>73.56 / 68.29 / 70.83</td>
<td>81.76 / 71.58 / 76.33</td>
<td>79.04 / 68.92 / 73.63</td>
<td>79.47 / 68.25 / 73.43</td>
</tr>
<tr>
<td></td>
<td>Left</td>
<td>59.61 / 55.34 / 57.39</td>
<td>77.82 / 68.13 / 72.65</td>
<td>76.70 / 66.88 / 71.45</td>
<td>76.08 / 65.34 / 70.30</td>
</tr>
<tr>
<td>Boost Factor = 2</td>
<td>Complete</td>
<td>65.35 / 50.83 / 57.18</td>
<td>78.59 / 60.88 / 68.61</td>
<td>76.33 / 60.36 / 67.41</td>
<td>75.58 / 58.40 / 65.89</td>
</tr>
<tr>
<td>Left</td>
<td>Right</td>
<td>80.62 / 62.71 / 70.55</td>
<td>87.75 / 67.98 / 76.61</td>
<td>86.28 / 68.23 / 76.20</td>
<td>86.25 / 66.65 / 75.19</td>
</tr>
<tr>
<td></td>
<td>67.98 / 52.87 / 59.48</td>
<td>85.70 / 66.39 / 74.82</td>
<td>83.67 / 66.16 / 73.89</td>
<td>83.21 / 64.30 / 72.54</td>
<td>82.57 / 64.30 / 72.30</td>
</tr>
<tr>
<td>Boost Factor = 4</td>
<td>Complete</td>
<td>71.43 / 48.49 / 57.77</td>
<td>81.55 / 59.32 / 68.68</td>
<td>78.99 / 59.03 / 67.57</td>
<td>78.63 / 57.04 / 66.11</td>
</tr>
<tr>
<td>Left</td>
<td>Right</td>
<td>84.73 / 57.53 / 68.53</td>
<td>89.93 / 65.42 / 75.74</td>
<td>88.58 / 66.20 / 75.77</td>
<td>88.64 / 64.30 / 74.53</td>
</tr>
<tr>
<td></td>
<td>74.38 / 50.50 / 60.16</td>
<td>88.66 / 64.50 / 74.67</td>
<td>86.10 / 64.35 / 73.65</td>
<td>86.00 / 62.39 / 72.31</td>
<td>85.58 / 62.31 / 72.11</td>
</tr>
<tr>
<td>Boost Factor = 8</td>
<td>Complete</td>
<td>71.43 / 44.52 / 54.85</td>
<td>81.41 / 57.14 / 67.15</td>
<td>78.76 / 56.91 / 66.08</td>
<td>78.34 / 54.74 / 64.45</td>
</tr>
<tr>
<td>Left</td>
<td>Right</td>
<td>85.06 / 53.02 / 65.32</td>
<td>90.14 / 63.27 / 74.35</td>
<td>88.21 / 63.74 / 74.00</td>
<td>88.42 / 61.78 / 72.73</td>
</tr>
<tr>
<td></td>
<td>73.89 / 46.06 / 56.75</td>
<td>88.59 / 62.18 / 73.08</td>
<td>86.47 / 62.48 / 72.54</td>
<td>86.44 / 60.39 / 71.11</td>
<td>85.83 / 60.00 / 70.63</td>
</tr>
</tbody>
</table>
### Table 4.4 – Summary of Baseline Multi-Class Experiment Results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>protein</td>
<td>58.62 / 70.83 / 64.15</td>
<td>72.68 / 63.43 / 67.74</td>
<td>70.83 / 62.03 / 66.14</td>
<td>71.28 / 60.19 / 65.27</td>
<td>70.37 / 62.00 / 65.92</td>
</tr>
<tr>
<td>DNA</td>
<td>61.61 / 65.71 / 63.59</td>
<td>72.68 / 63.01 / 57.10</td>
<td>52.55 / 70.59 / 60.25</td>
<td>47.11 / 69.60 / 56.19</td>
<td>51.00 / 67.64 / 58.16</td>
</tr>
<tr>
<td>RNA</td>
<td>0.00 / 0.00 / 0.00</td>
<td>55.10 / 57.45 / 56.25</td>
<td>50.00 / 74.29 / 59.77</td>
<td>50.00 / 62.50 / 55.56</td>
<td>51.16 / 63.77 / 57.02</td>
</tr>
<tr>
<td>cell type</td>
<td>51.79 / 73.55 / 60.78</td>
<td>51.42 / 72.84 / 60.28</td>
<td>52.94 / 82.89 / 64.62</td>
<td>50.09 / 81.31 / 61.99</td>
<td>59.56 / 78.00 / 67.55</td>
</tr>
<tr>
<td>cell line</td>
<td>32.39 / 67.86 / 43.85</td>
<td>50.00 / 50.60 / 50.30</td>
<td>56.94 / 55.03 / 55.97</td>
<td>53.53 / 43.75 / 48.15</td>
<td>47.72 / 51.64 / 49.61</td>
</tr>
<tr>
<td>Overall</td>
<td>53.18 / 70.79 / 60.73</td>
<td>63.68 / 63.63 / 63.66</td>
<td>64.15 / 65.39 / 64.76</td>
<td>62.97 / 63.16 / 63.06</td>
<td>64.23 / 64.50 / 63.45</td>
</tr>
</tbody>
</table>

### Table 4.5 – Baseline Multi-Class Experiment Results 1978-1989 Set

<table>
<thead>
<tr>
<th>named entity</th>
<th>complete match</th>
<th>right boundary match</th>
<th>left boundary match</th>
</tr>
</thead>
<tbody>
<tr>
<td>protein</td>
<td>357 (58.62 / 70.83 / 64.15)</td>
<td>464 (76.19 / 92.06 / 83.38)</td>
<td>371 (60.92 / 73.61 / 66.67)</td>
</tr>
<tr>
<td>DNA</td>
<td>69 (61.61 / 65.71 / 63.59)</td>
<td>96 (85.71 / 91.43 / 88.48)</td>
<td>73 (65.18 / 69.52 / 67.28)</td>
</tr>
<tr>
<td>RNA</td>
<td>0 (0.00 / 0.00 / 0.00)</td>
<td>0 (0.00 / 0.00 / 0.00)</td>
<td>0 (0.00 / 0.00 / 0.00)</td>
</tr>
<tr>
<td>cell type</td>
<td>203 (51.79 / 73.55 / 60.78)</td>
<td>273 (69.64 / 98.91 / 81.74)</td>
<td>215 (54.85 / 77.90 / 64.37)</td>
</tr>
<tr>
<td>cell line</td>
<td>57 (32.39 / 67.86 / 43.85)</td>
<td>90 (51.14 / 107.14 / 69.23)</td>
<td>65 (36.93 / 77.38 / 50.00)</td>
</tr>
<tr>
<td>[-ALL-]</td>
<td>686 (53.18 / 70.79 / 60.73)</td>
<td>923 (71.55 / 95.25 / 81.72)</td>
<td>724 (56.12 / 74.72 / 64.10)</td>
</tr>
</tbody>
</table>

### Table 4.6 – Baseline Multi-Class Experiment Results 1990-1999 Set

<table>
<thead>
<tr>
<th>named entity</th>
<th>complete match</th>
<th>right boundary match</th>
<th>left boundary match</th>
</tr>
</thead>
<tbody>
<tr>
<td>protein</td>
<td>1032 (72.68 / 63.43 / 67.74)</td>
<td>1200 (84.51 / 73.76 / 78.77)</td>
<td>1129 (79.51 / 69.39 / 74.11)</td>
</tr>
<tr>
<td>DNA</td>
<td>201 (52.21 / 63.01 / 57.10)</td>
<td>288 (74.81 / 90.28 / 81.82)</td>
<td>232 (60.26 / 72.73 / 65.91)</td>
</tr>
<tr>
<td>RNA</td>
<td>27 (55.10 / 57.45 / 56.25)</td>
<td>41 (83.67 / 87.23 / 85.42)</td>
<td>31 (63.27 / 65.96 / 64.58)</td>
</tr>
<tr>
<td>cell type</td>
<td>236 (51.42 / 72.84 / 60.28)</td>
<td>325 (70.81 / 100.31 / 83.01)</td>
<td>256 (55.77 / 79.01 / 65.39)</td>
</tr>
<tr>
<td>cell line</td>
<td>84 (50.00 / 50.60 / 50.30)</td>
<td>117 (69.64 / 70.48 / 70.06)</td>
<td>97 (57.74 / 58.43 / 58.08)</td>
</tr>
<tr>
<td>[-ALL-]</td>
<td>1580 (63.68 / 63.63 / 63.66)</td>
<td>1971 (79.44 / 79.38 / 79.41)</td>
<td>1745 (70.33 / 70.28 / 70.31)</td>
</tr>
</tbody>
</table>

### Table 4.7 – Baseline Multi-Class Experiment Results 2000-2001 Set

<table>
<thead>
<tr>
<th>named entity</th>
<th>complete match</th>
<th>right boundary match</th>
<th>left boundary match</th>
</tr>
</thead>
<tbody>
<tr>
<td>protein</td>
<td>1544 (70.83 / 62.03 / 66.14)</td>
<td>1815 (83.26 / 72.92 / 77.75)</td>
<td>1723 (79.04 / 69.22 / 73.81)</td>
</tr>
<tr>
<td>DNA</td>
<td>216 (52.55 / 70.59 / 60.25)</td>
<td>284 (69.10 / 92.81 / 79.22)</td>
<td>241 (58.64 / 78.76 / 67.22)</td>
</tr>
<tr>
<td>RNA</td>
<td>26 (50.00 / 74.29 / 59.77)</td>
<td>40 (76.92 / 114.29 / 91.95)</td>
<td>26 (50.00 / 74.29 / 59.77)</td>
</tr>
<tr>
<td>cell type</td>
<td>378 (52.94 / 82.89 / 64.62)</td>
<td>513 (71.85 / 112.50 / 87.69)</td>
<td>402 (56.30 / 88.16 / 68.72)</td>
</tr>
<tr>
<td>cell line</td>
<td>82 (56.94 / 55.03 / 55.97)</td>
<td>112 (77.78 / 75.17 / 76.43)</td>
<td>92 (63.89 / 61.74 / 62.80)</td>
</tr>
<tr>
<td>[-ALL-]</td>
<td>2246 (64.15 / 65.39 / 64.76)</td>
<td>2764 (78.95 / 80.47 / 79.70)</td>
<td>2484 (70.95 / 72.31 / 71.63)</td>
</tr>
</tbody>
</table>

### Table 4.8 – Baseline Multi-Class Experiment Results 1998-2001 Set

<table>
<thead>
<tr>
<th>named entity</th>
<th>complete match</th>
<th>right boundary match</th>
<th>left boundary match</th>
</tr>
</thead>
<tbody>
<tr>
<td>protein</td>
<td>2271 (71.28 / 60.19 / 65.27)</td>
<td>2666 (83.68 / 70.66 / 76.62)</td>
<td>2526 (79.28 / 66.95 / 72.60)</td>
</tr>
<tr>
<td>DNA</td>
<td>277 (47.11 / 69.60 / 56.19)</td>
<td>386 (65.65 / 96.98 / 78.30)</td>
<td>312 (53.06 / 78.39 / 63.29)</td>
</tr>
<tr>
<td>RNA</td>
<td>35 (50.00 / 62.50 / 55.56)</td>
<td>53 (75.71 / 94.64 / 84.13)</td>
<td>36 (51.43 / 64.29 / 57.14)</td>
</tr>
<tr>
<td>cell type</td>
<td>570 (50.09 / 81.31 / 61.99)</td>
<td>802 (70.47 / 114.41 / 87.22)</td>
<td>612 (53.78 / 87.30 / 66.56)</td>
</tr>
<tr>
<td>cell line</td>
<td>91 (53.53 / 43.75 / 48.15)</td>
<td>132 (77.65 / 63.46 / 69.84)</td>
<td>104 (61.18 / 50.00 / 55.03)</td>
</tr>
<tr>
<td>[-ALL-]</td>
<td>3244 (62.97 / 63.16 / 63.06)</td>
<td>4039 (78.40 / 78.64 / 78.52)</td>
<td>3590 (69.68 / 69.90 / 69.79)</td>
</tr>
</tbody>
</table>
4.5 Challenges and Problems

Running the baseline experiments provided a good first-hand experience and exposure to the usability and scalability challenges associated with named entity recognition and support vector machines. In this section we describe some of the issues encountered as they provide the motivation for our research work. We begin with a discussion of the scalability challenges then proceed with some of the practical problems and usability issues faced.

The Java-based JFEX feature extraction system provided by (Giuliano et al. 2005) provides a scripted approach to define which features are to be extracted, which adds flexibility to the feature extraction process. However, the system memory requirements are very high, especially with a large dataset and a selection of several complex features. The memory requirements can be reduced by breaking down the datasets into smaller sets grouped in a folder, yet similar memory needs reduction was not possible when complex features were to be extracted. For the sake of this experiment, word shape features for surrounding tokens could not be included due to the memory limitation. Also, extraction of character n-grams was not included in this system. The resulting set of features used in this experiment mostly includes simple orthographic information, contextual features, and the words themselves.

Different machine configurations were tried during the course of this research. Desktops and laptops with Pentium IV processors and 1-2 GB of RAM were used for single-class classification experiments. Attempts to run multi-class experiments failed on single processor machines due to the CPU-intensive nature of the classification process. The later experiment was tried on a Pentium III quad-processor Linux machine. The
process did not complete in 22 days and was aborted. The first successful multi-class experiment that was run on a Pentium IV quad-processor Linux-based machine completed in 17 days. Running the same experiment on a Xeon quad-processor 3.6GHz machine completed in 96 hours (4 days). Table 4.9 summarizes the training time and performance results for some single class tests with different kernel types and a multi-class test with a linear kernel. All tests reported in Table 4.9 were run on a dual core Xeon 3.6 GHz machine with 4 GB of RAM selecting a margin error of 0.1 and a maximum of 2GB of total memory per process.

Table 4.9 – Single and Multi-class Training Times

<table>
<thead>
<tr>
<th>Test Type</th>
<th>Kernel Type</th>
<th>Training Time</th>
<th>Recall</th>
<th>Precision</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protein</td>
<td>Linear</td>
<td>814.85 sec. 13.58 min.</td>
<td>68.13</td>
<td>59.33</td>
<td>63.43</td>
</tr>
<tr>
<td>Protein</td>
<td>Polynomial degree=2</td>
<td>390082.24 sec. 6501.37 min.</td>
<td>69.93</td>
<td>62.23</td>
<td>65.86</td>
</tr>
<tr>
<td>Protein</td>
<td>Polynomial degree=3</td>
<td>78506.33 sec. 1308.44 min.</td>
<td>68.47</td>
<td>62.44</td>
<td>65.32</td>
</tr>
<tr>
<td>Protein</td>
<td>Radial basis</td>
<td>--</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Multi-class</td>
<td>Linear</td>
<td>353367.03 sec. 5889.45 min.</td>
<td>70.09 (P)</td>
<td>61.93 (P)</td>
<td>65.76 (P)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>63.01 (A)</td>
<td>64.44 (A)</td>
<td>63.71 (A)</td>
</tr>
</tbody>
</table>

(P): Identified protein names; (A): All identified entities

Some initial parallelization single-class experiments were attempted successfully on a cluster of Linux machines at the US Air Force Academy (USAFA). The parallel SVM software developed by Zanni et al. (Zanni et al. 2006) based on (Zanghirati and Zanni 2003; Serafini et al. 2005) was used. However, the initial performance measures achieved by the serial SVM software were superior to those attained by the parallel algorithm, with no considerable gain in computational time. Further experiments are needed in order to determine the best use of the parallel SVM software in (Zanni et al. 2006) or other parallel algorithms.
Most of the NER and SVM usability issues were experienced during this exercise. Model parameters and kernel selection, identification of a suitable class labeling convention, developing preprocessing and post-processing modules to suit the needs of different tools, and not being able to capitalize on results obtained from different experiments without having to restart the learning process, were just a few examples of the common challenges facing developers of machine learning solutions in general.

Designing and building the experimentation infrastructure was the main practical problem faced. In order to compile and/or construct the resources needed for the machine learning solution, it was evident that in order to contribute to the research activity one has to spend a considerable amount of time and effort recreating existing solutions before getting to the point where more focused research and development could be possible. Despite the existence of previously developed tools within the research community, these tools lack standardization and require extensive work to preprocess the data according to different formats and to build interface modules to link the tools and construct a total solution. The problem is compounded by the lack of compatible components and reliable documentation on the type of preprocessing that may be needed by one or more of these components. In order to build a new research infrastructure from scratch, one needs to reproduce previous work to get to the point where new research may be started. This experience demonstrates the need for a complete yet flexible research infrastructure that facilitates the construction of new solutions from existing components and enables the addition of new research in focused areas. This is a software engineering issue in the first place and is often overlooked in the excitement of a challenging research activity.
In summary, we view the issues associated with large-scale named entity recognition using support vector machines as a two-fold problem: one of software engineering and one of machine learning, and recommend that they should be addressed as such. Although this research proposal will focus primarily on improving SVM’s scalability with large datasets and multiple classes, we recommend a new service-oriented framework to address the usability, maintainability, and expandability issues observed while running the baseline experiments. Some initial thoughts about the framework are presented in Chapter 10. We hope to explore and implement these ideas in future research projects.

In Chapter 6, we present a novel database-supported framework for classification using support vector machines, to be applied to the biomedical named entity recognition problem in particular.

### 4.6 NER/SVM Scalability Challenges

In this section we summarize the various usability and scalability challenges discussed in the previous two chapters. The NER/SVM problem combines questions inherent in both fields and compounds the software engineering issue. The biomedical NER using SVM poses several challenges which include:

- Higher level of ambiguity and uncertainly regarding biomedical entity shapes.
- Difficulty to annotate training data manually, a labor and time expensive process.
- Need for high-dimensional features to compensate for ambiguity in defining entities.
- SVM slow training and high memory requirements.
- Difficulty to select suitable SVM model parameters and kernel for given datasets.
• Complex learning with training data that represent multiple classes.
• Lack of integrated tools to build a total solution.
• Incompatibility of existing tools and the need to develop interfacing modules to fill the gaps and put the different pieces of the solution together.

A language and domain-independent named entity recognition system requires elimination of prior language or domain-specific knowledge. Therefore, any proposed solution needs to address a higher level of ambiguity and minimize expectations about the shape of named entities. Entities may take any shape or form and patterns may be difficult to discover. Entities are often composed of multiple words with unclear boundaries. When more than one named entity type is to be identified, one may face an unbalanced distribution of the entities belonging to different classes. Positive examples of such entities often appear scarcely and infrequently in the text.

Building large annotated corpora for a given language or domain is a difficult labor and time expensive task. The task requires manual annotation of unstructured data which may be inconsistent and error-prone. With the growing applications of information extraction, the need for using unannotated data or a mixture of both labeled and unlabeled data increases. Unsupervised or semi-supervised solutions are crucial for the widespread adoption of machine learning techniques in various domains, particularly named entity recognition.

Feature selection is the first step in any classification problem, especially those that are ambiguous by nature such as named entity recognition problems. The challenge is compounded by the aim to develop a solution that does not rely on prior language or domain knowledge. The plan proposes a feature discovery phase where an increasingly
large set of binary features from different textual definitions are extracted. The input space therefore consists of high-dimensional sparse vectors.

Support vector machines have been used successfully for named entity recognition. Newly emerging unsupervised or semi-supervised techniques such as support vector clustering (Ben-Hur et al. 2001) promise new grounds in this area. However, slow training and high memory requirements hinder the widespread adoption of support vector machines in applications with large data. Tuning training parameters and selecting and/or designing a kernel that is suitable for the problem at hand are major SVM usability issues.

In addition, multi-class identification and classification requires training several support vector machines on one class or a combination of classes when using One-Against-All, pairwise (one-against-one), or half-against-half multi-class solutions. The combined solution is an order of magnitude larger than the single class problem depending on the number of separate SVMs built. Training one SVM to classify all classes at the same time is a much bigger optimization problem and is impractical to use on large datasets until more efficient optimization algorithms are developed.

Finally, due to the lack of integrated systems that offer a complete solution to the NER/SVM problem, advancing the state of the art and/or constructing the solution requires the integration of several, often incompatible, components and the development of integration and interface tools. Reusability and maintainability of existing tools is difficult and the introduction of new algorithms and techniques requires the reconstruction of several components.
4.7 SVM Usability Issues

As discussed earlier, we view the usability issues of support vector machines as a two-fold problem: one of machine learning and one of software engineering. The machine learning usability issues are similar to those experienced with most machine learning approaches. They include the difficulty in selecting the best learning parameters, data preparation, and the need to restart the learning process when new training data is available. In addition, selecting the best kernel function for the data at hand is another factor that adds to the complexity of support vector machines.

As for the software engineering aspect, it stems from the lack of integrated and/or compatible SVM tools. Building a new learning environment requires the compilation of often incompatible tools to carry out any preprocessing needed, feature extraction, training and classification, then any additional post-processing requirements for the solution. Due to the lack of integrated tools, one may often need to build special interfacing modules to fit all the pieces together and build a total solution. The following are some of the usability questions for both sides of the problem.

4.7.1 Machine learning issues

- Model selection (parameter tuning): How to select model parameters? The number of parameters to be configured for a given set of data depends on the optimization algorithm and the kernel function. Common ways to help in deciding the parameters to be used are grid search, cross validation, decision tress, and use of heuristics.
  - Grid search: try different parameter values and compare the performance results obtained by each combination using a grid.
Cross validation: chunk the data into different pieces, train each piece with different parameter values then cross-validate the performance results using the rest of the data.

Decision trees, heuristics: starting with one parameter value, modify other parameters and use a decision tree to figure out the best combination. May also apply heuristics to speed up the decision process.

Kernel selection: How to select a kernel function that is suitable to a given problem data? Since it is difficult to choose a suitable kernel function simply by examining the data, the selection is based mostly on trying different functions and validating the results obtained until a good function is found. Also, coming up with a new kernel function for a specific type of data is a non-trivial task. Kernel selection is done by:

- Heuristics based on input dimensions
- Cross validation

Input data formatting, scaling, normalization: Training and test data needs to be prepared for use by the machine learning of choice. Preparation may include special formatting, scaling the vector values to a suitable range, or normalizing the vector values across different data sets.

Adding new training data requires restarting the learning process, which is a time consuming process.

4.7.2 Software engineering issues

Lack of integrated machine learning tools: many pieces of software and tools tackling specific processes such as feature extraction, learning or classification
exist, but using the pieces to build a total solution requires repeating or reinventing one or more parts of the solution and/or developing interfacing tools to fit the pieces together.

- Lack of standardization, incompatible interfaces, need to “reinvent the wheel” to fit pieces together: this is related to the previous issue. Available tools are often developed in isolation and no standard interfaces exist to make combining several tools together more streamlined.

- How to implement new algorithms for partial problems? In order to develop new algorithms for focused areas of the overall solution, for e.g., optimization, model selection, etc., one needs to rebuild the overall machine learning solution before getting to the point where new contributions could be made.

- How to incorporate optional components into the overall NER/SVM solution? It is often useful to compare results from different algorithms for the same component, for e.g., optimization. To accomplish this objective using existing tools, one needs to build parallel solutions where only one or two components are different.

For future research, we propose a dynamic service-oriented machine learning architecture that promotes reusability, expandability, and maintainability of the various components needed to implement the machine learning solution. The aim of the dynamic architecture is to provide a research environment with a flexible infrastructure such that researchers may easily focus on specific components without spending much time on rebuilding the experimentation infrastructure. The proposed architecture’s design will be service-oriented with a clear definition of the inter-modules interaction and interfaces.
This future research suggestion aims to advance the state of the art by offering a novel machine learning framework that promotes reusability, expandability, and maintainability of the solution and provides an architecture that encourages future work. More details about the recommended architecture are discussed in Chapter 10. We consider the database solution presented in Chapter 6 to be a first step towards the implementation of the full SOA architecture.

In the next chapter, we tackle the main scalability challenge of the NER solution using SVM, which is slow training. We present the mathematical formulation of the basic multi-class classification problem, followed by other formulations that lower the number of variables considered for optimization. We then introduce and discuss a new cutting plane algorithm to accelerate training while providing good out-of-the-box problem.
Chapter 5
Improving Multi-Class SVM Training

To address the scalability issues of the NER/SVM solution using high-dimensional input space, we begin by tackling the training computational time component. In this chapter, we explore the mathematical foundation of multi-class support vector machines and the evolution of the problem formulation aiming to reduce the training time by improving the optimization algorithms. We then introduce a new cutting plane algorithm that accelerates the training process while achieving good out-of-the-box performance in linear time. The cutting plane algorithm is first implemented as a standalone C executable in order to study its effectiveness in reducing the computational time, and is later integrated into the database framework and embedded solution presented in Chapter 6.

5.1 Basic All-Together Multi-Class SVM

The basic All-Together multi-class SVM idea is similar to the One-Against-All approach. It constructs \( k \) two-class rules where the \( j \)th function \( \mathbf{w}_j^T \phi(\mathbf{x}) + b \) separates training vectors of the class \( j \) from the other vectors. There are \( k \) decision functions but all are obtained by solving one problem. Similar to the non-separable binary SVM case, the objective of the machine is to maximize the margin separating the different classes while minimizing the classification error of each data point, as represented by the slack variable. For a \( k \)-class problem with \( n \) training points, the multi-class support vector machine can be formulated as the minimization of:

\[
Q(\mathbf{w}, b, \xi) = \frac{1}{2} \sum_{j=1}^{k} \mathbf{w}_j^T \mathbf{w}_j + C \sum_{i=1}^{n} \sum_{j=1}^{k} \xi_{ij} \quad \ldots \quad \ldots \quad \ldots \quad (16)
\]
subject to: 
\[ \mathbf{w}_j^T \phi(\mathbf{x}_i) + b_j \geq \mathbf{w}_j^T \phi(\mathbf{x}_i) + b_j + 1 - \xi_{ij}, \quad \xi_{ij} \geq 0, i = 1, \ldots, n \text{ and } j \in \{1, \ldots, k\}, j \neq y_i \] 

where, \( x_i \) is the input vector for data point \( i \), \( y_i \) is the correct class for data point \( i \), \( \xi_{ij} \) is the slack variable associated with data point \( i \) relative to class \( j \) (i.e., the error measure for misclassifying \( i \) as belonging to class \( j \)), and \( C \) is the regularization parameter determining the trade-off between the maximization of the margin and the minimization of the classification error. Figure 5.1 is an illustration of the different slack variables relative to individual classes, which is the basic way of formulating the multi-class SVM problem.

In the basic multi-class SVM formulation, the machine needs to minimize \( k \times n \) slack variables, in addition to maximizing \( k \) margins. The multi-class classification decision

Figure 5.1 – Multi-Class SVM Error Representation
function is defined by \( \text{argmax}_{j=1,...,k} (w_j^T \phi(x) + b_j) \), i.e., a data point \( x \) is classified as the class \( j \) whose weights maximize the classification score for the point \( x \).

The constrained problem in (16) and (17) can be transformed into its unconstrained equivalent formulation by introducing the non-negative Lagrange multipliers \( \alpha_{ij} \) and \( \beta_{ij} \):

\[
Q(w, b, \xi, \alpha, \beta) = \frac{1}{2} \sum_{j=1}^{k} w_j^T w_j + C \sum_{i=1}^{n} \sum_{j=1}^{k} \xi_{ij} + \sum_{i=1}^{n} \sum_{j \neq y_i}^{k} \alpha_{ij} ((w_{y_i} - w_j)^T \phi(x_i) + b_{y_i} - b_j - 1 + \xi_{ij}) - \sum_{i=1}^{n} \sum_{j \neq y_i}^{k} \beta_{ij} \xi_{ij}
\]

\[
= \frac{1}{2} \sum_{j=1}^{k} w_j^T w_j - \sum_{i=1}^{n} \sum_{j \neq y_i}^{k} z_{y_i} (w_j^T \phi(x_i) + b_j - 1) - \sum_{i=1}^{n} \sum_{j \neq y_i}^{k} (\alpha_{ij} + \beta_{ij} - C) \xi_{ij}
\]

where, \( z_{y_i} = \begin{cases} \sum_{j \neq y_i}^{k} \alpha_{ij} & \text{for } j = y_i \\ - \alpha_{ij} & \text{otherwise} \end{cases} \) \( \ldots \ldots \ldots \ldots (19) \)

and the conditions for optimality are:

\[
\alpha_{ij} ((w_{y_i} - w_j)^T \phi(x_i) + b_{y_i} - b_j - 1 + \xi_{ij}) = 0
\]

for \( j \neq y_i, j = 1, \ldots, k, i = 1, \ldots, n \) \( \ldots \ldots \ldots \ldots (20) \)

\[
\beta_{y_i} \xi_{ij} = 0
\]

for \( j \neq y_i, j = 1, \ldots, k, i = 1, \ldots, n \) \( \ldots \ldots \ldots \ldots (21) \)

in addition to \( Q(w, b, \xi, \alpha, \beta) \) being minimized in \( w, b, \xi \) (derivatives equal to zero). The dual formulation is obtained by reducing (18) to (21) using the kernel function \( K(x, y) = \phi(x)^T \phi(y) \). The dual formulation is to maximize:

\[
Q(\alpha) = \sum_{i=1}^{n} \sum_{j \neq y_i}^{k} \alpha_{ij} - \frac{1}{2} \sum_{i=1}^{n} \sum_{j \neq y_i}^{k} z_{y_i} z_{y_j} K(x_i, x_j)
\]

subject to:
\[ \sum_{i=1}^{n} z_{ij} = 0 \quad \text{for} \quad j \neq y_i, j = 1, \ldots, k \quad \ldots \ldots \ldots \ldots (23) \]

\[ 0 \leq \alpha_{ij} \leq C \quad \text{for} \quad j \neq y_i, j = 1, \ldots, k, i = 1, \ldots, n \quad \ldots \ldots \ldots \ldots (24) \]

Finally, the decision function for class \( j \) is given by:

\[ f_j(x) = \sum_{i=1}^{n} z_{ij} K(x_i, x) + b_j \quad \ldots \ldots \ldots \ldots (25) \]

and the classification task for data point \( x \) is to find class \( j \) to satisfy \( \arg\max_{j=1,\ldots,k} f_j(x) \).

By examining the dual formulation of the basic multi-class SVM learning problem as defined in (22) to (24), we observe the following:

- Number of variables \( \alpha_{ij} \) in the optimization problem is equal to the number of training points \( n \) times the number of classes \( k \), i.e., \( n \times k \) variables.
- Number of constraints to be satisfied \( z_{ij} \) is equal to number of training points \( n \).
- The upper limit for the weight variables \( \alpha_{ij} \) is the regularization parameter \( C \).

With \( n \times k \) variables, the quadratic optimization problem would require \( O(n^3k^3) \) computational time to solve. This is the main issue with the multi-class SVM training. One can easily see that the training time would become prohibiting when a large number of training data points and classes is used for the learning task.

### 5.2 Improving SVM Training Time

From the discussion of the basic multi-class formulation, we see that any improvement in training time would require tackling one of the sources of delay by either: reducing the number of data points, reducing the data points dimensionality, and/or lowering the number of variables and constraints for the optimization problem.
Each of these possibilities has been the subject of research activities aiming to improve SVM training. In this work, we will focus on the later approach which is to accelerate training by lowering the number of variables and/or constraints for the optimization task.

(Crammer and Singer 2001) reduces the number of optimization variables by reducing the number of slack variables ξ_{ij} to ξ_i = \text{max}(ξ_{ij}) for j=1, ..., k. In other words, it reduces the size of the optimization problem by considering only the highest slack for each data point across all classes, thereby having one slack variable per data point. The primal and dual formulations are derived in the same way as for the basic SVM formulation. The mathematical proof is provided in (Crammer and Singer 2001) and extended in (Abe 2005) to include bias terms. We will simply provide the initial optimization problem and the final dual formulation in order to contrast it with the basic formulation. Using the n-slack formulation, the optimization problem is:

\[
Q(w, b, ξ) = \frac{1}{2} \sum_{j=1}^{k} w_j^T w_i + C \sum_{i=1}^{n} ξ_i
\]  

... ... ... ... (26)

s.t. \( (w_j^T - w_j^T)ϕ(x_i) + b_j - b_j \geq 1 - ξ_j \) for \( i = 1, ..., n, j = 1, ..., k, j \neq y_i \) ... ... (27)

The dual formulation is to maximize:

\[
Q(α) = \sum_{i=1}^{n} α_i \cdot \frac{1}{2} \sum_{j=1}^{k} \sum_{j=1}^{k} z_{ij} α_i α_j K(x_i, x_j)
\]  

... ... ... ... (28)

subject to:

\[
\sum_{i=1}^{n} z_{ij} α_i = 0 \quad \text{for } j \neq y_i, j = 1, ... , k
\]  

... ... ... ... (29)

\[
0 \leq (n-1) α_i ≤ C \quad \text{for } j \neq y_i, j = 1, ... , k, i = 1, ... , n
\]  

... ... ... ... (30)

and the decision function for class j is given by:
\[ f_j(x) = \sum_{i=1}^{n} z_i \alpha_j K(x_i, x) + b_j \quad \ldots \quad (31) \]

Comparing the basic formulation (22) to (24) to that of (Crammer and Singer 2001) in (26) to (28), we observe that the reduction of the number of slack variables from \( n \times k \) to \( n \) did not increase the number of constraints and the final optimization problem is much simpler than the basic SVM problem. However, using a large number of data points \( n \), the optimization time remains high as it requires \( O(n^3) \) to complete. The optimization algorithm in (Crammer and Singer 2001) is implemented in \textit{SVM-Multiclass} (Tsochantaridis et al. 2004).

In order to further improve the training time, one may attempt to reduce the number of data points considered by the optimization algorithm. This may be achieved by approximating the total accurate solution by one that attempts to reach a close accuracy by using a smaller number of data points. Cutting plane algorithms are developed to approximate convex optimization problems by finding data points that approach the optimal solution and discarding the other points.

### 5.2.1 SVM Structural 1-Slack Formulation

Following the baseline experiments presented in the previous chapter which aimed to identify the scalability issues with All-Together multi-class SVM, we performed a number of experiments using both single class and multi-class cases in order to identify ways to improve the multi-class training time. Using \textit{SVM-Light} (Joachims 1999, 2002) for the single class experiments and \textit{SVM-Multiclass} (Crammer and Singer 2001; Tsochantaridis et al. 2004) for multi-class problems, we observe that the number of support vectors generated in both cases is \( O(n^{0.8}) \). In fact, \textit{SVM-Multiclass} uses \textit{SVM-Light}'s quadratic optimizer, where the input to the optimizer is a realization of (Crammer
and Singer 2001). Details of the single-class and multi-class scalability experiments and results using different tools and datasets are presented in Chapters 7 and 8. The training time using *SVM-Light* and *SVM-Multiclass* is $O(n^2)$, where the multi-class case is $O(k^2)$ slower than the single case case, $k$ being the number of classes.

With the availability of the improved binary SVM implementation in *SVM-Perf* (Joachims 2006) which reduces the training time to linear time, and knowing that the multi-class solution can be built using the single class one, we analyzed the improved binary implementation in *SVM-Perf* in order to investigate ways to extend the solution to support multi-class training. *SVM-Perf* (Joachims 2006) is based on a newer SVM formulation than that of (Crammer and Singer 2001) which attempts to reduce the number of slack variables from $n$ variables to just one. The new formulation is referred to as the 1-slack formulation.

The 1-slack structural formulation is (excluding the bias terms):

$$Q(w, \xi) = \frac{1}{2} \sum_{j=1}^{k} w_j^T w_j + C \xi$$

$$\forall c \in \{0,1\}^n : \frac{1}{n} w^T \sum_{i=1}^{n} c_i y_i x_i \geq \frac{1}{n} \sum_{i=1}^{n} c_i - \xi$$

s.t.

where, $\xi = \sum_{i=1}^{n} \xi_i$  \hspace{1cm} (33)

In this formulation, one common slack variable $\varepsilon$ which constitutes an upper bound on all training errors. However, the number of constraints in this case is $2^n$, one for each possible vector $c = (c_1, c_2, ..., c_n) \in \{0,1\}^n$. Each constraint vector corresponds to the sum of a subset of constraints from the $n$-slack formulation in (26) and (27). The new constraint vectors constitute the input to the quadratic optimizer.
SVM-Perf (Joachims 2006) uses a binary cutting plane algorithm in order to reduce the number of constraints in the problem while providing an approximate solution in a constant number of iterations. Proof of equivalence of the 1-slack formulation to the \( n \)-slack formulation and convergence of the cutting plane algorithm are provided in (Joachims 2006). SVM-Perf binary training algorithm is the following:

\begin{enumerate}
\item Input: \( S = \{(x_1, y_1), \ldots, (x_n, y_n)\}, y \in \{-1, 1\}, C, \varepsilon \)
\item \( C = \emptyset \) (\( C \) is the set of constraints for input to the optimizer)
\item repeat
\item \( (w, \xi) \leftarrow \arg\min_{w, \xi} \frac{1}{2} w^T w + C\xi \)
\item s.t. \( \forall c \in C : \frac{1}{n} w^T \sum_{i=1}^{n} c_i y_i x_i \geq \frac{1}{n} \sum_{i=1}^{n} c_i - \xi \)
\item for \( i = 1, \ldots, n \) do
\item \( c_i = \begin{cases} 1 & y_i (w^T x_i) < 1 \\ 0 & \text{otherwise} \end{cases} \)
\item end of
\item \( C \leftarrow C \cup \{c\} \)
\item until \( \frac{1}{n} \sum_{i=1}^{n} c_i - \frac{1}{n} w^T \sum_{i=1}^{n} c_i y_i (w^T x_i) \leq \xi + \varepsilon \)
\item return\( (w, \xi) \)
\end{enumerate}

5.3 SVM-PerfMulti: New Multi-Class Instantiation

In the general case, the SVM structural 1-slack formulation (Joachims 2005) is the following (unbiased formulation):

\[ Q(w, \xi) = \frac{1}{2} \sum_{j=1}^{k} w_j^T w_j + C\xi \]  \hspace{1cm} \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots (34)

\[ \text{s.t. } \forall \bar{y} \in \bar{Y} \setminus \bar{y} : w^T [\Psi(\bar{x}, \bar{y}) - \Psi(\bar{x}, \bar{y}')] \geq \Delta(\bar{y}', \bar{y}) - \xi \]  \hspace{1cm} \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots (35)
where,\( \bar{y} \in \bar{Y} = (y_1, y_2, \ldots, y_k) \) is a set of possible \( k \) labels, and \( \Psi(\bar{x}, \bar{y}) \) is a function that describes the match between \( (x_1, \ldots, x_n) \) and \( (\bar{y}_1, \ldots, \bar{y}_n) \). The objective is to maximize \( w^T \Psi(\bar{x}, \bar{y}) \), where:

\[
\Psi(\bar{x}, \bar{y}) = \sum_{i=1}^{n} y'_i x_i 
\]

\[ \cdots \cdots \cdots \cdots \cdots (36) \]

Designing the structure of the function \( \Psi(\bar{x}, \bar{y}) \) and a suitable training loss function \( \Delta(\bar{y}', \bar{y}) \) for a given problem such that the \textit{argmax} is computed efficiently is the main objective for a particular instantiation of \textit{SVM-Struct} V3.0 and is left to the designer of the solution. The general algorithm to solve a quadratic optimization problem using the multivariate SVM 1-slack formulation (Joachims 2005) is the following:

\begin{algorithm}
\caption{Algorithm for solving multivariate quadratic optimization problems}
\begin{algorithmic}[1]
\State Input: \( \bar{x} = (x_1, \ldots, x_n), \bar{y} = (y_1, y_2, \ldots, y_k), C, \varepsilon \)
\State \( C = \emptyset \) (C is the set of constraints for input to the optimizer)
\Repeat
\State \( \bar{y}' \leftarrow \operatorname{argmax}_{\bar{y} \in \bar{Y}} \{ \Delta(\bar{y}', \bar{y}) + w^T \Psi(\bar{x}, \bar{y}') \} \)
\State \( \xi \leftarrow \max_{\bar{y} \in C} \left\{ \max \{0, \Delta(\bar{y}', \bar{y}) - w^T [\Psi(\bar{x}, \bar{y}) - \Psi(\bar{x}, \bar{y}')]\} \right\} \)
\State \textbf{if} \( \Delta(\bar{y}', \bar{y}) - w^T [\Psi(\bar{x}, \bar{y}) - \Psi(\bar{x}, \bar{y}')] > \xi + \varepsilon \) \textbf{then}
\State \( C \leftarrow C \cup \{ \bar{y}' \} \)
\State \( w \leftarrow \text{optimize SVM objective over } C \)
\State \textbf{end if}
\Until \( C \) has not changed during an iteration
\State return(\( w \))
\end{algorithmic}
\end{algorithm}

(Tsochantaridis et al. 2004) shows that the algorithm terminates after a polynomial number of iterations. The set of constraints \( C \) is iteratively filled with the most violated constraints found in the input training dataset. Algorithm 1 is an instantiation of Algorithm 2 for the binary classification case.
5.3.1 Accelerated Cutting Plane Algorithm

The 1-slack SVM structural formulation expedites the optimization process by reducing the number of variables to be optimized while shifting the decision on how to prepare the quadratic optimizer’s input data to the solution designer. Using an error rate loss function, $SVM-Perf$ builds one constraint vector for all data points and adds it to the input data vectors for the optimization process. The individual constraint value is zero if the point is correctly classified, or one otherwise. Inspired by the improved training time of $SVM-Perf$ as compared to that of $SVM-Light$, we develop a cutting plane algorithm for handling multiple classes at the same time using a loss function based on error rate.

In this section we describe a new multi-class instantiation of $SVM-Struct$ V3.0 (Tsochantaridis et al. 2005) – $SVM-PerfMulti$. Using the SVM 1-slack formulation, we introduce a cutting plane algorithm that identifies the most violated constraints to be used for the quadratic optimization. The cutting plane algorithm is inspired by the geometrical intuition behind support vector machines illustrated in Figure 3.2 and Figure 5.1. Considering the general non-linearly separable case, the objective of the optimization problem is to:

- Maximize the margin(s) separating the hyperplanes
- Minimize the slack error for each data point

In addition to the optimization objectives, the algorithm also attempts to boost the classification performance of the machine. The cutting plane algorithm iteratively increases the gap between the positive and negative examples for each class, which refines the input to the quadratic optimizer during consecutive learning iterations and accentuates the impact of the scarce positive examples.
Algorithm 3: Algorithm for training multi-class classification SVM
1: Input: $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}, y \in \{1, \ldots, k\}, C, \varepsilon, a$
2: \hspace{1em} $C = \emptyset$ (C is the set of constraints for input to the optimizer)
3: \hspace{1em} $l = \text{initial example loss value} = 100.0 / n$
4: repeat
5: \hspace{1em} $(w, \xi) \leftarrow \arg\min \frac{1}{2} \sum_{i=1}^{k} w_i^T x_i + C \xi$
6: \hspace{1em} \hspace{1em} s.t. $\forall c \in \{0,1\}^n : \frac{1}{n} w^T \sum_{i=1}^{n} c_i y_i x_i \geq \frac{1}{n} \sum_{i=1}^{n} c_i - \xi$
7: \hspace{1em} \hspace{1em} where $w = \{w_1, \ldots, w_k\}$
8: \hspace{1em} \hspace{1em} for $i=1, \ldots, n$ do
9: \hspace{1em} \hspace{1em} \hspace{1em} $c_i = \begin{cases} 1 & (w^T x_i) - \max_{j \in \{1, \ldots, k\}} \{w_j^T x_i\} \leq l \\ 0 & \text{otherwise} \end{cases}$
10: \hspace{1em} \hspace{1em} end for
11: \hspace{1em} \hspace{1em} $C \leftarrow C \cup \{c\}$
12: \hspace{1em} \hspace{1em} $l \leftarrow l + a \times \max_{i=1, \ldots, n} \{w^T y_i x_i\}$
13: until $\frac{1}{n} \sum_{i=1}^{n} c_i - \frac{1}{n} w^T \sum_{i=1}^{n} c_i y_i (w^T x_i) \leq \xi + \varepsilon$
14: return $(w, \xi)$

Lines 6 to 8 of Algorithm 3 aim to satisfy line 4 of Algorithm 2 by finding the set of most violated constraints that $\bar{y}'$ maximizes $\Psi(x, \bar{y}') = \sum_{i=1}^{n} y_i' x_i$.

The SVM-PerfMulti (Habib 2008) cutting plane algorithm identifies the most violated constraints by maximizing the difference between the classification score of a data point relative to its own class and the best score among all other classes. If the difference is greater than a loss measure threshold, the data point is considered to be correctly classified and therefore its associated constraint is not violated. Otherwise, the constraint is violated. In other words, the greater the difference in classification score between the correct classification and the next best, the more we consider than the trained model is capable of correctly classifying unknown data points. Since the cutting plane decision is
based on a slack error criteria, it falls in the category of a slack-rescaling algorithm. However, we use the margin-rescaling option during the optimization process, where individual weights are scaled by a fixed margin factor that is independent of the current loss value.

The initial loss measure threshold is equivalent to that of a total loss, i.e., 100% loss distributed among all data points. The loss threshold is increased after each iteration thereby separating the correct vs. incorrect classifications by a larger distance. We use a heuristic increment based on a fraction of the highest correctly classified score.

To find those \( \bar{y}' \) positions that maximize line 4 of Algorithm 2 –

\[
\bar{y}' \leftarrow \arg\max_{\bar{x}, \bar{y}'} \{ \Delta(\bar{y}', \bar{y}) + w^T \Psi(\bar{x}, \bar{y}') \} - \text{a sufficient condition is to maximize } \Psi(\bar{x}, \bar{y}')
\]

by classifying the input vectors using the trained model after each optimization cycle. If an input example is incorrectly classified, its associated constraint would be considered violated. In this case, finding the most violated constraints’ criteria would be:

\[
c_i = \begin{cases} 
1 & (w_{y_i}^T x_i) \leq \max_{j \neq y_i} \{w_j^T x_i\} \\
0 & \text{otherwise} 
\end{cases} \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad (37)
\]

We will now show that all constraints found by the criteria in (37) – all incorrectly classified points – are also found by line 7 of Algorithm 3 where a constraint is considered violated if:

\[
(w_{y_i}^T x_i) - \max_{j \neq y_i} \{w_j^T x_i\} \leq l
\]

i.e.,

\[
(w_{y_i}^T x_i) \leq \max_{j \neq y_i} \{w_j^T x_i\} + l \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad (38)
\]
If the correct classification score for data point \( i \) is less than the maximum score for all incorrect classes, 
\[
(w_j^T x_i) \leq \max_{j \neq y_i \leq 1, \ldots, k} (w_j^T x_i),
\]
then it will also be less than the maximum score plus a loss threshold \( l \), 
\[
(w_j^T x_i) \leq \max_{j \neq y_i \leq 1, \ldots, k} (w_j^T x_i) + l.
\]
This means that all violated constraints that need to be found in order to satisfy line 4 of Algorithm 2 are also found by line 7 of Algorithm 3, even if no loss threshold comparison is performed (\( l=0 \)).

However, adding the loss threshold comparison will cause a subset of the correctly classified data points to be flagged as incorrectly classified and added to the violated constraints. The loss threshold term \( l \) is therefore making the correct classification criteria more strict and requiring that the correct example is as far as possible for all other classes. Moreover, by incrementing \( l \) after each optimization cycle, the classification is further refined and the separation between classes is widened.

### 5.3.2 Boosting Classification Performance

We hypothesize that the effect of the stricter correct classification decision has the effect of boosting the classification performance, which is confirmed by the empirical results presented in Chapter 8. Widening the gap between the correct and incorrect classification for an example has the effect of boosting the weight of the positive examples for the correct class. As part of the investigative experiments presented in Chapter 4, one of the single class experiments using SVM-Light\(^2\) assessed the effect of boosting the weights of positive examples by some multiplier factor. The experiment results are presented in Table 4.3. Using a preset boosting factor led to improved

performance measures up to a certain level, up to a certain point after which performance decreased with higher boosting factors.

In Algorithm 3, we do not apply a preset boosting factor but rather use a fraction of the maximum correctly classified score to increase the loss threshold measure. This is a heuristic value indicating the highest sphere of correct scores. One may use other heuristic measures, such as the current value of the loss function $\Delta(\tilde{y}', \tilde{y})$. However, we observed that using the maximum correct score led to the best and more consistent results with different experimental datasets.

Algorithm 3 falls under the category of slack-rescaling algorithms because the violation criteria is based on the difference between the correctly classified score and the incorrect ones. We use slack rescaling for finding the most violated constraints yet use margin rescaling for the optimization phase. According to the learning constraint in (30), all learned weights are limited by the value of the regularization parameter $C$ which governs the trade-off between the slack minimization and the margin maximization. Using the performance boosting mechanism in Algorithm 3 leads to a faster stabilization of both optimization objectives thereby causing the trained model to reach a good out-of-the-box performance independent of the value of $C$ when binary features are used.

### 5.4 Reducing Memory Requirements

Using the SVM structural formulation for either binary or multi-class learning, the training time is improved by combining feature vectors into vector(s) of most violated constraints. The generated vectors require larger memory as the size of each constraint vector is $O(f)$ in the binary case and $O(kf)$ in the multi-class case, where $f$ is the number of features in the training set and $k$ is the number of classes. For e.g., for a training set
with 1,000,000 features and 10 classes – and assuming 8-bytes per feature for the feature number and its weight – a binary constraint vector may need up to 8MB of memory while a multi-class vector may need up to 80MB. These estimates constitute a worse-case scenario, where all features are represented in each vector for all classes. In practice, using the JNLPBA-04 training dataset with over a million features and 11 classes, the multi-class constraint vector size was about 0.5MB.

Although the support vector size in multi-class training could reach $O(k)$ multiples of the corresponding size in the binary case, experiments found that the multi-class overall memory requirements using SVM-PerfMulti do not approach this worst case possibility. Figure 6.3 compares the actual memory consumption for several binary and multi-class experiments. Since the memory needed depends on the number of constraints – or in other words, the number of learning iterations – reducing the number of iterations will lead to a lowered memory consumption. Algorithm 3 accelerates the learning process thereby reducing the total number of iterations (and support vectors).

In order to ensure that all memory allocations performed during the learning process are necessary, we performed extensive process time and memory profiling as part of the empirical analysis. We identified one area of improvement in SVM-Struct V3.0 where the trained model is copied at the end of the learning iterations. With the high amount of memory needed for a trained model using a large high-dimensional dataset, experiments using larger data sizes failed due to lack of memory and the overall program aborts without saving the trained model although the actual learning has been completed. Another area of improvement that we identified is to alter the way that SVM-Light shrinks the working set of constraint vectors by using a negative number of iterations-to-shrink.
This alteration has the effect of lowering the number of support vectors based on inconsistency independent of how long the support vector has been deactivated. The final effect is a reduced number of support vectors in the working set.

Having addressed ways to reduce the necessary online memory needed for the learning process, additional reduction may be achieved by using a different medium to store examples and/or support vectors. In the following chapter, we describe a database-supported architecture to alleviate the online memory needs as well as provide a user-friendly framework for SVM learning and classification.
Chapter 6
Database Framework for SVM Learning

In this chapter, we present an SVM solution assisted by a special database schema and embedded database modules. The solution incorporates the learning and optimization algorithms of SVM-Perf and SVM-PerfMulti. The database schema design allows storage of input data, evolving training model(s), pre-computed kernel outputs and dot products, and output data. The aim of this approach is to improve scalability by reducing the online memory requirements and to foster SVM usability by providing a framework for easy reusability and manageability of the learning environment and experimentation results.

Using a relational database to support SVM has been attempted in (Rüping 2002) and a more complete yet different solution is included in the Oracle 10g data mining product (ODM) (Milenova et al. 2005). MySvmDB (Rüping 2002) addresses the high memory requirements by using a relational database to store the input data and parameters. It does not handle the computational time limitations. In fact, communicating constantly with the database system is known to negatively impact the performance due to the cost of fetching stored data.

The only SVM database implementation that tackles usability and scalability issues is Oracle 10g commercialized SVM integration into the Oracle Data Mining (ODM) product (Milenova et al. 2005). Oracle’s approach to reducing the number of data points considered for training uses “adaptive learning” where a small model is built then used to classify all input data. New informative data points are selected from all remaining input data and the process is repeated until convergence or until reaching the maximum
allowed number of support vectors. Our approach does not reduce the input data size in order to evaluate the efficacy of the database-embedded modules in providing a scalable solution.

Oracle’s multi-class implementation uses a One-Against-All classification method where several binary machines are built and scoring is performed by voting for the best classification. The number of binary machines in this case is equal to the number of classes in the training data. Our SVM-MultiDB approach uses All-Together training and classification where only one machine is built and used for classification. The new embedded database modules supporting both the single class case as well as the All-Together multi-class case can be used to implement the other multi-class learning approaches combining binary machines, if needed.

Building a growing list of previously identified and annotated named entities will be made possible by the database repository, which would provide a valuable resource to constantly improve the classification performance. The evolving gazetteer list can be used during preprocessing or post-processing to annotate and/or correct the classification of newly discovered named entities thereby boosting the overall performance.

6.1 SVM Database Architecture

We considered the potential choices for which database management system (DBMS) to use to carry out the research plan. We decided to select an open-source DBMS that would make easier the potential extension into a service-oriented architecture to be used by other users and researchers. The two main open-source database management systems
are MySQL\(^3\) and PostgreSQL\(^4\). Our selection criteria included the adherence to standards, the support for internal and embedded functions in different programming languages (preferably C), as well as performance and scalability. We decided to use PostgreSQL due to its rich features, adherence to standards, and the flexible options to extend DBMS via internal or embedded functions. While PostgreSQL is known to scale up and benchmarks have shown its scalability potential up to several terabytes of data, performance was an issue in the past. MySQL lacks some of the rich features of PostgreSQL and does not provide the same flexibility in extending the database server with embedded modules, yet it used to have a superior performance. Newer versions of both products are moving towards compensating for the potential weaknesses. MySQL 5.0 implements the previously missing features such as views and subqueries, while PostgreSQL 8.x versions focused on enhancing performance and scalability. As a result of the improved performance potential of the newer version of PostgreSQL and its established rich features and better adherence to standards, we decided to select PostgreSQL as the database management server supporting the proposed architecture.

In order to reduce the communication overhead with the database backend, we extend the database server with embedded C-Language functions. This also provides a better integration of all components. Database triggers are used for frequently updated values to ensure data integrity and improve the potential parallelization of the learning and database processes. Figure 6.1 presents the architecture used in the current implementation. Pre-processing (feature extraction) and post-processing (evaluation)

---


modules are kept outside of the database modules for simplicity. Additional supporting modules exist to import/export training and test examples, import/export training model(s), and trigger functions to compute derived data fields. To improve the usability of the SVM solution, a web-based user-friendly interface may be provided to allow the user to define new learning problems and parameters, import/export training and testing data and/or model(s), and monitor the execution of the learning process.

For the sake of simplicity, we will refer to the binary classification component of the database implementation as **SVM-PerfDB** and will refer to the multi-class component as **SVM-MultiDB**. The embedded database modules are written in C-language using
PostgreSQL’s Server-Side Programming Interface (SPI) to access and manipulate the data. The main objectives of using a database-supported solution are:

- Use improved learning algorithms in order to reduce the training time. This is achieved by using \textit{SVM-Perf} and \textit{SVM-PerfMulti} as a basis of the implementation.
- Reduce the online memory requirements by storing input examples and generated constraint vectors. A known concern of using a database in place of memory-based data structures is the potentially negative impact on computational time due to the need of frequent access to permanent storage. To remedy for this adverse reaction, one needs to minimize the need to refetch data, possibly by storing intermediate results of smaller size in memory.
- Improve usability of the SVM solution and provide a practical framework for SVM learning and classification.

### 6.2 \textit{SVM-MultiDB} Database Schema

The database schema design of \textit{SVM-MultiDB} aims to provide a practical framework for binary and multi-class SVM learning and classification. Figure 6.2 represents the main database schema. A detailed data dictionary is provided in Appendix A.

The objectives of the database schema are the following:

- Reduce online memory needs of a learning exercise by storing input examples and generated constraint vectors.
- Provide a way to store training and/or testing example datasets independent of a learning exercise.
- Be able to define multiple SVM experiments using the same training and/or testing datasets.
• Be able to use a subset of existing datasets for a learning experiment. This would be useful to conduct a grid search of the best learning parameters.

• Be able to reuse the same SVM exercise definition with different learning parameters.

• Be able to label the same example dataset differently in different learning exercises. For e.g., the same dataset may be used for binary classification of different named entities or for multi-class classification using a different number of classes as part of individual learning exercises without the need to reload the example dataset.

• Provide a way to store intermediate kernel evaluations and dot products of constraint feature vectors.

• Provide a way to store the generated most violated constraint vectors and learned model(s) for future classification use and potentially for new incremental learning algorithms.

• Be able to classify different test datasets at any time using existing learned model(s).

• Easily maintain the definition of learning experiments and parameters and examine their results.

• Provide a common environment for data exchange between learning and classification modules, which can serve as a basis for the recommended service-oriented-architecture presented in Chapter 10.

• Store textual classification labels in addition to their numeric representation to assist in building a growing dictionary of named entities.
Figure 6.2 - SVM Learning Data Model
As presented in Figure 6.2, the main table definitions supporting SVM learning and classification are the following:

- **Example_Set**: Defines a new set of training and/or testing examples.
- **Example**: Individual example input vectors that belong to a given example set. Note that the label stored is the original textual label, for e.g., B-protein, and not a given class number in order to facilitate the use of class identifications within different exercises.
- **Label_Set**: Defines a set of class labels.
- **Label**: Individual textual to class number mapping that belongs to a given label_set.
- **Kernel**: A lookup table of valid kernel types.
- **SVM**: Defines a selection of all or part of an example set and a given label set, to be used for a learning exercise.
- **Learn_Param**: Defines a set of learning parameters.
- **SVM_Learn**: Defines a specific learning exercise for an SVM definition and a set of learning parameters.
- **SVM_Model**: A set of generated constraints belonging to a given learning exercise. Support vectors that are selected for the final learned model are marked using the selected Boolean field. Computed alphas of the final learned model for each selected support vector are stored.
- **SVM_Model_Kernel**: This table may be used to store kernel evaluations (dot products for the linear case) in order to reduce computational redundancy and the need to refetch feature vectors for kernel computation.
- **SVM_Classify**: Defines a classification exercise.

- **Classified_Example**: Stores computed predictions of classified examples for a given classification exercise.

### 6.3 Embedded Database Functions

The following embedded server-side database functions are provided to perform the main learning and classification tasks as well as other supporting tasks.

- **import_examples**(example_set_id, filename): Used to load a set of examples for the given example set ID from a file into the database. Each example appears on a separate line and contains the original word, an array of features expressed as number:weight pairs, and a textual label classification of the word. In order to provide flexibility in the label/class translation and to allow reuse of the same example set in different learning/classification tasks with different class numbering – for instance, using the same example set in binary classification of varying named entities and/or for multi-class classification— we import the original label text for a given example, for e.g., B-protein.

- **export_examples**(example_set_id, filename): Export the examples belonging to the given example set ID from the database to a text file. Examples are written in a format suitable for re-import into the database.

- **export_examples**(example_set_id, label_set_id, filename): Export the examples belonging to the given example set ID from the database to a text file in SVM-Light format. Translate the textual labels to class numbers according to the numbering scheme defined by the given label set ID. Note that PostgreSQL supports polymorphism so the same function is usable with different arguments.
• **update_example_set**(example_set_id): Re-compute all computed fields in the `Example_Set` table and the `SVM` table for all pre-defined SVM entries that use the given example set ID. Fields such as `max_featnum`, `num_examples`, `num_labeled`, `num_unlabeled`, etc. are re-computed. Note that this function is mostly redundant as all computed fields are automatically updated on insert and/or update of the `Example_Set` table or the `SVM` table using functions and triggers.

• **svm_learn_trig()**: Trigger function invoked on insert and/or update of data in the `SVM` table. The function is associated with a `before` trigger named `tlearn_before` that updates all computed fields prior to carrying out the insert/update operation.

• **svm_learn**(svm_id, learn_param_id): Perform the learning operation defined by the given SVM ID using the learning parameters defined by the given learning parameters ID. Upon completion of the learning task, the trained model will be stored in the database and temporarily saved in an external file for optional use with external classification tasks. The beginning and ending timestamps for the learning operation are also logged in the database.

• **svm_classify**(svm_id, learn_paramid, example_set_id): Classify the given example set using the machine defined by SVM ID and previously trained using the given learning parameters ID.

• **export_model**(svm_id, learn_param_id): Export the trained model for SVM ID using the set of learning parameters ID to an external file.

These functions may be used either through an interactive `psql` sessions or via a user interface, preferably web-based. All direct database access operations are grouped in a single source file – `svm_db.c` – in order to facilitate porting of the database operations to
a different relational database management server or to a client-based environment. The user-accessible database functions and modifications to the C source code for SVM learning refer to generic database access routines defined in \textit{svm\_db.c}.

### 6.4 SVM-MultiDB Usage Example

A typical usage example of the database-integrated SVM learning module(s) is as follows (some definition steps need not be in a particular order):

1. Define new example set(s) using unique name ID(s).
2. Import examples for pre-defined example set ID(s).
3. Define a new set of class labels using a unique label set ID.
4. Define a new SVM learning environment using an existing example set ID and an existing label set ID. May optionally indicate a starting and ending example number if the learning task only uses a subset of the available examples in the example set.
5. Define a new set of learning parameters to be used by one or more SVM learning environments and assign a unique learning parameters ID to it.
6. Define a specific learning task using a pre-defined SVM learning environment and a pre-defined set of learning parameters.
7. Carry out a pre-defined learning task and store the trained model in the database.
8. Define a specific classification task using a previously trained model identified by an SVM learning environment ID and a set of learning parameters ID, and examples from a pre-stored example set.
9. Perform a pre-defined classification task and store the classified examples in the database. May optionally map the computed prediction values to their corresponding labels according to the label set defined in the SVM environment.

### 6.5 Improved SVM Solution Usability

The embedded SVM database solution promotes the usability of SVM learning and classification tasks by providing the following:

- Ability to define several SVM learning environments using the same example set.
- Ability to define sets of learning parameters for use with different learning tasks.
- Multiple label/class translation sets may be defined for later use with one or more example sets in different learning environments.
- Easily repeat the learning task with different parameters without losing the learning/classification results from a previous learning exercise. With the ability to define SVM environments using subset(s) from the same example set and/or different learning parameters, it is easy to develop additional supporting modules to perform cross-validation, grid search of parameters, kernel selection, etc..
- Since classified examples are stored in the database along with their prediction scores and/or classification labels, these results may be used to build a growing named entities dictionary, improve existing trained models, perform post-processing tasks to evaluate the model and/or boost the classification performance, to name a few possibilities.
- Building user-friendly interfaces to the database environment to carry out the definition of tasks and environments, perform the tasks, and monitor their results,
is as easy as developing a simple database application interface in a web-based environment or others.

6.6 Tradeoff of Training Time vs. Online Memory Needs

Using the SVM structural formulation for either binary or multi-class learning, the training time is improved by combining feature vectors into vector(s) of most violated constraints. The generated vectors require larger memory as the size of each constraint vector is $O(f)$ in the binary case and $O(fm)$ in the multi-class case, where $f$ is the number of features in the training set and $m$ is the number of classes. For e.g., for a training set with 1,000,000 features and 10 classes – and assuming 8-bytes per feature to store feature number and its weight – a binary constraint vector may need up to 8MB of memory while a multi-class vector may need up to 80MB. These estimates constitute a worse-case scenario, where all features are represented in each vector for all classes. The support vectors are represented by a sparse vector of number:weight pairs where only non-zero weight features are stored. Since the constraint vectors are composed of a weighted summation of several example feature vectors, they are much denser than individual example vectors, especially with larger training datasets.

In practice, using the JNLPBA-04 training dataset with over a million features and 11 classes, the multi-class constraint vector size was about 0.5MB. Figure 6.3 presents the online memory requirements with varying training data size for binary training (using regularization factor $C=0.01$, 0.14, and 1.0) as well as for the multi-class training. Note that although the multi-class constraint vector size is potentially 11 times larger in this experiment, the actual memory needs are less than this estimate, and not much larger than that needed for binary classification with a larger regularization factor $C$. 
By examining the time spent in different parts of the learning algorithm, it is noted that about 50% of the time is spent computing \( \text{argmax} \) to find the most violated constraints using the original input vectors, and the other 50% is spent optimizing the model using the constraint vectors. We will examine the impact of storing each vector type in the database on the overall training time.

### 6.6.1 Effect of Fetching Examples from Database

\( SVM-MultiDB \) provides a configurable example caching with three different options: no caching (i.e., examples are always fetched from the database), full caching (all examples are pre-fetched into memory), and a limited cache size where a predefined number of example records is fetched as needed. As expected, increasing the cache size minimizes the time impact up to a certain size after which we see some impact due to longer prefetch time. However, since the example vectors are requested only once and in a sequential manner to compute the most violated constraint, the overall impact of keeping example vectors in the database and fetching them as needed had a minimal impact on time in the binary case and almost no impact on the multi-class case. A
comparison of the training time using an example cache size of 500 is presented in Figure 7.1 for the binary case, and in Figure 8.3 and Figure 8.4 for the multi-class case.

The impact of the examples cache size on the overall computational time is presented in Figure 6.4 for the binary case, and in Figure 6.5 for the multi-class case. The binary experiments are conducted using a training size of 250,000 examples, $C=0.01$, and cache sizes varying from full cache, no cache, and sizes up to 10% of the training size. The multi-class experiments use a training data size of 100,000 examples, $C=0.01$, and same variation of cache sizes as the binary case.

![Figure 6.4 - SVM-PerfDB Training Time vs. Examples Cache Size](image)

Since example vectors are accessed during the process of finding the most violated constraints and computation of $\arg\max$ and not during the learning and optimization process, the effect of the example cache size is more noticeable with smaller values of $C$ in the binary case and decrease as the values of $C$ increases. Using a small example cache size offers the best trade-off between computational time and memory consumption as larger cache sizes would increase the time needed to replenish the case in each iteration.

---

5 The negative example cache size -1000 denotes full memory caching, i.e., all examples are pre-fetched and loaded into online memory. Full caching is denoted by a negative number just for visibility on graph.
In the multi-class case, we notice that the example cache size has less impact on the overall computational time, as presented in Figure 6.5, since more time is spent in the learning and optimization phase than in the computation of $\text{argmax}$ and the identification of the most violated constraints. Therefore, using a small cache size would reduce the online memory needs while providing the same effect on computational time.

### 6.6.2 Effect of Fetching Support Vectors from Database

Constraint and support vectors occupy more memory than example vectors and would result in a huge saving of online memory if maintained in the database. However, the existing C implementation of $\text{SVM-Perf}$ and $\text{SVM-PerfMulti}$ require frequent access to the feature vectors during the optimization process, mostly to compute kernel products and update linear weights. Moreover, using a variable support vector cache size may not be useful due to the frequent non-sequential access to the support vectors. An initial direct porting of the C implementations to database implementation without in-memory support cache negatively impacted the overall training time, which was expected. An

---

6 The negative example cache size -1000 denotes full memory caching.
efficient database implementation requires optimizing access to the constraint feature vectors by caching kernel evaluations in memory and minimizing the number of loops used to compute other intermediate results. An initial test of kernel product caching (with no constraint caching) resulted in about 50% time reduction.

Using larger cache sizes for constraint (support) vectors is not particularly beneficial since more time would be required to replenish the cache during each learning iteration. In our experiments, we observed that not caching the support vectors and fetching them one vector at a time had the same effect as keeping a small cache size. Since the constraint vectors are potentially very large in size, using no cache would provide the highest reduction in online memory needs while not impacting the computational time. The key to achieving the best trade-off is to minimize the number of loops that require access to the support vectors, ideally limiting such access to once per learning iteration. Kernel products caching in active memory and an incremental computation of trained weights make this objective possible, especially with a linear kernel.

In Chapter 7 and 8, we report the results of a series of scalability experiments using existing SVM implementations as well as our newly developed standalone and database-supported solution. Chapter 7 examines the results of single class experiments classifying protein named entities in the JNLPBA-04 biomedical data. In Chapter 8, we compare the outcomes of multi-class experiments using the same biomedical datasets where the following five named entity types are discovered: protein, DNA, RNA, cell line, and cell type. In addition, we apply the existing and new multi-class solutions to two additional datasets, namely the CoNLL-02 Spanish and Dutch challenge data aiming to classify general named entities such as a person, location, or organization.
Chapter 7

Single-Class Experiments

In this chapter, we examine the results of the first set of scalability experiments using single-class SVM. These experiments use the same training and test datasets described in Chapter 4 and Appendix B. The datasets represent a real-world problem, namely the biomedical named entity recognition, to identify the names of proteins, DNA, RNA, cell lines, and cell types in biomedical abstracts. The approach used promotes language and domain independence by eliminating the use of prior language-specific and domain-specific knowledge. Pre-processing of the training and test datasets is limited to extracting morphological and contextual features describing words in the biomedical abstracts and representing each vector with a high-dimensional binary vector, as described in Chapter 4. The input dimensionality of the complete training data exceeds a million features. The training data is composed of 492,551 examples and the test data includes 101,039 tokens.

The scalability experiments train single-class support vector machines to identify protein names using chunks of the training dataset with increasing size. The trained model is then used to classify protein named entities in the complete test dataset. The training time is noted in each experiment as well as the number of support vectors and the accuracy measures achieved. Although the focus of this work is to train an SVM to identify all named entities at the same time using an All-Together multi-class approach, we begin our experiments with a set of single-class experiments to accomplish the following objectives:
• Assess the ability of SVM to recognize patterns in high-dimensional input space in the simple binary case prior to an extended multi-class assessment.

• Establish a comparison baseline for the classification performance of one representative class, in this case the protein named entity class. The All-Together multi-class performance of individual classes is expected to be superior to that achieved using several binary machines, as depicted in Figure 3.4.

• Compare the $n$-slack and 1-slack SVM formulation improvement in training time to evaluate the potential gain that could be achieved by developing a multi-class instantiation using the 1-slack formulation. In the binary case, $SVM-Light^7$ represents the $n$-slack formulation while $SVM-Perf^8$ makes use of the 1-slack SVM formulation.

• Since our database framework incorporates both the single-class and multi-class SVM implementations, we need to evaluate the performance of the database-supported implementation and compare it to the non-database implementation. The objective is to achieve the same performance measures in both implementations, reduce the online memory requirements while minimizing the database access impact on the overall training time.

In the following sections, we report and examine the results of several sets of experiments conducted using increasing training data sizes, which include:

---


• Single-class experiments identifying protein names using Thorsten Joachims’ popular *SVM-Light* (Joachims 2002, 1999, 1998), and a regularization factor $C=0.01$ and 1.0.

• A set of experiments using 150,000 training examples and varying values of $C$ ranging from 0.01 to 1.0, at 0.01 increments, i.e., a total of 100 tests. These experiments aim to examine the impact of $C$ on training time and tune the parameter value for best performance which was found to occur at $C=0.14$.

• A set of performance validation experiments using varying training data sizes and different $C$ values, where the performance variation was found to be consistent.

• Single-class experiments identifying protein names only using the new SVM implementation, *SVM-Perf* (Joachims 2006, 2005; Tsochantaridis et al. 2004; Tsochantaridis et al. 2005), and a regularization factor $C=0.01$, 0.14, and 1.0.

• Single-class experiments identifying protein names only using our database embedded solution, *SVM-PerfDB*, and a regularization factor $C=0.01$, 0.14, 1.0.

• The training data chunks range from 1,000 examples to 492,551 examples (the complete training dataset). Each set of experiments consists of 51 tests.

All experiments use a linear kernel and a margin error of 0.1. The tests run mostly on an Intel Core 2 quad-processor 2.66 GHz machine and some run on a Xeon quad-processor 3.6 GHz machine. Running the same test on both machines completed in similar training time. We summarize the experiments results and analyze the findings in the following sections.
7.1 Single-Class Scalability Results

Using *SVM-Light* (Joachims 2002, 1999, 1998), a single-class support vector machine is trained to recognize protein name sequences. The trained machine is then used to classify proteins in the test data. Since no pre-processing was performed on the training and testing data besides features extraction, the positive examples in the data sets remained scarce. Training the *SVM-Light* machine with the complete training dataset and a regularization factor $C=0.01$ completed in about 28.5 minutes. The recall, precision, and F-score achieved in this case are 62.62, 56.07, and 59.17 respectively. Increasing $C$ to 1.0 raised the training time to about 269 minutes, and improved the accuracy measures to 68.76, 58.83, and 63.41 respectively.

The same set of experiments is repeated using *SVM-Perf* (Joachims 2006, 2005; Tsochantaridis et al. 2004; Tsochantaridis et al. 2005), which improves training time of linear machines to be linear w.r.t. the training data size. The training time improvement using *SVM-Perf* is several orders of magnitude as compared to that using *SVM-Light*, with the same classification results when trained with equivalent learning parameters. Since *SVM-Perf* optimizes vectors based on a linear combination of all training examples, the value of $C$ to be used in order to get equivalent results to those of *SVM-Light* is computed as follows:

$$ C_{\text{perf}} = C_{\text{light}} \times \frac{\text{training\_data\_size}}{100} \quad \ldots \cdot \ldots \cdot \ldots \cdot (39) $$

*SVM-PerfDB* embeds *SVM-Perf* into the database environment. To simplify its usage, the user selection of the $C$ factor is based on the *SVM-Light* value. This way, the user only selected one value independently of the training data size. *SVM-PerfDB* takes care of adjusting the regularization factor $C$ according to Equation (39).
Figure 7.1 compares the training time using both \textit{SVM-Light}, \textit{SVM-Perf}, and \textit{SVM-PerfDB} with the same data and equivalent learning parameters. The training time using \textit{SVM-Light} is polynomial $O(n^2)$ while being linear using \textit{SVM-Perf} and \textit{SVM-PerfDB}.

The number of support vectors using \textit{SVM-Light} is $O(n^{0.8})$ w.r.t. the training data size. However, using \textit{SVM-Perf}, the number of support vectors is only a very small fraction of the training data size and increased slightly with increased data size. The reduced number of support vectors is the main basis for the improved training time of \textit{SVM-Perf}. The relationship between the number of support vectors and the training time is observed to be the same as that of the training data size, i.e., training time is polynomial w.r.t. the number of support vectors with \textit{SVM-Light}, and linear with \textit{SVM-Perf}. Figure 7.2 and Figure 7.3 reflect the variation in the number of support vectors in the trained model with the number of examples in the training dataset. Using all 492,551 training examples, \textit{SVM-Light} produced 74,306 support vectors vs. 32 vectors with \textit{SVM-Perf}. 

![Figure 7.1 - SVM-Light, SVM-Perf, and SVM-PerfDB Training Time vs. Training Data Size](image-url)
7.2 Effect of Regularization Factor $C$

As a result of the observation of the enhanced accuracy measures with different training error vs. margin error trade-off factor values, and the associated impact on training time, we conducted a series of experiments investigating the effect of the trade-off factor $C$ on training time using $SVM-Perf$. Since the training time is consistently longer with $SVM-Light$ than it is with $SVM-Perf$, one can easily extrapolate the impact of
the trade-off factor $C$ on *SVM-Light* training time. The variation of training time w.r.t. the variation of the regularization factor $C$ is presented in Figure 7.4. The training time is found to scale in $O(\sqrt{C})$.

![Figure 7.4 - SVM-Perf Training Time vs. Variation of C-Factor](image)

As the variation of $C$ clearly impacts the training time, we investigated its impact on various accuracy measures in order to assess whether it is possible to perform a grid search for tuning the learning parameters using a subset of the training dataset. A set of experiments using different training data sizes and a range of $C$ values is conducted and the accuracy measures achieved are noted. Table 7.1 presents a sample of these results using different training data sizes and the complete set and some values of $C$. These experiments show that the accuracy improvement with varying $C$ is consistent with different training data sizes, where the accuracy measures reach their best values within a very close range of $C$ values (in this case $C$ values of 0.12 and 0.14 led to the best performance measures). Using the same learning parameter for multi-class training reflects similar consistency in accuracy enhancement.
### Table 7.1 - Consistent Performance Change with Varying C-Factor⁹

<table>
<thead>
<tr>
<th>Training Data Size</th>
<th>Number of Active Features</th>
<th>Regularization Factor C</th>
<th>Performance Recall/Precision/F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>67,023</td>
<td>C=0.01 27.71 / 35.76 / 31.22</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.12 48.17 / 44.61 / 46.32</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.14 48.83 / 45.02 / 46.85</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.20 48.75 / 44.60 / 46.58</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=1.00 48.40 / 44.34 / 46.28</td>
<td></td>
</tr>
<tr>
<td>25,000</td>
<td>132,126</td>
<td>C=0.01 32.72 / 38.55 / 35.40</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.12 53.74 / 48.87 / 51.19</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.14 53.51 / 48.66 / 50.97</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.20 53.58 / 48.28 / 50.79</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=1.00 52.29 / 47.54 / 49.80</td>
<td></td>
</tr>
<tr>
<td>50,000</td>
<td>219,678</td>
<td>C=0.01 36.97 / 43.38 / 39.92</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.12 57.48 / 52.91 / 55.10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.14 57.16 / 52.81 / 54.90</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.20 57.00 / 52.28 / 54.54</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=1.00 55.24 / 49.88 / 52.42</td>
<td></td>
</tr>
<tr>
<td>100,000</td>
<td>367,711</td>
<td>C=0.01 44.94 / 47.17 / 46.02</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.12 62.97 / 56.58 / 59.61</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.14 62.81 / 56.21 / 59.33</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.20 61.53 / 55.48 / 58.35</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=1.00 58.61 / 52.51 / 55.39</td>
<td></td>
</tr>
<tr>
<td>150,000</td>
<td>493,206</td>
<td>C=0.01 51.63 / 50.32 / 50.96</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.12 65.31 / 58.21 / 61.56</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.14 65.41 / 58.25 / 61.62</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.20 64.31 / 57.78 / 60.87</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=1.00 61.05 / 54.46 / 57.57</td>
<td></td>
</tr>
<tr>
<td>250,000</td>
<td>728,402</td>
<td>C=0.01 58.32 / 53.68 / 55.90</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.12 69.11 / 60.09 / 64.29</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.14 69.05 / 60.01 / 64.21</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.20 68.11 / 59.11 / 63.29</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=1.00 65.03 / 56.88 / 60.69</td>
<td></td>
</tr>
<tr>
<td>350,000</td>
<td>920,538</td>
<td>C=0.01 61.03 / 54.80 / 57.75</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.12 71.62 / 61.30 / 66.06</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.14 71.40 / 60.98 / 65.78</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.20 70.99 / 60.67 / 65.42</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=1.00 67.26 / 57.96 / 62.26</td>
<td></td>
</tr>
<tr>
<td>492,551</td>
<td>1,161,970</td>
<td>C=0.01 62.72 / 56.12 / 59.23</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.12 73.04 / 62.48 / 67.34</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.14 72.72 / 62.16 / 67.03</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=0.20 72.52 / 61.81 / 66.74</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C=1.00 68.06 / 59.29 / 63.37</td>
<td></td>
</tr>
</tbody>
</table>

⁹ These experiments were conducted on a 32-bit Linux operating system. A slight difference in floating-point computation precision is observed on 64-bit operating system.
This observation is very useful for tuning the learning parameters with a subset of the training dataset in order to achieve better accuracy with larger training data size during the final learning phase. Note that the F-score for protein single-class classification improved considerably using a $C$ value of 0.12, with an eight points improvement over that achieved with $C$ value of 0.01. Increasing the trade-off factor $C$ any further led to a decline in the F-score measure, as observed in Table 7.1. This shows that one can reach a reasonable trade-off between training time and the final accuracy measures by tuning the support vector machine with a fraction of the training dataset. Figure 7.4 illustrates the variation of the performance measures with varying $C$ using $SVM$-$Perf$ and a training data size of 150,000 examples.

![Figure 7.5 - SVM-Perf Performance vs. Regularization Factor C](image)

Figure 7.5 - $SVM$-$Perf$ Performance vs. Regularization Factor $C$

Figure 7.6 summarizes the protein classification performance variation with training data size and a regularization factor $C$ of 1.0 using $SVM$-$Perf$. The three different implementations lead to similar trends with minor floating-point computation precision differences.
As mentioned in Chapter 6, we alter the criteria for shrinking the training model in *SVM-Perf* when ported the database environment *SVM-PerfDB*. We use a negative value for the number of iterations used to decide whether to remove a support vector from the working set which changes the shrinking decision to be based on inconsistency only. This modification accelerates training, especially with higher values of $C$. It also reduces the number of support vectors thereby lowering the memory consumption. For example, the number of support vectors when training with the complete dataset and $C=1.0$ is 226 vectors without modification and 164 vectors with modification. The training time is reduced from 47.2 minutes to 19.1 minutes. The final recall/precision/F-score measures are 68.76 / 58.83 / 63.41 with the original *SVM-Perf* vs. 69.07 / 58.73 / 63.48 with the negative value of `iterations_to_shrink`.

Given the improved training time, number of support vectors, and memory usage, we applied the modification to *SVM-Perf* and conducted more experiments in order to
provide an apples-to-apples fair comparison of the database vs. the non-database implementations where the training environment is exactly the same with the only exception being the use of the database repository, leading to exactly the same performance measures and number of support vectors. Figure 7.7 reflects the memory needs with increasing training data size using $C=0.01$, 0.14, and 1.0 and a negative value of iterations_to_shrink. The online memory consumed is linearly proportional to the training data size, with higher slopes as $C$ increases.

Figure 7.7 – Modified SVM-Perf Memory vs. Varying $C$

Figure 7.10 and Figure 7.11 compare the training time of SVM-Perf and SVM-PerfDB in order to examine the impact of fetching the examples feature vectors from the database more closely. Figure 7.1 shows that both SVM-Perf (modified) and SVM-PerfDB improve the training time as compared to SVM-Light, with a slight increase in time when caching examples in the database. The example cache size used in this case is 500 examples. Figure 7.8 examines the training time difference with $C=0.01$ while Figure 7.9 performs the same with $C=1.0$. The training time remains linear using both implementations.
Comparing SVM-PerfDB to the original SVM-Perf, we notice that the training time using the database implementation with an example cache size of 500 is slightly higher than SVM-Perf with a small value of $C=0.01$. However, increasing the value of $C$ shows that using SVM-PerfDB trains faster than the original SVM-Perf despite the impact of fetching the examples from the database. This is due to the accelerated training and reduced number of support vectors with the altered shrinking decision criteria. Figure
7.10 and Figure 7.11 compare $SVM$-$PerfDB$ to the original $SVM$-$Perf$ with $C=0.01$ and $C=1.0$ respectively.

![Figure 7.10 – Original $SVM$-$Perf$ vs. $SVM$-$PerfDB$ Training Time: $C=0.01$](image1)

![Figure 7.11 – Original $SVM$-$Perf$ vs. $SVM$-$PerfDB$ Training Time: $C=1.0$](image2)

In the next chapter, we examine the scalability of multi-class SVM using the $n$-slack $SVM$-$Multiclass$ implementation and compare it to our new instantiation of the 1-slack SVM formulation in $SVM$-$PerfMulti$ as well as the database-supported solution $SVM$-$MultiDB$. 
Chapter 8
Multi-Class Experiments

In this chapter, we examine the results of multi-class scalability experiments using three datasets representing different languages and domains: the biomedical JNLPBA-04 challenge task, the CoNLL-02 Spanish task, and the CoNLL-02 Dutch task data. Details about these large-scale experimentation datasets is presented in Appendix B. The biomedical named entity recognition aims to identify the names of proteins, DNA, RNA, cell lines, and cell types in biomedical abstracts, while the objective of the CoNLL-02 multilingual named entity recognition task is to identify general names of persons, locations, organizations, and miscellaneous entities.

As in the single-class experiments, the approach used eliminates prior language-specific and domain-specific knowledge and limits pre-processing to the extraction of morphological and contextual features describing words in the text with high-dimensional binary vectors. The input dimensionality of the training and testing datasets is summarized in Table 8.1.

<table>
<thead>
<tr>
<th>Data Source</th>
<th>Dataset</th>
<th># of Examples</th>
<th># of Classes</th>
<th>Features Dimensionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biomedical JNLPBA-04</td>
<td>Training</td>
<td>492,551</td>
<td>11</td>
<td>1,161,970</td>
</tr>
<tr>
<td></td>
<td>Testing</td>
<td>101,039</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoNLL-02 Spanish</td>
<td>Training</td>
<td>264,715</td>
<td>9</td>
<td>1,775,408</td>
</tr>
<tr>
<td></td>
<td>Development</td>
<td>54,837</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Testing</td>
<td>53,049</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoNLL-02 Dutch</td>
<td>Training</td>
<td>202,931</td>
<td>9</td>
<td>1,900,465</td>
</tr>
<tr>
<td></td>
<td>Development</td>
<td>40,656</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Testing</td>
<td>74,189</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The class labeling scheme we use for the multi-class experiments is two class labels per named entity: one indicating the beginning of an entity (B-XXX) and one indicating a continuation of an entity (I-XXX), in addition to one class label representing other non-NER words (O). With this scheme, we have 11 classes in the biomedical datasets and 9 classes in the multilingual datasets.

The scalability experiments train the multi-class support vector machines using chunks of the training dataset with increasing size. The trained model is then used to classify named entities in the complete test dataset. In each experiment, we note the training time, the memory usage, the number of support vectors generated, and the performance measures achieved (recall / precision / F-score). For each of the three tasks, we conducted several sets of experiments, which include:

- Multi-class experiments identifying all named entities using Joachims’ multi-class implementation, *SVM-Multiclass* (Crammer and Singer 2001; Tsochantaridis et al. 2004) with a regularization factor $C=0.01$.
- Multi-class experiments identifying all named entities using our new multi-class instantiation, *SVM-PerfMulti* (Habib 2008) with $C=0.01$ and an acceleration factor=0.0001.
- Multi-class experiments identifying all named entities using our database embedded solution, *SVM-MultiDB* with $C=0.01$ and an acceleration factor=0.0001.
- The training data chunks range from 1,000 examples to the complete set of training examples in each case. Each set of biomedical experiments consists of 51 tests, the Spanish case includes 43 tests, and the Dutch case has 41 tests.
All experiments use a linear kernel and a margin error of 0.1. Most of the biomedical tests run on an Intel Core 2 quad-processor 2.66 GHz machine and some run on an Xeon quad-processor 3.6 GHz machine. Running the same test on both machines completed in similar training time. The CoNLL-02 Spanish and Dutch experiments are performed on the Intel Core 2 quad-processor 2.66 GHz machine.

In the next section, we examine the results of multi-class learning and classification of the biomedical named entities using the JNLPBA-04 datasets. Experiments using other datasets are presented in subsequent sections.

8.1 Multi-Class Scalability Results

The SVM-Multiclass implementation by T. Joachims is based on (Crammer and Singer 2001) and uses a different quadratic optimization algorithm described in (Tsochantaridis et al. 2004). Hsu and Lin (Hsu and Lin 2002) note that “as it is computationally more expensive to solve multi-class problems, comparisons of these methods using large-scale problems have not been seriously conducted. Especially for methods solving multi-class SVM in one step, a much larger optimization problem is required so up to now experiments are limited to small data sets.” The multi-class experiments presented herein attempt to solve a real-world large-scale problem using an All-Together classification method. The training data is composed of 11 classes where each named entity is represented by two classes – one denoting the beginning of an entity and the other denoting a continuation token within the same entity – in addition to one class denoting non-named entity tokens.

To explore the scalability issues of the All-Together multi-class SVM implementation, a series of experiments using different training data sizes is conducted.
with a low value for the C learning parameter equal to 0.01. The training time with 1,000 examples was 3.187 seconds and it increased considerably with increased data size to reach 416,264.251 seconds (6,937.738 minutes or 4.8 days) with the complete training data set of 492,551 examples, running on the same machine.

The \textit{SVM-Multiclass} (Crammer and Singer 2001; Tsochantaridis et al. 2004) implementation uses the learning algorithms implemented in \textit{SVM-Light}\(^\textsuperscript{10}\). The training time remains polynomial \(O(n^2)\) w.r.t. the training data size with a factor of \(O(k^2)\) increase in time as compared to the single-class \textit{SVM-Light} time, where \(k\) is the number of classes. The training time required for All-Together multi-class training is prohibiting to using this approach with large datasets.

By examining the number of support vectors selected using \textit{SVM-Multiclass}, we observe that the number of SVs is \(O(n^{0.8})\) w.r.t. the training data size, which is the same relationship as in the binary classification case using \textit{SVM-Light}. Since \textit{SVM-Multiclass} uses the learning algorithms implementation of \textit{SVM-Light}, and based on the effect that lowering the number of support vectors using linear combinations of input training examples has had on reducing the training time using \textit{SVM-Perf}, it is reasonable to anticipate similar gain if the new 1-slack SVM learning formulation is extended to the multi-class classification problem. This observation constitutes the main motivation behind developing our multi-class cutting plane algorithm. Figure 8.1 represents the number of support vectors variation with the training data size using \textit{SVM-Multiclass}. Figure 8.2 depicts the same relationship using \textit{SVM-PerfMulti}. The highest number of

support vectors when training with the complete training dataset is 159,165 vectors using
*SVM-Multiclass*, compared to 132 vectors with *SVM-PerfMulti*.

Figure 8.1 - *SVM-Multiclass* Number of Support Vectors vs. Training Data Size

Figure 8.2 - *SVM-PerfMulti* Number of Support Vectors vs. Training Data Size

*SVM-PerfMulti* and *SVM-MultiDB* reduce the training time by using an improved cutting plane algorithm in conjunction with the linear learning algorithm of *SVM-Perf* which is based on the 1-slack SVM formulation. Table 8.2 and Figure 8.3 compare the training time using the three methods. Using the 1-slack SVM formulation and the new
cutting plane algorithm, the training time is improved by several orders of magnitude. For instance, using the complete training dataset, the training time is 416,264.251 seconds (6,937.738 minutes or 4.8 days) using SVM-Multiclass compared to 5,890.031 seconds (98.17 minutes) using SVM-PerfMulti, or 1/70 of the SVM-Multiclass time. In addition, the performance measures (recall/precision/F-score) are improved from 62.4/64.5/63.5 using SVM-Multiclass to 67.9/66.4/67.2 with SVM-PerfMulti, using the default learning parameters in both cases ($\epsilon = 0.1$, $C = 0.01$, $a = 0.0001$).

Table 8.2 - Comparison of SVM-PerfMulti, SVM-MultiDB, and SVM-Multiclass Training Time vs. Training Data Size

<table>
<thead>
<tr>
<th>Training Data Size</th>
<th>SVM-Multiclass</th>
<th>SVM-PerfMulti</th>
<th>SVM-MultiDB (Examples Cache Size=500)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,000</td>
<td>94.213</td>
<td>15.370</td>
<td>15.553</td>
</tr>
<tr>
<td>10,000</td>
<td>355.101</td>
<td>38.110</td>
<td>34.798</td>
</tr>
<tr>
<td>25,000</td>
<td>2,214.092</td>
<td>104.721</td>
<td>108.257</td>
</tr>
<tr>
<td>50,000</td>
<td>7,784.148</td>
<td>254.326</td>
<td>268.901</td>
</tr>
<tr>
<td>75,000</td>
<td>16,662.753</td>
<td>494.168</td>
<td>492.843</td>
</tr>
<tr>
<td>100,000</td>
<td>23,531.920</td>
<td>788.916</td>
<td>716.954</td>
</tr>
<tr>
<td>125,000</td>
<td>45,260.549</td>
<td>1,059.173</td>
<td>1,105.600</td>
</tr>
<tr>
<td>150,000</td>
<td>65,524.410</td>
<td>1,294.051</td>
<td>1,156.002</td>
</tr>
<tr>
<td>175,000</td>
<td>72,108.693</td>
<td>1,618.301</td>
<td>1,533.246</td>
</tr>
<tr>
<td>200,000</td>
<td>91,632.975</td>
<td>1,866.653</td>
<td>1,830.997</td>
</tr>
<tr>
<td>225,000</td>
<td>88,282.629</td>
<td>2,153.460</td>
<td>1,932.653</td>
</tr>
<tr>
<td>250,000</td>
<td>100,189.058</td>
<td>2,483.609</td>
<td>2,204.677</td>
</tr>
<tr>
<td>275,000</td>
<td>158,229.577</td>
<td>2,535.886</td>
<td>2,369.009</td>
</tr>
<tr>
<td>300,000</td>
<td>165,178.301</td>
<td>2,528.715</td>
<td>2,594.110</td>
</tr>
<tr>
<td>325,000</td>
<td>206,240.711</td>
<td>2,851.640</td>
<td>3,006.574</td>
</tr>
<tr>
<td>350,000</td>
<td>248,070.056</td>
<td>3,549.062</td>
<td>3,733.130</td>
</tr>
<tr>
<td>375,000</td>
<td>215,760.927</td>
<td>3,589.297</td>
<td>3,766.130</td>
</tr>
<tr>
<td>400,000</td>
<td>310,408.398</td>
<td>4,488.086</td>
<td>4,523.788</td>
</tr>
<tr>
<td>425,000</td>
<td>352,636.685</td>
<td>4,066.876</td>
<td>4,327.744</td>
</tr>
<tr>
<td>450,000</td>
<td>358,716.303</td>
<td>4,615.680</td>
<td>5,011.895</td>
</tr>
<tr>
<td>492,551</td>
<td>416,264.251</td>
<td>5,890.031</td>
<td>6,292.167</td>
</tr>
</tbody>
</table>
Figure 8.3 - SVM-Multiclass, SVM-PerfMulti, and SVM-MultiDB Training Time vs. Training Data Size

SVM-Multiclass (Top), SVM-PerfMulti and SVM-MultiDB are very close (Bottom)

Figure 8.4 - SVM-MultiDB and SVM-PerfMulti Training Time vs. Training Data Size

SVM-MultiDB curve is the one ending up

Figure 8.4 takes a closer look at the impact of examples caching in SVM-MultiDB using a cache size of 500 examples. Note the minimal to no impact on training time in this case. The overhead of fetching example feature vectors from the database is minimized by limiting access to each example to one access per learning iteration. This overhead is much smaller than in the single-class case because each example fetch
corresponds to more processing in order to classify the example against all classes. The additional processing lowers the ratio of the time spent fetching the examples from the database to the time spent in learning and/or classification.

Lastly, we examine the memory consumed during the learning process, which is mostly composed of training example feature vectors and the generated constraint (support) feature vectors. As discussed in Chapter 6, the worst-case memory estimate is $O(nf) + O(kf)$, where $k$ is the number of classes, $n$ is the number of examples, and $f$ is the dimensionality of the feature vectors. The number of constraint vectors is proportional to the number of learning iterations, which is difficult to estimate a priori, but is known to be a small fraction of the number of training examples as discussed in (Joachims et al. 2008). For instance, using the JNLPBA-04 training dataset with 492,551 training examples with 1,116,970 features and 11 classes, and assuming each feature is represented by 8 bytes and about 1000 learning iterations, the worst-case memory needed would be 550GB for the example vectors and 12GB for the support vectors. In practice, the example vectors contain a very small fraction of all features per vector and not all features are present in each class’ portion in the constraint vectors.

In the biomedical JNLPBA-04 training case, the maximum number of features per example is 68, and each constraint vector size is about 0.5MB. As the number of learning iterations increases with the value of the regularization factor $C$ in the binary case, yet is mostly constant using our multi-class cutting plane algorithm, the memory needed for multi-class training is not much higher than binary training with higher $C$. Figure 8.5 compares the memory requirements for binary training (using regularization factor $C=0.01, 0.14,$ and 1.0) and for the multi-class training.
Figure 8.5 – Comparison of Single-Class and Multi-Class Memory Usage vs. Training Data Size

8.2 Multi-Class Performance Measures

Figure 8.6 presents the impact of the training data size on the multi-class classification performance measures in terms of precision, recall, and $F_{\beta=1}$-score. Figure 8.7 presents the protein classification performance variation with training data size using the multi-class approach. Note that the protein performance measures in this case are superior to the best achieved using binary classification. The final protein F-score in the binary case with $C=0.14$ is 67.27 as compared to an F-score of 69.17 in the multi-class classification case. The higher classification performance is expected of the All-Together multi-class learning approach, as all hyperplanes separating the classes are maximized at the same time and no unclassifiable regions result using this approach. Our multi-class empirical study confirms this expectation and makes the additional effort to develop an efficient All-Together multi-class SVM technique worthwhile. The performance results using $SVM-\text{PerfMulti}$ or $SVM-\text{MultiDB}$ are identical. The out-of-the-box performance achieved using default learning parameters are superior to those of $SVM-\text{Multiclass}$, where the final F-score if improved from 63.41% to 67.15%.
In Table 8.3, we compare the out-of-the-box performance results of *SVM-PerfMulti* to other published results using the same biomedical JNLPBA-04 datasets. The performance we achieve places this simplified approach in second place as compared to the others. It is important to note that the system in the first place (Zhou and Su 2004) achieved an F-score of 60.3% prior to boosting the performance using external resources such as dictionaries and gazetteer lists, which can still be applied to our classified output.
Table 8.3 – Performance of BioNLP Systems Using SVM vs. SVM-PerfMulti Results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhou (Zhou and Su 2004) (with post-processing)</td>
<td>75.3 / 69.5 / 72.3</td>
<td>77.1 / 69.2 / 72.9</td>
<td>75.6 / 71.3 / 73.8</td>
<td>75.8 / 69.5 / 72.5</td>
<td>76.0 / 69.4 / 72.6</td>
</tr>
<tr>
<td>Habib (with no post-processing)</td>
<td>59.5 / 70.3 / 64.5</td>
<td>69.9 / 66.3 / 68.1</td>
<td>69.4 / 67.2 / 68.2</td>
<td>68.1 / 65.1 / 66.6</td>
<td>67.9 / 66.4 / 67.2</td>
</tr>
<tr>
<td>Giuliano (Giuliano et al. 2005)</td>
<td>-- / -- / 60.3</td>
<td>-- / -- / 60.3</td>
<td>-- / -- / 60.3</td>
<td>-- / -- / 60.3</td>
<td>-- / -- / 60.3</td>
</tr>
<tr>
<td>Song (Song et al. 2004)</td>
<td>60.3 / 66.2 / 63.1</td>
<td>71.2 / 65.6 / 68.2</td>
<td>69.5 / 65.8 / 67.6</td>
<td>68.3 / 64.0 / 66.1</td>
<td>67.8 / 64.8 / 66.3</td>
</tr>
<tr>
<td>Rössler (Rössler 2004)</td>
<td>59.2 / 60.3 / 59.8</td>
<td>70.3 / 61.8 / 65.8</td>
<td>68.4 / 61.5 / 64.8</td>
<td>68.3 / 60.4 / 64.1</td>
<td>67.4 / 61.0 / 64.0</td>
</tr>
<tr>
<td>Habib (Habib and Kalita 2007)</td>
<td>53.2 / 70.8 / 60.7</td>
<td>63.7 / 63.6 / 63.7</td>
<td>64.2 / 65.4 / 64.8</td>
<td>63.0 / 63.2 / 63.1</td>
<td>62.3 / 64.5 / 63.4</td>
</tr>
<tr>
<td>Park (Park et al. 2004)</td>
<td>62.8 / 55.9 / 59.2</td>
<td>70.3 / 61.4 / 65.6</td>
<td>65.1 / 60.4 / 62.7</td>
<td>65.9 / 59.7 / 62.7</td>
<td>66.5 / 59.8 / 63.0</td>
</tr>
<tr>
<td>Lee (Lee, Hwang et al. 2004)</td>
<td>42.5 / 42.0 / 42.2</td>
<td>52.5 / 49.1 / 50.8</td>
<td>53.8 / 50.9 / 52.3</td>
<td>52.3 / 48.1 / 50.1</td>
<td>50.8 / 47.6 / 49.1</td>
</tr>
<tr>
<td>Baseline (Kim et al. 2004)</td>
<td>47.1 / 33.9 / 39.4</td>
<td>56.8 / 45.5 / 50.5</td>
<td>51.7 / 46.3 / 48.8</td>
<td>52.6 / 46.0 / 49.1</td>
<td>52.6 / 43.6 / 47.7</td>
</tr>
</tbody>
</table>

Table 8.4 – Summary of SVM-PerfMulti Experiment Results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>protein</td>
<td>65.02 / 70.84 / 67.81</td>
<td>77.61 / 66.19 / 71.44</td>
<td>75.28 / 64.08 / 69.23</td>
<td>75.55 / 62.44 / 68.37</td>
<td>75.00 / 64.19 / 69.17</td>
</tr>
<tr>
<td>DNA</td>
<td>66.96 / 65.79 / 66.37</td>
<td>59.74 / 66.67 / 63.01</td>
<td>59.61 / 74.92 / 66.40</td>
<td>53.74 / 71.98 / 61.54</td>
<td>57.89 / 70.64 / 63.63</td>
</tr>
<tr>
<td>RNA</td>
<td>100.00 / 50.00 / 66.67</td>
<td>69.39 / 66.67 / 68.00</td>
<td>53.85 / 66.67 / 59.57</td>
<td>55.71 / 58.21 / 56.93</td>
<td>59.30 / 62.96 / 61.08</td>
</tr>
<tr>
<td>cell type</td>
<td>59.95 / 74.84 / 66.57</td>
<td>59.48 / 72.22 / 65.23</td>
<td>59.52 / 81.89 / 69.84</td>
<td>56.94 / 80.90 / 66.84</td>
<td>58.49 / 78.58 / 67.06</td>
</tr>
<tr>
<td>cell line</td>
<td>34.66 / 59.22 / 43.73</td>
<td>57.14 / 53.93 / 55.49</td>
<td>61.81 / 53.61 / 57.42</td>
<td>58.24 / 43.81 / 50.00</td>
<td>52.43 / 51.19 / 51.80</td>
</tr>
<tr>
<td>Overall</td>
<td>59.53 / 70.33 / 64.48</td>
<td>69.93 / 66.30 / 68.07</td>
<td>69.35 / 67.16 / 68.24</td>
<td>68.11 / 65.13 / 66.58</td>
<td>67.93 / 66.39 / 67.15</td>
</tr>
</tbody>
</table>

Table 8.5 – SVM-PerfMulti Results 1978-1989 Set

<table>
<thead>
<tr>
<th>named entity</th>
<th>complete match</th>
<th>right boundary match</th>
<th>left boundary match</th>
</tr>
</thead>
<tbody>
<tr>
<td>protein</td>
<td>396 (609)</td>
<td>485 (79.64 / 86.76 / 83.05)</td>
<td>416 (68.31 / 74.42 / 71.23)</td>
</tr>
<tr>
<td>DNA</td>
<td>75 (112)</td>
<td>97 (86.61 / 85.09 / 85.84)</td>
<td>83 (74.11 / 72.81 / 73.45)</td>
</tr>
<tr>
<td>RNA</td>
<td>1 (100.00 / 50.00 / 66.67)</td>
<td>1 (100.00 / 50.00 / 66.67)</td>
<td>1 (100.00 / 50.00 / 66.67)</td>
</tr>
<tr>
<td>cell type</td>
<td>235 (59.95 / 74.84 / 66.57)</td>
<td>286 (72.96 / 91.08 / 81.02)</td>
<td>248 (63.27 / 78.98 / 70.25)</td>
</tr>
<tr>
<td>cell line</td>
<td>61 (34.66 / 59.22 / 43.73)</td>
<td>96 (54.55 / 93.20 / 68.82)</td>
<td>76 (43.18 / 73.79 / 54.48)</td>
</tr>
<tr>
<td>[-ALL-]</td>
<td>768 (59.53 / 70.33 / 64.48)</td>
<td>965 (74.81 / 88.37 / 81.02)</td>
<td>824 (63.88 / 75.46 / 69.19)</td>
</tr>
</tbody>
</table>

Table 8.6 – SVM-PerfMulti Results 1990-1999 Set

<table>
<thead>
<tr>
<th>named entity</th>
<th>complete match</th>
<th>right boundary match</th>
<th>left boundary match</th>
</tr>
</thead>
<tbody>
<tr>
<td>protein</td>
<td>1102 (1420)</td>
<td>1242 (87.46 / 74.59 / 80.52)</td>
<td>1215 (85.56 / 72.97 / 78.77)</td>
</tr>
<tr>
<td>DNA</td>
<td>230 (385)</td>
<td>310 (80.52 / 89.86 / 84.93)</td>
<td>256 (66.49 / 74.20 / 70.14)</td>
</tr>
<tr>
<td>RNA</td>
<td>34 (69.39 / 66.67 / 68.00)</td>
<td>45 (91.84 / 88.24 / 90.00)</td>
<td>37 (75.51 / 72.55 / 74.00)</td>
</tr>
<tr>
<td>cell type</td>
<td>273 (59.48 / 72.22 / 65.23)</td>
<td>352 (76.69 / 93.12 / 84.11)</td>
<td>299 (65.14 / 79.10 / 71.45)</td>
</tr>
<tr>
<td>cell line</td>
<td>96 (57.14 / 53.93 / 55.49)</td>
<td>119 (70.83 / 66.85 / 68.79)</td>
<td>110 (65.48 / 61.80 / 63.58)</td>
</tr>
<tr>
<td>[-ALL-]</td>
<td>1735 (2481)</td>
<td>2068 (83.35 / 79.02 / 81.13)</td>
<td>1917 (77.27 / 73.25 / 75.21)</td>
</tr>
</tbody>
</table>

11 The overall F-score achieved in (Zhou and Su 2004) is 72.6% after applying additional post-processing using dictionaries, named entity lists, etc. Without post-processing, the F-score achieved is 60.3%.
### Table 8.7 – SVM-PerfMulti Results 2000-2001 Set

<table>
<thead>
<tr>
<th>named entity</th>
<th>complete match</th>
<th>right boundary match</th>
<th>left boundary match</th>
</tr>
</thead>
<tbody>
<tr>
<td>protein</td>
<td>1641 (75.28 / 64.08 / 69.23)</td>
<td>1878 (86.15 / 73.33 / 79.22)</td>
<td>1830 (83.94 / 71.46 / 77.20)</td>
</tr>
<tr>
<td>DNA</td>
<td>245 (59.61 / 74.92 / 66.40)</td>
<td>295 (71.78 / 90.21 / 79.95)</td>
<td>262 (63.75 / 80.12 / 71.00)</td>
</tr>
<tr>
<td>RNA</td>
<td>28 (53.85 / 66.67 / 59.57)</td>
<td>41 (78.85 / 97.62 / 87.23)</td>
<td>28 (53.85 / 66.67 / 59.57)</td>
</tr>
<tr>
<td>cell type</td>
<td>425 (59.52 / 81.89 / 68.94)</td>
<td>540 (75.63 / 104.05 / 87.29)</td>
<td>449 (62.89 / 86.51 / 72.83)</td>
</tr>
<tr>
<td>cell line</td>
<td>89 (61.81 / 53.61 / 57.42)</td>
<td>114 (79.17 / 68.67 / 73.55)</td>
<td>97 (67.36 / 58.43 / 62.58)</td>
</tr>
<tr>
<td>[-ALL-]</td>
<td>2428 (69.35 / 67.16 / 68.24)</td>
<td>2868 (81.92 / 79.34 / 80.61)</td>
<td>2666 (76.15 / 73.75 / 74.93)</td>
</tr>
</tbody>
</table>

### Table 8.8 – SVM-PerfMulti Results 1998-2001 Set

<table>
<thead>
<tr>
<th>named entity</th>
<th>complete match</th>
<th>right boundary match</th>
<th>left boundary match</th>
</tr>
</thead>
<tbody>
<tr>
<td>protein</td>
<td>2407 (75.55 / 62.44 / 68.37)</td>
<td>2760 (86.63 / 71.60 / 78.40)</td>
<td>2682 (84.18 / 69.57 / 76.18)</td>
</tr>
<tr>
<td>DNA</td>
<td>316 (53.74 / 71.98 / 61.54)</td>
<td>404 (68.71 / 92.03 / 78.68)</td>
<td>342 (58.16 / 77.90 / 66.60)</td>
</tr>
<tr>
<td>RNA</td>
<td>39 (55.71 / 58.21 / 56.93)</td>
<td>54 (77.14 / 80.60 / 78.83)</td>
<td>40 (57.14 / 59.70 / 58.39)</td>
</tr>
<tr>
<td>cell type</td>
<td>648 (56.94 / 80.90 / 66.84)</td>
<td>858 (75.40 / 107.12 / 88.50)</td>
<td>691 (60.72 / 86.27 / 71.27)</td>
</tr>
<tr>
<td>cell line</td>
<td>99 (58.24 / 43.81 / 50.00)</td>
<td>132 (77.65 / 58.41 / 66.67)</td>
<td>109 (64.12 / 48.23 / 55.05)</td>
</tr>
<tr>
<td>[-ALL-]</td>
<td>3509 (68.11 / 65.13 / 66.58)</td>
<td>4208 (81.68 / 78.10 / 79.85)</td>
<td>3864 (75.00 / 71.71 / 73.32)</td>
</tr>
</tbody>
</table>

### Table 8.9 – SVM-PerfMulti Overall Results

<table>
<thead>
<tr>
<th>named entity</th>
<th>complete match</th>
<th>right boundary match</th>
<th>left boundary match</th>
</tr>
</thead>
<tbody>
<tr>
<td>protein</td>
<td>5546 (75.00 / 64.19 / 69.17)</td>
<td>6365 (86.07 / 73.67 / 79.39)</td>
<td>6143 (83.07 / 71.10 / 76.62)</td>
</tr>
<tr>
<td>DNA</td>
<td>866 (57.89 / 70.64 / 63.63)</td>
<td>1106 (73.93 / 90.21 / 81.26)</td>
<td>943 (63.03 / 76.92 / 69.29)</td>
</tr>
<tr>
<td>RNA</td>
<td>102 (59.30 / 62.96 / 61.08)</td>
<td>141 (81.98 / 87.04 / 84.43)</td>
<td>106 (61.63 / 65.43 / 63.47)</td>
</tr>
<tr>
<td>cell type</td>
<td>345 (52.43 / 51.19 / 51.80)</td>
<td>461 (70.06 / 68.40 / 69.22)</td>
<td>392 (59.57 / 58.16 / 58.86)</td>
</tr>
<tr>
<td>cell line</td>
<td>1581 (58.49 / 78.58 / 67.06)</td>
<td>2036 (75.32 / 101.19 / 86.36)</td>
<td>1687 (62.41 / 83.85 / 71.56)</td>
</tr>
<tr>
<td>[-ALL-]</td>
<td>8440 (67.93 / 66.39 / 67.15)</td>
<td>10109 (81.37 / 79.52 / 80.43)</td>
<td>9271 (74.62 / 72.93 / 73.77)</td>
</tr>
</tbody>
</table>

The detailed performance results for the different abstract collections in the JNLPBA-04 data and the classification performance of each of the five named entities: protein, DNA, RNA, cell_type, and cell_line are summarized in Table 8.4 through Table 8.9.

### 8.3 Effect of Acceleration Factor $a$

In this section, we examine the effect of the acceleration factor we introduced in the cutting plane algorithm of $SVM$-$PerfMulti$. As discussed in Chapter 5, this factor refines the criteria used when finding the most violated constraints by increasing the gap between the positive classification and the maximum negative classification score. The adjustment is a fraction of the maximum positive score achieved during the previous learning iteration, where the acceleration factor controls this fraction.
Figure 8.8 shows the effect of increasing the acceleration factor on the overall training time, using 50,000 training examples. As expected, training time is decreased as the acceleration factor increases. We also examine the effect this acceleration has on the performance measures achieved. Figure 8.9 and Figure 8.10 represent the performance variation with the acceleration factor for the overall classification of all five classes and that of the protein classification, respectively. We observe a very small variation in performance as long as the acceleration factor is synchronized with the final error criteria. In the experiment results shown below, the maximum acceleration factor $a$ is $1/100$ of the error rate $\varepsilon$. Note that the training time comparison presented earlier in Table 8.2 and Figure 8.3 corresponds to the slower training time in Figure 8.8 using $a = 0.0001$.

Similar to the effect of boosting the positive examples weights with $SVM-Light^{12}$, a decline in performance – in particular the precision and F-score – may occur with increased positive boosting as the trained model becomes overfitted to the positive examples. We observe the same with the acceleration factor as it has a positive boosting effect in the learning process. However, the performance achieved in much lower training time is still superior to that achieved using $SVM-Multiclass$. For instance, using 50,000 training examples, the performance measures (recall/precision/F-score) achieved with $a = 0.0001$ are 53.54/53.54/53.54, with $a = 0.0005$ are 50.26/52.77/51.48, and the lowest with $a = 0.001$ are 48.39/48.85/48.62. The performance measures achieved using $SVM-Multiclass$ and 50,000 training examples are 44.03/50.77/47.16. The training time in the three $SVM-PerfMulti$ cases is 256.119, 199.611, and 108.041 seconds, respectively, compared to 7,784.148 seconds using $SVM-Multiclass$.

---

12 Positive examples boosting is not supported by $SVM-Perf$ and the 1-slack SVM formulation.
Figure 8.8 – *SVM-PerfMulti* Training Time vs. Acceleration Factor

Figure 8.9 – *SVM-PerfMulti* Overall Performance vs. Acceleration Factor

Figure 8.10 – *SVM-PerfMulti* Protein Performance vs. Acceleration Factor
8.4 Experiments Using CoNLL-02 Data

In this section, we report the experimentation results using two named entity tasks for languages other than English, namely the CoNLL-02 Spanish and Dutch datasets. These are examples of general named entity recognition where the names of persons, locations, organizations, and miscellaneous entities are to be identified in news articles.

8.4.1 CoNLL-02 Spanish Dataset

Figure 8.11 compares the training time using \textit{SVM-Multiclass} and \textit{SVM-PerfMulti}. We observe the same time reduction as that observed with the biomedical JNLPBA-04 data. The overall classification performance with different training data sizes is presented in Figure 8.12. We compare the performance measures achieved using \textit{SVM-Multiclass} and \textit{SVM-PerfMulti} on the development and testing CoNLL-02 Spanish datasets in Table 8.10 and Table 8.11. In this NER task, \textit{SVM-PerfMulti} out-of-the-box performance is superior to that of \textit{SVM-Multiclass} using the default learning parameters, where the final testing accuracy, precision, recall, and F-score are 96.54, 66.65, 70.97, and 68.74 respectively with \textit{SVM-MultiClass}, compared to 96.98, 69.55, 74.32, and 71.86 using \textit{SVM-PerfMulti} – a ~3% boost in F-score. The training time is 39,082.52 seconds (10.86 hours) with \textit{SVM-Multiclass} and 1,804.05 seconds (30.06 minutes) using \textit{SVM-PerfMulti}.

<table>
<thead>
<tr>
<th>Named Entity</th>
<th>Development Dataset (Accuracy)</th>
<th>Test Dataset (Accuracy)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision/Recall/F-Score</td>
<td>Precision/Recall/F-Score</td>
</tr>
<tr>
<td>person</td>
<td>74.16 / 73.73 / 73.94</td>
<td>71.91 / 86.39 / 78.49</td>
</tr>
<tr>
<td>organization</td>
<td>65.20 / 65.24 / 65.22</td>
<td>65.73 / 73.43 / 69.37</td>
</tr>
<tr>
<td>location</td>
<td>59.22 / 78.58 / 67.54</td>
<td>70.89 / 70.76 / 70.82</td>
</tr>
<tr>
<td>miscellaneous</td>
<td>36.31 / 27.42 / 31.24</td>
<td>36.78 / 28.24 / 31.95</td>
</tr>
<tr>
<td>[-ALL-]</td>
<td>(95.27) 63.74 / 66.77 / 65.22</td>
<td>(96.54) 66.65 / 70.97 / 68.74</td>
</tr>
</tbody>
</table>
### Table 8.11 – CoNLL-02 SVM-PerfMulti Spanish Performance Results
Using Default Learning Parameters (C=0.01, a=0.0001, ε=0.1)

<table>
<thead>
<tr>
<th>Named Entity</th>
<th>Development Dataset (Accuracy) Precision/Recall/F-Score</th>
<th>Test Dataset (Accuracy) Precision/Recall/F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>78.59 / 80.20 / 79.38</td>
<td>74.74 / 89.39 / 81.41</td>
</tr>
<tr>
<td>organization</td>
<td>69.37 / 69.00 / 69.18</td>
<td>69.26 / 75.79 / 72.37</td>
</tr>
<tr>
<td>location</td>
<td>63.79 / 79.59 / 70.82</td>
<td>75.19 / 72.97 / 74.06</td>
</tr>
<tr>
<td>miscellaneous</td>
<td>42.09 / 42.47 / 42.28</td>
<td>40.00 / 40.00 / 40.00</td>
</tr>
<tr>
<td>[-ALL-]</td>
<td>(95.95) 67.72 / 71.83 / 69.71</td>
<td>(96.98) 69.55 / 74.32 / 71.86</td>
</tr>
</tbody>
</table>

![Figure 8.11](image1.png)  
**Figure 8.11 – CoNLL-02 Spanish SVM-PerfMulti vs. SVM-Multiclass Training Time**

![Figure 8.12](image2.png)  
**Figure 8.12 – CoNLL-02 Spanish SVM-PerfMulti Overall Performance**
8.4.2 CoNLL-02 Dutch Dataset

The same set of scalability and performance experiments in the previous section is repeated using the CoNLL-02 Dutch datasets. The training time comparison of SVM-Multiclass and SVM-PerfMulti is presented in Figure 8.13. The overall classification performance with different training data sizes is presented in Figure 8.14. Table 8.12 and Table 8.13 compare the performance measures achieved using both SVM implementations with default learning parameters. The final testing accuracy, precision, recall, and F-score are 97.24, 67.27, 68.79, and 68.02 respectively with SVM-MultiClass, compared to 97.57, 69.71, 72.55, and 71.10 using SVM-PerfMulti – a ~3% boost in F-score. The training time is 25,708.97 seconds (7.14 hours) with SVM-Multiclass and 850.62 seconds (14.17 minutes) using SVM-PerfMulti. Table 8.14 shows further improvement in performance using a slower acceleration factor of 0.00002, where the accuracy, precision, recall, F-score reach 97.62, 70.92, 73.15, and 72.02 respectively in 871.034 seconds (14.57 minutes).

Figure 8.13 – CoNLL-02 Dutch SVM-PerfMulti vs. SVM-Multiclass Training Time
Figure 8.14 – CoNLL-02 Dutch SVM-PerfMulti Overall Performance

Table 8.12 – CoNLL-02 SVM-Multiclass Dutch Performance Results
Using Default Learning Parameters (C=0.01, ε=0.1)

<table>
<thead>
<tr>
<th>Named Entity</th>
<th>Development Dataset (Accuracy) Precision/Recall/F-Score</th>
<th>Test Dataset (Accuracy) Precision/Recall/F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>55.87 / 80.51 / 65.97</td>
<td>60.25 / 84.34 / 70.28</td>
</tr>
<tr>
<td>organization</td>
<td>71.27 / 48.10 / 57.44</td>
<td>67.48 / 53.63 / 59.76</td>
</tr>
<tr>
<td>location</td>
<td>71.07 / 70.77 / 70.92</td>
<td>76.88 / 75.19 / 76.03</td>
</tr>
<tr>
<td>miscellaneous</td>
<td>73.59 / 68.18 / 70.78</td>
<td>70.53 / 61.50 / 65.71</td>
</tr>
<tr>
<td>[-ALL-]</td>
<td>(96.58) 65.95 / 66.70 / 66.32</td>
<td>(97.24) 67.27 / 68.79 / 68.02</td>
</tr>
</tbody>
</table>

Table 8.13 – CoNLL-02 SVM-PerfMulti Dutch Performance Results
Using Default Learning Parameters (C=0.01, a=0.0001, ε=0.1)

<table>
<thead>
<tr>
<th>Named Entity</th>
<th>Development Dataset (Accuracy) Precision/Recall/F-Score</th>
<th>Test Dataset (Accuracy) Precision/Recall/F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>59.85 / 80.80 / 68.77</td>
<td>64.70 / 85.79 / 73.77</td>
</tr>
<tr>
<td>organization</td>
<td>71.21 / 53.35 / 61.00</td>
<td>66.24 / 59.18 / 62.51</td>
</tr>
<tr>
<td>location</td>
<td>73.47 / 72.86 / 73.17</td>
<td>79.12 / 76.87 / 77.98</td>
</tr>
<tr>
<td>miscellaneous</td>
<td>72.49 / 71.52 / 72.01</td>
<td>72.40 / 67.40 / 69.81</td>
</tr>
<tr>
<td>[-ALL-]</td>
<td>(96.86) 67.94 / 69.50 / 68.71</td>
<td>(97.57) 69.71 / 72.55 / 71.10</td>
</tr>
</tbody>
</table>
Table 8.14 – CoNLL-02 SVM-PerfMulti Dutch Performance Results
Using Different Acceleration Rate (C=0.01, a=0.00002, ε=0.1)

<table>
<thead>
<tr>
<th>Named Entity</th>
<th>Development Dataset (Accuracy) Precision/Recall/F-Score</th>
<th>Test Dataset (Accuracy) Precision/Recall/F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>person</td>
<td>60.70 / 81.51 / 69.58</td>
<td>65.50 / 86.61 / 74.59</td>
</tr>
<tr>
<td>organization</td>
<td>72.37 / 54.23 / 62.00</td>
<td>68.16 / 60.20 / 63.94</td>
</tr>
<tr>
<td>location</td>
<td>74.84 / 73.90 / 74.37</td>
<td>79.25 / 76.49 / 77.84</td>
</tr>
<tr>
<td>miscellaneous</td>
<td>72.40 / 71.52 / 71.96</td>
<td>74.43 / 68.16 / 71.15</td>
</tr>
<tr>
<td>[-ALL-]</td>
<td>(96.94) 68.69 / 70.11 / 69.39</td>
<td>(97.62) 70.92 / 73.15 / 72.02</td>
</tr>
</tbody>
</table>

8.5 Other Experimental Datasets

Table 8.15 compares empirical results using some datasets often used for machine learning experimentation in the research community. The training and testing feature vectors are available for download at the LIBSVM (Chang and Lin 2001) website http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets. These sets represent various problems other than named entity recognition. While not related to the main focus of this work, we use these experimental datasets to validate the multi-class SVM solution.

We do not perform any pre- or post-processing on the data. The feature weights in all datasets – except the dna set – are real-valued. No specific knowledge about the semantics of the features selected or how their weights are generated. The classification precision results below are those reported by the SVM classification software which represent the average loss on the test set. Training using SVM-Multiclass and small datasets is fast, yet we do observe a difference in the test precision attained using SVM-PerfMulti. For instance, the vehicle test set has a precision of 66.8% using SVM-Multiclass and 78.0% using SVM-PerfMulti and the same value of the regularization factor C. Most of these datasets are small in size, so training time differences are
observed in the larger sets such as news20, protein, mnist, and shuttle. All SVM-PerfMulti tests used an acceleration factor of 0.0005.

<table>
<thead>
<tr>
<th>Dataset</th>
<th># of Examples</th>
<th># of Classes</th>
<th># of Features</th>
<th>Trade-Off Factor C</th>
<th>SVM-Multiclass</th>
<th>SVM-PerfMulti</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Training Time</td>
<td>Precision</td>
</tr>
<tr>
<td>vowel</td>
<td>528</td>
<td>11</td>
<td>10</td>
<td>256</td>
<td>112.851</td>
<td>41.56</td>
</tr>
<tr>
<td>usps</td>
<td>7,291</td>
<td>10</td>
<td>256</td>
<td>0.1</td>
<td>214.820</td>
<td>92.08</td>
</tr>
<tr>
<td>news20</td>
<td>15,935</td>
<td>20</td>
<td>62061</td>
<td>0.01</td>
<td>226.953</td>
<td>79.99</td>
</tr>
<tr>
<td>dna</td>
<td>1,400</td>
<td>3</td>
<td>180</td>
<td>0.01</td>
<td>0.895</td>
<td>93.68</td>
</tr>
<tr>
<td>protein</td>
<td>14,895</td>
<td>3</td>
<td>257</td>
<td>0.001</td>
<td>429.159</td>
<td>66.65</td>
</tr>
<tr>
<td>vehicle</td>
<td>846</td>
<td>4</td>
<td>18</td>
<td>0.01</td>
<td>0.449</td>
<td>66.80</td>
</tr>
<tr>
<td>mnist</td>
<td>42,000</td>
<td>10</td>
<td>780</td>
<td>0.01</td>
<td>718.977</td>
<td>92.48</td>
</tr>
<tr>
<td>satimage</td>
<td>3,104</td>
<td>6</td>
<td>36</td>
<td>0.01</td>
<td>1.886</td>
<td>79.35</td>
</tr>
<tr>
<td>segment</td>
<td>2,310</td>
<td>7</td>
<td>19</td>
<td>0.01</td>
<td>1.313</td>
<td>83.40</td>
</tr>
<tr>
<td>shuttle</td>
<td>30,450</td>
<td>7</td>
<td>9</td>
<td>0.01</td>
<td>33.147</td>
<td>91.92</td>
</tr>
<tr>
<td>letter</td>
<td>10,500</td>
<td>26</td>
<td>16</td>
<td>0.1</td>
<td>50.541</td>
<td>70.02</td>
</tr>
</tbody>
</table>

As we were about to conclude this research work, a new yet-to-be-published paper brought to our attention a new SVM-Multiclass implementation based on the 1-slack SVM formulation. We present the results of comparative experiments and an analysis of the differences between our cutting plane algorithm and that implemented in the newer SVM-Multiclass in the next chapter. Using our SVM-PerfMulti and the corresponding SVM-MultiDB database implementation, we are able to achieve better out-of-the-box performance in shorter training time.
Chapter 9
Subsequent Development: New SVM-Multiclass

As we conclude this research work, we ran a final literature review and came across a new yet-to-be-published paper (Joachims et al. 2008) that discusses the importance of cutting plane algorithms in lowering the training time using SVM-Struct. The paper presents some applications using the new SVM 1-slack formulation (Joachims 2006). A careful reading of the new paper reveals a brief mention of a multi-class instantiation, although not fully discussed. As the paper’s brief description of the multi-class case addresses the $n$-slack formulation, which is the version used earlier as part of this work, we checked a new copy of the source code and noticed differences suggesting that it is in fact using the new 1-slack formulation. In this chapter, we report on another set of scalability experiments that we ran using the new multi-class instantiation – SVM-Multiclass V2.12 – and examine the differences in the cutting plane algorithms and the experimentation results in terms of training time, memory, and classification performance.

9.1 Difference in Cutting Plane Algorithms

By examining the source code of the new SVM-Multiclass V2.12, and according to the brief description in (Joachims et al. 2008), this instantiation finds the most violated constraints by explicit enumeration of the classes. Using either margin-rescaling or slack-rescaling, the data points are simply classified using the current model and the point is assigned to the class with the maximum vote. The $\Psi(x_i, y_i)$ function computes a feature vector for one data point at a time and the cutting plane algorithm identifies each violated
constraint separately. The individual feature vectors are summed up as part of the
learning algorithm, not during the \textit{argmax} computation.

The cutting plane algorithm we introduced in Chapter 5 builds one combined vector
for all violated constraints during the \textit{argmax} computation and uses an additional
separation criteria to distinguish the positive and negative examples for each class. The
algorithm refines the trained model by introducing a positive boosting effect. The
experimental results highlight the effect of the different approaches used. Using
Algorithm 3, the resulting SVM achieves better performance out-of-the-box without the
need for parameter tuning, and takes less time to complete training. The memory
consumption using both cutting plane algorithms is comparable.

\section{Experimental Results Comparison}

In this section, we compare the training time, memory consumption, and
classification performance measures attained using \textit{SVM-Multiclass} V2.12 and \textit{SVM-PerfMulti}. Since both machines are based on the 1-slack SVM formulation, the main
difference is in the decision criteria of the cutting plane algorithm. The following
experiments compare the results using default learning parameters as well as those
achieved with parameter tuning for \textit{SVM-Multiclass} V2.12. For \textit{SVM-PerfMulti}, we use
default learning parameters only. The experiments demonstrate that \textit{SVM-PerfMulti} is
capable of achieving high performance measures without tuning. Reaching the same
performance measures using \textit{SVM-Multiclass} V2.12 requires learning parameter search
and when a suitable parameter value is identified, the training time needed is found to be
longer than that of \textit{SVM-PerfMulti}, not including the time required to perform the
parameter tuning phase. The results are consistent for the JNLPBA-04 biomedical data, as well as the CoNLL-02 Spanish and Dutch data.

Figure 9.1, Figure 9.3, and Figure 9.5 compare the training time needed to reach comparable performance results using *SVM-Multiclass* and *SVM-PerfMulti* for the biomedical JNLPBA-04 data, the CoNLL-02 Spanish data, and the CoNLL-02 Dutch data, respectively. Figure 9.2, Figure 9.4, and Figure 9.6 compare the memory usage of the two instantiations for the same datasets.

**Figure 9.1 – Biomedical JNLPBA-04 Training Time Comparison**

**Figure 9.2 – Biomedical JNLPBA-04 Memory Comparison**
Figure 9.3 – Spanish CoNLL-02 Training Time Comparison

Figure 9.4 – Spanish CoNLL-02 Memory Comparison

Figure 9.5 – Dutch CoNLL-02 Training Time Comparison
A comparison of the classification performance, learning parameters, and training time is reported in Table 9.1 for the biomedical JNLPBA-04 data, Table 9.2 for the CoNLL-02 Spanish data, and Table 9.3 for the CoNLL-02 Dutch data. Using the default learning parameters, the F-score achieved using \textit{SVM-Multiclass} and \textit{SVM-PerfMulti} is 63.35\% vs. 67.15\% for the biomedical JNLPBA-04 case, 68.77\% vs. 71.86\% for the CoNLL-02 Spanish case, and 67.78\% vs. 71.10\% for the CoNLL-02 Dutch case.

In order to achieve comparable performance results, the training time of \textit{SVM-Multiclass} compared to \textit{SVM-PerfMulti} is 12,231.320 sec. to 5,890.032 sec. for the JNLPBA-04 case, 3207.441 sec. to 1804.054 sec. for the CoNLL-02 Spanish case, and 1963.621 sec. to 941.293 sec. for the CoNLL-02 Dutch case. While the training time using both algorithms is several orders of magnitude faster than the original \textit{SVM-Multiclass} implementation using the $n$-slack SVM formulation, there is approximately a 2:1 difference in training time between \textit{SVM-Multiclass} V2.12 and \textit{SVM-PerfMulti}. 

![Figure 9.6 – Dutch CoNLL-02 Memory Comparison](image-url)
Table 9.1 – Biomedical JNLPBA-04 Performance Measures Comparison
Test Dataset Classification Results – Format: Recall/Precision/F-Score

<table>
<thead>
<tr>
<th>Named Entity</th>
<th>SVM-Multiclass V2.12</th>
<th>SVM-Multiclass V2.12</th>
<th>SVM-PermMulti</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Default $C=0.01, \varepsilon=0.1$</td>
<td>With tuning: $C=0.07, \varepsilon=0.1$</td>
<td>Default $C=0.01, \varepsilon=0.1, \alpha=0.0001$</td>
</tr>
<tr>
<td>protein</td>
<td>70.37 / 61.85 / 65.84</td>
<td>74.86 / 64.60 / 69.35</td>
<td>75.00 / 64.19 / 69.17</td>
</tr>
<tr>
<td>DNA</td>
<td>50.47 / 67.17 / 57.63</td>
<td>59.49 / 65.68 / 62.43</td>
<td>57.89 / 70.64 / 63.63</td>
</tr>
<tr>
<td>RNA</td>
<td>52.33 / 61.64 / 56.60</td>
<td>55.81 / 66.67 / 60.76</td>
<td>59.30 / 62.96 / 61.08</td>
</tr>
<tr>
<td>cell line</td>
<td>47.57 / 51.82 / 49.60</td>
<td>56.23 / 49.07 / 52.41</td>
<td>52.43 / 51.19 / 51.80</td>
</tr>
<tr>
<td>cell type</td>
<td>51.31 / 79.12 / 62.25</td>
<td>59.34 / 76.93 / 67.00</td>
<td>58.49 / 78.58 / 67.06</td>
</tr>
<tr>
<td>[-ALL-]</td>
<td>62.37 / 64.37 / 63.35</td>
<td>68.38 / 65.81 / 67.07</td>
<td>67.93 / 66.39 / 67.15</td>
</tr>
<tr>
<td>Training Time (sec)</td>
<td>6,642.820</td>
<td>12,231.320</td>
<td>5,890.032</td>
</tr>
</tbody>
</table>

Table 9.2 – Spanish CoNLL-02 Performance Measures Comparison
Test Dataset Classification Results – Format: (Accuracy) Precision/Recall/F-Score

<table>
<thead>
<tr>
<th>Named Entity</th>
<th>SVM-Multiclass V2.12</th>
<th>SVM-Multiclass V2.12</th>
<th>SVM-PermMulti</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Default $C=0.01, \varepsilon=0.1$</td>
<td>With tuning: $C=0.1, \varepsilon=0.1$</td>
<td>Default $C=0.01, \varepsilon=0.1, \alpha=0.0001$</td>
</tr>
<tr>
<td>person</td>
<td>72.43 / 86.12 / 78.68</td>
<td>76.30 / 88.03 / 81.74</td>
<td>74.74 / 89.39 / 81.41</td>
</tr>
<tr>
<td>organization</td>
<td>66.20 / 73.57 / 69.69</td>
<td>68.97 / 76.21 / 72.41</td>
<td>69.26 / 75.79 / 72.37</td>
</tr>
<tr>
<td>location</td>
<td>70.96 / 70.57 / 70.77</td>
<td>73.86 / 73.25 / 73.55</td>
<td>75.19 / 72.97 / 74.06</td>
</tr>
<tr>
<td>miscellaneous</td>
<td>35.09 / 27.35 / 30.74</td>
<td>41.59 / 41.47 / 41.53</td>
<td>40.00 / 40.00 / 40.00</td>
</tr>
<tr>
<td>[-ALL-]</td>
<td>(96.52) 66.82 / 70.83 / 68.77 (97.00) 69.55 / 74.43 / 71.91 (96.98) 69.55 / 74.32 / 71.86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training Time (sec)</td>
<td>1699.860</td>
<td>3207.441</td>
<td>1804.054</td>
</tr>
</tbody>
</table>

Table 9.3 – Dutch CoNLL-02 Performance Measures Comparison
Test Dataset Classification Results – Format: (Accuracy) Precision/Recall/F-Score

<table>
<thead>
<tr>
<th>Named Entity</th>
<th>SVM-Multiclass V2.12</th>
<th>SVM-Multiclass V2.12</th>
<th>SVM-PermMulti</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Default $C=0.01, \varepsilon=0.1$</td>
<td>With tuning: $C=0.1, \varepsilon=0.1$</td>
<td>Default $C=0.01, \varepsilon=0.1, \alpha=0.0001$</td>
</tr>
<tr>
<td>person</td>
<td>59.93 / 83.52 / 69.79</td>
<td>65.67 / 86.07 / 74.50</td>
<td>64.70 / 85.79 / 73.77</td>
</tr>
<tr>
<td>organization</td>
<td>68.11 / 53.51 / 59.94</td>
<td>67.57 / 59.30 / 63.16</td>
<td>66.24 / 59.18 / 62.51</td>
</tr>
<tr>
<td>location</td>
<td>75.19 / 75.97 / 75.58</td>
<td>78.78 / 76.74 / 77.75</td>
<td>79.12 / 76.87 / 77.98</td>
</tr>
<tr>
<td>miscellaneous</td>
<td>69.85 / 61.67 / 65.50</td>
<td>73.06 / 68.07 / 70.48</td>
<td>72.40 / 67.40 / 69.81</td>
</tr>
<tr>
<td>[-ALL-]</td>
<td>(97.22) 66.84 / 68.74 / 67.78 (97.59) 70.46 / 72.82 / 71.62 (97.57) 69.71 / 72.55 / 71.10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training Time (sec)</td>
<td>1314.670</td>
<td>1963.621</td>
<td>941.293</td>
</tr>
</tbody>
</table>

9.3 Effect of Regularization Parameter $C$

For these experiments, we used the slack-rescaling option of $SVM-Multiclass$. Although both margin-rescaling and slack-rescaling are said to be equivalent in (Joachims et al. 2008), using the same regularization factor $C$ and margin-rescaling failed to produce results (all performance measures were zero). For a successful operation, we had to use very large values of $C$ to get reasonable results with margin-rescaling.
As observed with the SVM-PerfMulti experiments, varying the value of $C$ had no effect on the learning process and final performance results. This is due to the stabilization of the trained weights during the learning process after a certain point in training with slack-rescaling. We investigated this observation with SVM-Multiclass and noticed that the training time varies linearly with the value of $C$ up to a certain point where no more changes are recorded in time or performance results. Figure 9.7 records the training time variation with the value of $C$ using a training set size of 50,000 JNLPA-04 examples.

Using SVM-PerfMulti, the cutting plane algorithm reaches the point of saturation at much lower values of $C$ due to the positive boosting effect introduced. The time and performance results using the default learning parameters of SVM-PerfMulti (same defaults as SVM-Multiclass) and higher were insensitive to the variation of $C$. The linear component of Figure 9.7 is much shorter with SVM-PerfMulti and occurs at very small values of $C$.

![Figure 9.7 – Effect of Regularization Parameter C on Training Time](image)
The effect of the variation of $C$ on the overall performance results of $SVM$-$Multi$ is presented in Figure 9.8. Using $SVM$-$PerformMulti$, we observed that the performance stabilization occurs at or very close to the peak performance.

![Figure 9.8 – Effect of Regularization Parameter $C$ on Performance](image)

The following chapter concludes this dissertation with a discussion of the known concerns and how we addressed them, a self-evaluation of the success criteria and contributions, as well as some brainstorming on some research directions that we would like to pursue in the future.
Chapter 10
Conclusion and Future Research

In this research work, we explored the scalability issues associated with solving the named entity recognition using support vector machines and high-dimensional input. We also assessed the usability of the machine learning environment and the available tools. Training an SVM to classify multiple independent classes at once is a complex optimization problem with many variables. We reviewed the basic All-Together multi-class problem formulation and two newer formulations that attempt to reduce the size of the optimization problem by reducing the number of variables to be optimized. The first method is presented in (Crammer and Singer 2001) where the number of slack variables is reduced from $n \times k$ to $n$ slack variables, where $n$ is the number of training data points and $k$ is the number of classes. (Tsochantaridis et al. 2005) simplifies the optimization task further by reducing the number of slack variables to just one, yet this improvement increases the number of constraints from polynomial to exponential $O(n^k)$ constraints. Using a cutting plane algorithm suitable for a given problem can limit the number of constraints within a linear subset.

To address the scalability issues of NER using SVM, we developed a new multi-class cutting plane algorithm – $SVM$-$PerfMulti$ – that accelerates the learning process while boosting the classification performance. The algorithm produces good out-of-the-box performance with linear kernels and limits the need to tune the learning parameters. On the other hand, given the potentially large memory needed to hold the generated constraints vectors – $O(kf)$ where $f$ is the number of features – we explored ways to
reduce the number of constraint vectors and an alternative solution where all vectors need not be resident in online memory. A new SVM database architecture and solution is presented that supports database caching of examples and support vectors.

The database architecture also provides a framework for SVM machine learning that promotes usability of the solution and reusability of prior learning exercises and classification results. The framework incorporates both binary and multi-class learning and allows flexibility in the definition of learning environments and class labeling options using the same set of training and testing examples. We demonstrated that caching examples’ feature vectors in the database does not pose a significant overhead on the overall training time, especially in the multi-class learning case where the overhead is almost non-existent. To eliminate any potential overhead due to support vectors storage in the database and not in online memory, we investigated main memory caching of kernel products and trained weights, which proved to be beneficial in cutting down the database access overhead while lowering the online memory needs.

### 10.1 Known Concerns and Solutions

In this section we summarize some of the concerns that we anticipated to face as part of the challenge of improving NER/SVM scalability and how they were addressed:

1. Multi-class SVM learning is a complex problem and finding ways to accelerate the training process is challenging: This was the most important and challenging step in the solution. It required careful analysis of existing support vector optimization and learning algorithms especially for the All-Together multi-class problem. Extensive process monitoring and profiling was required. Due to the lack of documentation for existing SVM implementations, we needed to undergo
a detailed analysis of existing code matching it to published algorithms in order to identify ways to develop the new solution.

2. A trade-off usually exists between training time and memory consumption: Using the new 1-slack SVM formulation in *SVM-Struct* V3.0 and our new multi-class instantiation *SVM-PerfMulti* with a large training dataset such as the JNLPBA-04 biomedical data, the size of the constraint vectors generated during the learning process caused several experiments to fail on memory allocation requests. We traced all memory allocations to identify potential areas of reduction, altered the learning environment to reduce the number of inactive constraints in memory, investigated the use of 32-bit and 64-bit operating systems, and developed a database solution that allows caching of examples and support vectors while limiting the need for frequent database access in order to lower overhead.

3. Designing a database schema to support SVM learning while minimizing the overhead of accessing data in permanent storage: One of the important design decisions related to the representation of feature vectors. If feature *number:weight* pairs are represented individually, access to any vector would require multiple fetches from the database and possibly several data type conversion and array building steps. While this representation would satisfy normalization of the data, it would incur a heavy overhead whenever a feature vector needs to be retrieved, which is frequently the case during the learning process. Moreover, storing individual information would require additional fields to identify each pair in a vector within a set of examples or support vectors. These identification fields would dramatically increase the data size needed to store each vector. After
careful analysis, we decided to store feature vectors – for either examples or support vectors – as binary arrays thereby representing the whole vector as a field in the corresponding table. This decision has the advantage of storing and retrieving these vectors in one database access without the need for data type conversion or array building. The resulting schema – although not fully normalized per the 3NF definition – is much simpler and faster to use.

4. The database schema and solution will incorporate both the binary and multi-class learning and classification and attempt to support the future direction for a service-oriented architecture that allows for the addition/replacement/deletion of modules: Using PostgreSQL server-side API, we built an extension shared library that eliminated redundancy in the learning and classification modules supporting the binary and multi-class case by making use of function pointers to identify unique modules while maintaining generic function names throughout the other modules. The basic data exchange needed for an SOA architecture is provided by the database schema where multiple learning environments and results may be maintained independently. For future work, we plan to extend the schema to support the definition of new modules and integrate them into the solution.

5. Implementation issues and/or limitations may arise with the use of low-level PostgreSQL SPI interface: The decision to develop embedded server-side modules improves efficiency while making the solution less portable. We attempted to isolate direct database access functions such that future porting of the solution to other RDBMS systems or to a client-based environment would require localized source code changes. We faced some limitations with the SPI supported
capabilities with earlier versions of PostgreSQL that impacted our implementation. For instance, scrolling cursors are not supported in versions earlier than 8.3. We currently use version 8.3.3 (the latest so far). This version has a limited support for updatable cursors, especially scrolling cursors, so we had to workaround this limitation. Also, PostgreSQL SPI does not currently support transactions or nested calls to SPI (as a potential workaround to transactions). Each extension function call is currently considered one transaction where failure of the function at any time would rollback any changes to the database. If we need to support the storage of partial learning results – for instance, permanently storing the trained model after each learning iteration – we recommend separating the quadratic optimizer module to run independently of the learning iterations.

6. The integration of the SVM learning and classification modules into the database solution may be a major challenge: The integration required low-level analysis of SVM-Light and SVM-Struct V3.0 source code in order to identify how and where to incorporate database access function calls in a way that minimizes the need for frequent or unnecessary access. With the heavy use of pointers to data structures (often with multiple levels of redirection), careful and detailed examination was needed. The resulting integration pinpoints and minimizes database access function calls. We also built the source code base as one integrated base for building the three solutions: SVM-Perf, SVM-PerfMulti, and SVM-MultiDB.

10.2 Evaluation of Success Criteria

The following is a self-evaluation of the success criteria of this research work:
1. Using the JNLPBA-04 datasets, the out-of-the-box performance results (recall, precision, and F-score) are comparable to published results which shows that the proposed language-independent, domain-independent approach using high-dimensional input space is valid. The performance results place the solution in second place after (Zhou and Su 2004) final results. However, the F-score achieved by (Zhou and Su 2004) prior to post-processing where many external knowledge sources are used to refine the classification was only 60.3. We are able to achieve an F-score of 67.15 using the machine learning step with no post-processing. If external knowledge such as specialized dictionaries or gazetteer lists of known named entities are available, they can be used to refine the classification results and boost the final performance measures. We consider our out-of-the-box results to be a baseline that can still be used in conjunction with other solutions if available and/or needed.

2. The scalability improvement is assessed by using incrementally increasing input data size and monitoring training time and online memory usage. Compared to SVM-Multiclass using the $n$-slack formulation, our solution is several orders of magnitude faster. Compared to the new SVM-Multiclass using the 1-slack formulation, our solution achieves better out-of-the-box performance faster and without the need for repeated experiments to tune the learning parameters. The database-supported solution provides ways to lower the online memory needs.

3. Scalability and performance results are reproducible with other datasets from other languages and/or domains, as demonstrated using the CoNLL-02 Spanish
and Dutch training and testing datasets. We compare the results using the same pre-processed data and do not consider post-processing possibilities.

4. To assess the improved usability of the database-supported solution, we defined several learning environments for binary and multi-class training using different data sizes, where all environments refer to the same example set loaded just once into the database. The solution made it possible to train many SVM models using the same example set and different class labeling schemes just by calling the `svm_learn()` function with varying SVM and learning parameter IDs.

### 10.3 Contributions

This research work contributes to the state of the art in biomedical named entity recognition using support vector machines by offering the following:

1. Improved SVM scalability: The support vector machine can train with large amounts of data in a reasonable total training time.

2. Improved multi-class training: New cutting plane algorithm for All-Together multi-class training that lowers the total training time and produces good out-of-the-box performance.

3. Improved SVM solution usability: Using the database-supported solution, users have the ability to define multiple SVM learning environments and class labeling schemes using the same example sets, store and reuse trained models, perform and store classification results using different trained machines, and take advantage of the database environment to construct other solutions such as the construction of dictionaries and gazetteer lists.
4. Using high-dimensional input space that eliminates prior language and/or domain knowledge is shown to be a feasible approach to named entity recognition, and is validated by the experiments results in the biomedical and multilingual examples.

5. An NER solution that is portable across languages and domains: Since the solution does not incorporate prior language and domain knowledge and does not require preprocessing beyond feature extraction, it can be used in different contexts with the same level of complexity. With fast learning and good out-of-the-box baseline performance, incorporating additional knowledge when available would be expected to further improve the classification performance.

10.4 Future Work

For future research, we would like to continue our work on natural language processing using machine learning in several directions including the extension of the database solution to support the recommended service-oriented architecture, incremental learning, structural multi-word named entity recognition, feature selection, and unsupervised learning. In this section, we present some ideas to investigate in each area.

10.4.1 Database Solution Extension

The SVM database framework presented in Chapter 6 offers a basis for a complete machine learning environment. To expand on the work presented in this document, we would like to refine the kernel matrix pre-computation when a new constraint vector is added to enhance the support vector caching options. Initial work on kernel matrix caching is very promising and shows at least 50% reduction in database access. Memory caching of incremental computation of the model’s weights can be introduced in order to
minimize the need to re-compute these weights during each learning iteration. Using a linear kernel, such incremental computation is possible.

Further enhancements to the database framework include the use of sub-transactions to store partially trained models and allow a failed or aborted learning exercise to resume from the last saved learning iteration results. If future versions do not support sub-transactions in server-side extension code, these may be simulated using further decomposition of the SVM learning and optimization functions to run within separate SPI environments. It is also possible to write a new client-based version of SVM-MultiDB using PostgreSQL’s client libraries. Although a client-based version would require more communication between the client and server processes, keeping the client software housed on the same server as the RDBMS could reduce the communication overhead. Another enhancement is to enable threading for the database SVM modules and make them reentrant thereby promoting the parallelization of the machine.

Finally we would like to expand the database framework to support our recommended service-oriented architecture for machine learning, presented in the following section.

10.4.2 Recommended Service-Oriented Architecture

We recommend a dynamic service-oriented machine learning architecture that promotes reusability, expandability, and maintainability of the various components needed to implement the machine learning solution. The aim of the dynamic architecture is to provide a research environment with a flexible infrastructure such that researchers may easily focus on specific components without spending much time on rebuilding the experimentation infrastructure. The proposed architecture’s design will be service-oriented with a clear definition of the inter-modules interaction and interfaces.
Figure 10.1 summarizes our view of an integrated machine learning environment. Within the recommended architecture, machine learning modules providing specific services are separated such that one may select the different components needed to build the total solution from existing alternative or complementary components. The architecture is assisted by a database schema specially designed to provide data exchange services between the different modules. With clear definition of the modules’ interfaces, one does not need to worry about compatibility issues. The dynamic architecture allows the co-existence of alternative implementations for the same service thereby providing a streamlined way to build different solutions for different problems or for comparison purposes.

Improving SVM usability is considered a by-product of the proposed architecture. Modules that address the model parameter selection or the kernel selection may be easily incorporated into the overall solution. In fact, several modules offering different solutions to the same problem may be included. The user can define the overall learning solution and/or improve on existing ones by selecting the different components that compose a learning task. These optional modules may include cross-validation for kernel selection, heuristic-based approaches, grid search modules for model selection, decision trees, or any other techniques. Model and kernel parameters are considered attributes of a given learning instance in the proposed database schema. Model and kernel selection modules are to be configurable options in the architecture. Oracle’s (Milenova et al. 2005) SVM solution provides built-in model and kernel selection methods and/or heuristics. Our recommended approach is to provide an architecture where several solutions may be simultaneously implemented and selected for a given learning task. In fact, optional
modules that further improve on SVM’s scalability such as instance pruning may also be incorporated by a user or a researcher.

![Figure 10.1 – Recommended Service-Oriented Architecture](image)

### 10.4.3 Incremental Learning

In addition to lowering the overhead due to access to permanent storage, the previous improvements would also pave the way for incremental learning, as we can reuse and refine the stored trained model and weight vectors in future training exercises. The incremental learning approach allows for easy incorporation of new training data when such data becomes available as it does not require a restart of the training process from scratch. The retraining process may be triggered either explicitly (through a user’s action) or implicitly (when a new input data chunk is available).
Finding the most violated constraints iteratively may be considered a form of incremental learning, where additional constraint vectors are introduced to the learner in each iteration. If new training data becomes available, one can incorporate this data into the database stored model and resume the learning process to refine the model and its weights using the new data. We anticipate that the incremental learning process may proceed as follows. Let the original training data set be \( S_{\text{init}} = \{(x_1, y_1), \ldots, (x_n, y_n)\} \) and that the newly available training data be \( S_{\text{new}} = \{(x_1, y_1), \ldots, (x_a, y_a)\} \), where \( y \in \{1, \ldots, k\} \):

**Algorithm 4:** Algorithm for multi-class incremental training

1. \( S_{\text{init}} = \{(x_1, y_1), \ldots, (x_n, y_n)\}, S_{\text{new}} = \{(x_1, y_1), \ldots, (x_a, y_a)\} \)
2. \( S_a = \emptyset, S_{\text{missed}} = S_{\text{new}} \)
3. for \( i = 1, \ldots, a \) do
   (classify new data points using trained model)
   4. if \( y_i = \text{argmax }_{j=1,\ldots,k} \{w_j^T x_i\} \) then
   5. \( S_a \leftarrow S_a \cup \{(x_i, y_i)\} \)
   6. \( S_{\text{missed}} \leftarrow S_{\text{missed}} - \{(x_i, y_i)\} \)
   7. end if
8. end for
9. \( C \leftarrow C + \Psi(\{(x_i, y_i)\} \in S_a) \) \( \) (incorporate new correctly classified points into existing constraint vectors)
10. if \( S_{\text{missed}} = \emptyset \) then
   (all new data points are correctly classified)
   11. return \( \) (no more training is necessary)
12. else
13. \( S = S_a \cup S_{\text{new}} \)
   Resume training using \( S \) as new input to Algorithm 3, starting
   with new constraint vector \( C \) and trained model \( M \)
14. end if
15. return new \((w, \xi)\)

Algorithm 4 starts by classifying the new data points using the existing trained model. All correctly classified points are grouped in the set \( S_a \). If all new points are correctly classified, incorporate their feature vectors into the existing constraint vectors set \( C \) by summing up the feature weights to their corresponding positions according to the
function \( \Psi(\overline{x}, \overline{y}) \) and exit as no more training is needed. Otherwise, incorporate the correctly classified feature vectors into \( C \) and resume training using all training data points starting with the modified constraint vectors set \( C \) and the trained model.

One possible modification on lines 9-15 would be to incorporate all new feature vectors using the function \( \Psi(\overline{x}, \overline{y}) \) into the constraint vectors \( C \) in the case where misclassified points exist, as this will be an additional measure of violated constraints. We would like to investigate both possibilities as part of future research work.

10.4.4 **Structural Multi-Word NER**

Another direction of future work is inspired by the ability to optimize structured problems using SVM-Struct. One of the main problems with biomedical named entity recognition is the limitation in fully classifying multi-word entities. By observing the accuracy of identifying individual words, left boundary words, right boundary words, and complete matches, it is clear that the machine learning solutions that focus on training with separate words fall short in achieving high accuracy with complete matches, while the accuracy of single word classification is much higher. A possible area of investigation is to construct the machine learning task such that all words belonging to one entity are optimized at the same time. Using the SVM-Struct API, since the optimization task is performed on a structured input that is defined by the loss function, the cutting plane algorithm, and the way in which the \( \Psi'(\overline{x}, \overline{y}') \) function is computed, building a structured named entity solution may be possible.

10.4.5 **Feature Selection**

One of the practical problems in machine learning solutions is feature selection and feature weight representation. While working with a high-dimensional input features is
becoming more feasible with the availability of faster training algorithms, designing a suitable feature selection and representation for a given problem is more art than science. For instance, choosing a reasonable window size to represent a sequence of words for named entity recognition is performed by trial and error. During the investigation phase of this work, we experimented with different window sizes ranging from [-1, 1] to [-5, 5] and found that a window of [-3, 3] was the best for the JNLPBA-04 datasets, decided by the performance measures achieved using the different feature vectors. However, using the CoNLL-02 Dutch datasets, a window of [-1, 1] achieved the best performance with an F-score of 73.24. We also investigated the elimination of unique features, merging feature vectors of similar word/class occurrences as well as scaling feature weights by feature rate per class. Exploring the automation of feature selection and representation to achieve the best performance measures is another area of interest.

10.4.6 Unsupervised Learning

Using support vector machines offers the possibility of a one-class classification, where the objective of the machine is to detect all points that likely belong to the same domain description or the same class where all other points are considered outliers. The idea is based on finding those points that lie within the same hypersphere in feature space and consider them all to be members of the same class.

The one-class classification ability of SVM has been extended to SVM clustering (Ben-Hur et al. 2001), where the machine objective is to identify hyperspheres in feature space to indicate the presence of a cluster in input space. The idea is to map the input space into a higher dimension feature space, for example, map a two-dimensional space to a three-dimensional feature space, where it is easier to identify enclosing spheres.
Reverting back to input space, those points that lie within the hypersphere are considered to form a cluster. While the SVM clustering is a promising means for unsupervised or semi-supervised learning, it had not been applied to large real-world data due to slow training. With the development of new learning and optimization algorithms that reduce the training time in the supervised learning case, we would like to explore the possibility of extending these algorithms to support one-class classification and clustering.
Appendix A

SVM Data Dictionary

The SVM data dictionary translates the schema presented in Figure 6.2. The schema supports the definition of training and testing datasets, multiple labeling designations, learning environments, trained models, and classification outputs. Binary and multi-class learning environments are integrated in the database schema and the embedded modules.

In the following data dictionary, primary key and foreign key constraints are indicated. Computed fields hold values that are computed during the learning and/or classification processes, or generated using insert/update triggers in order to ensure the data integrity. For instance, data fields in the table SVM that describe the example set in use are pre-computed using triggers invoked on insert/update of the table SVM or Example_Set. The pre-computation of these metadata fields eliminates the need of re-computing those limits prior to every learning exercise. It also provides a quick way of gathering statistics about the training/testing datasets.

In order to streamline the data representation of feature vectors, reduce database access during fetches, and eliminate the need for data type conversion and/or array building whenever feature vectors are needed during the learning and/or classification processes, we represent feature vectors as a binary array holding an array of number:weight pairs, as defined by the WORD data structure in svm_common.h (Joachims 1999, 2002):

```c
typedef struct word {
    int32_t wnum;    /* word number */
    float weight;    /* word weight */
} WORD;
```
<table>
<thead>
<tr>
<th>Table Name</th>
<th>Column Name</th>
<th>Data Type</th>
<th>Key</th>
<th>Constraints</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example_Set</td>
<td>example_set_id</td>
<td>CHAR(30)</td>
<td>PK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>example_set_desc</td>
<td>VARCHAR(255)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_featnum</td>
<td>INTEGER</td>
<td></td>
<td></td>
<td>Computed field</td>
</tr>
<tr>
<td></td>
<td>max_exwords</td>
<td>INTEGER</td>
<td></td>
<td></td>
<td>Computed field</td>
</tr>
<tr>
<td>Example</td>
<td>example_set_id</td>
<td>CHAR(30)</td>
<td>PK, FK</td>
<td>REFERENCES Example_Set</td>
<td></td>
</tr>
<tr>
<td></td>
<td>example_id</td>
<td>INTEGER</td>
<td>PK</td>
<td>NOT NULL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>example_text</td>
<td>VARCHAR(255)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>label</td>
<td>CHAR(30)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>features</td>
<td>BYTEA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Label_Set</td>
<td>label_set_id</td>
<td>CHAR(30)</td>
<td>PK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>label_set_desc</td>
<td>VARCHAR(255)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Label</td>
<td>label_set_id</td>
<td>CHAR(30)</td>
<td>PK, FK</td>
<td>REFERENCES Label_Set</td>
<td></td>
</tr>
<tr>
<td></td>
<td>label</td>
<td>CHAR(30)</td>
<td>PK</td>
<td>NOT NULL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>class</td>
<td>INTEGER</td>
<td>NOT NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kernel</td>
<td>kernel_type</td>
<td>CHAR(10)</td>
<td>PK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>kernel_desc</td>
<td>VARCHAR(255)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVM</td>
<td>svm_id</td>
<td>CHAR(30)</td>
<td>PK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>svm_type</td>
<td>CHAR(30)</td>
<td>NOT NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>example_set_id</td>
<td>CHAR(30)</td>
<td>FK</td>
<td>NOT NULL REFERENCES Example_Set</td>
<td></td>
</tr>
<tr>
<td></td>
<td>label_set_id</td>
<td>CHAR(30)</td>
<td>FK</td>
<td>NOT NULL REFERENCES Label_Set</td>
<td></td>
</tr>
<tr>
<td></td>
<td>kernel_type</td>
<td>CHAR(10)</td>
<td>FK</td>
<td>NOT NULL REFERENCES Kernel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ex_start</td>
<td>INTEGER</td>
<td>DEFAULT 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ex_end</td>
<td>INTEGER</td>
<td>DEFAULT -1</td>
<td></td>
<td>-1 denotes full set</td>
</tr>
<tr>
<td></td>
<td>num_examples</td>
<td>INTEGER</td>
<td></td>
<td></td>
<td>Computed field</td>
</tr>
<tr>
<td></td>
<td>max_featnum</td>
<td>INTEGER</td>
<td></td>
<td></td>
<td>Computed field</td>
</tr>
<tr>
<td></td>
<td>max_exwords</td>
<td>INTEGER</td>
<td></td>
<td></td>
<td>Computed field</td>
</tr>
<tr>
<td></td>
<td>num_labeled</td>
<td>INTEGER</td>
<td></td>
<td></td>
<td>Computed field</td>
</tr>
<tr>
<td></td>
<td>num_unlabeled</td>
<td>INTEGER</td>
<td></td>
<td></td>
<td>Computed field</td>
</tr>
</tbody>
</table>

158
<table>
<thead>
<tr>
<th>Table Name</th>
<th>Column Name</th>
<th>Data Type</th>
<th>Key</th>
<th>Constraints</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learn_Param</td>
<td>learn_param_id</td>
<td>CHAR(30)</td>
<td>PK</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>learn_algorithm</td>
<td>INTEGER</td>
<td></td>
<td>DEFAULT 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>loss_function</td>
<td>INTEGER</td>
<td></td>
<td>DEFAULT 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>loss_type</td>
<td>INTEGER</td>
<td></td>
<td>DEFAULT 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>error_margin</td>
<td>REAL</td>
<td></td>
<td>DEFAULT 0.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C_light</td>
<td>REAL</td>
<td></td>
<td>DEFAULT 0.01</td>
<td></td>
</tr>
<tr>
<td></td>
<td>accelerator</td>
<td>REAL</td>
<td></td>
<td>DEFAULT 0.0001</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ex_cache</td>
<td>BOOLEAN</td>
<td></td>
<td>DEFAULT true</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ex_cache_size</td>
<td>INTEGER</td>
<td></td>
<td>DEFAULT -1</td>
<td>-1 denotes full cache</td>
</tr>
<tr>
<td></td>
<td>supvec_cache</td>
<td>BOOLEAN</td>
<td></td>
<td>DEFAULT false</td>
<td></td>
</tr>
<tr>
<td></td>
<td>supvec_cache_size</td>
<td>INTEGER</td>
<td></td>
<td>DEFAULT 50</td>
<td>-1 denotes full cache</td>
</tr>
<tr>
<td></td>
<td>log_level</td>
<td>CHAR(10)</td>
<td></td>
<td></td>
<td>DEFAULT 'INFO'</td>
</tr>
<tr>
<td>SVM_Learn</td>
<td>svm_id</td>
<td>CHAR(30)</td>
<td>PK, FK</td>
<td>REFERENCES SVM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>learn_param_id</td>
<td>CHAR(30)</td>
<td>PK, FK</td>
<td>REFERENCES Learn_Param</td>
<td>Computed field</td>
</tr>
<tr>
<td></td>
<td>learning_start</td>
<td>TIMESTAMP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>learning_end</td>
<td>TIMESTAMP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVM_Model</td>
<td>svm_id</td>
<td>CHAR(30)</td>
<td>PK, FK</td>
<td>REFERENCES SVM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>learn_param_id</td>
<td>CHAR(30)</td>
<td>PK, FK</td>
<td>REFERENCES Learn_Param</td>
<td>Computed field</td>
</tr>
<tr>
<td></td>
<td>supvec_id</td>
<td>INTEGER</td>
<td>PK</td>
<td>NOT NULL</td>
<td>Computed field</td>
</tr>
<tr>
<td></td>
<td>selected</td>
<td>BOOLEAN</td>
<td></td>
<td>DEFAULT false</td>
<td>Computed field</td>
</tr>
<tr>
<td></td>
<td>alpha</td>
<td>DOUBLE PRECISION</td>
<td></td>
<td></td>
<td>Computed field</td>
</tr>
<tr>
<td></td>
<td>features</td>
<td>BYTEA</td>
<td></td>
<td></td>
<td>Computed field, WORD array</td>
</tr>
<tr>
<td>SVM_Model_Kernel</td>
<td>svm_id</td>
<td>CHAR(30)</td>
<td>PK, FK</td>
<td>REFERENCES SVM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>learn_param_id</td>
<td>CHAR(30)</td>
<td>PK, FK</td>
<td>REFERENCES Learn_Param</td>
<td>Computed field</td>
</tr>
<tr>
<td></td>
<td>supvec_id</td>
<td>INTEGER</td>
<td>PK</td>
<td>NOT NULL</td>
<td>Computed field</td>
</tr>
<tr>
<td></td>
<td>to_supvec_id</td>
<td>INTEGER</td>
<td></td>
<td>NOT NULL</td>
<td>Computed field</td>
</tr>
<tr>
<td></td>
<td>kernel_prod</td>
<td>DOUBLE PRECISION</td>
<td></td>
<td></td>
<td>Computed field</td>
</tr>
<tr>
<td>Table Name</td>
<td>Column Name</td>
<td>Data Type</td>
<td>Key</td>
<td>Constraints</td>
<td>Notes</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
<td>-----------------</td>
<td>----------</td>
<td>----------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>SVM_Classify</td>
<td>svm_id</td>
<td>CHAR(30)</td>
<td>PK, FK</td>
<td>REFERENCES SVM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>learn_param_id</td>
<td>CHAR(30)</td>
<td>PK, FK</td>
<td>REFERENCES Learn_Param</td>
<td></td>
</tr>
<tr>
<td></td>
<td>example_set_id</td>
<td>CHAR(30)</td>
<td>PK, FK</td>
<td>NOT NULL REFERENCES Example_Set</td>
<td></td>
</tr>
<tr>
<td></td>
<td>classification_start</td>
<td>TIMESTAMP</td>
<td></td>
<td></td>
<td>Computed field</td>
</tr>
<tr>
<td></td>
<td>classification_end</td>
<td>TIMESTAMP</td>
<td></td>
<td></td>
<td>Computed field</td>
</tr>
<tr>
<td>Classified_Example</td>
<td>svm_id</td>
<td>CHAR(30)</td>
<td>PK, FK</td>
<td>REFERENCES SVM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>learn_param_id</td>
<td>CHAR(30)</td>
<td>PK, FK</td>
<td>REFERENCES Learn_Param</td>
<td></td>
</tr>
<tr>
<td></td>
<td>example_set_id</td>
<td>CHAR(30)</td>
<td>PK, FK</td>
<td>NOT NULL REFERENCES Example_Set</td>
<td></td>
</tr>
<tr>
<td></td>
<td>example_id</td>
<td>INTEGER</td>
<td>PK, FK</td>
<td>NOT NULL REFERENCES Example</td>
<td></td>
</tr>
<tr>
<td></td>
<td>prediction</td>
<td>DOUBLE PRECISION</td>
<td></td>
<td></td>
<td>Computed field</td>
</tr>
<tr>
<td></td>
<td><strong>label</strong></td>
<td>CHAR(30)</td>
<td></td>
<td></td>
<td>Optional computed field holding the prediction to label translation,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>as defined by the label_set_id for the corresponding svm_id</td>
</tr>
</tbody>
</table>
Appendix B
NER Experimentation Datasets

One of the first steps in setting up experiments is to identify the datasets and evaluation tools to use in order to provide a fair comparison ground for NER systems. In this section we describe the three main datasets commonly used for experimentation purposes: the JNLPBA-04 task (Kim et al. 2004), the CoNLL-02 task (Tjong Kim Sang 2002a), and the CoNLL-03 task (Tjong Kim Sang and De Meulder 2003). The baseline experiments presented in Chapter 4 use the JNLPBA-04 training and testing datasets. The three datasets provide a suitable experimentation collection for language and domain independence. The JNLPBA-04 dataset represents the English biomedical domain and the CoNLL datasets represent the multilingual general domain. Successfully identifying named entities in the three datasets without incorporating domain or language knowledge will support the thesis of this research proposal.

B.1 BioNLP JNLPBA-04 Dataset

The expanding application of natural language processing in the biomedical literature and the emergence of new systems and techniques for information retrieval and extraction in this field raise the importance of having common and standardized evaluation and benchmarking methods in order to compare and evaluate the efficiency and effectiveness of the IR and IE methods used. Hirschman et al. (Hirschman et al. 2002) propose creating challenge evaluations specifically for the methods used to extract information in the biomedical field. Hirshman et al. identify the ingredients of a
successful evaluation as follows: a challenge problem, a task definition, training data, test
data, and an evaluation methodology.

The JNLPBA-04 challenge task (Kim et al. 2004) offers all of the necessary
ingredients identified by (Hirschman et al. 2002) for a successful evaluation. This
challenge task provides a standard testing environment for biomedical NER research. The
challenge task defines:

2. A task definition: identifying the names of proteins, cell lines, cell types, DNA
   and RNA entities in Medline abstracts.
3. Training and test data: annotated abstracts from the GENIA corpus (Kim et al.
   2003). The training and test datasets are described in more details below.
4. An evaluation methodology: a set of Perl scripts to evaluate the performance of
   the participating systems in terms of precision, recall, and F-score of their output
data. Since the named entities are often composed of multiple words, the
   evaluation scripts measure the performance relative to complete NER matches,
   left boundary matches, and right boundary matches.

The training and testing data use the GENIA annotated corpus (Kim et al. 2003) of
Medline articles (NLM 2007a), where the names of proteins, cell lines, cell types, DNA
and RNA entities are previously labeled. The named entities are often composed of a
sequence of words. The training data includes 2,000 annotated abstracts (consisting of
492, 551 tokens). The testing data includes 404 abstracts (consisting of 101, 039 tokens)
annotated for the same classes of entities: half of the test abstracts are from the same
domain as the training data and the other half of them are from the super-domain of
‘blood cells’ and ‘transcription factors’. The testing data sets are grouped in four subsets, covering abstracts from different year ranges. The fraction of positive examples with respect to the total number of tokens in the training set varies from about 0.2% to about 6%. Basic statistics about the data sets as well as the absolute and relative frequencies for named entities within each set can be found in Table B.1 and Table B.2.

### Table B.1 – Basic Statistics for the JNLPBA-04 Data Sets

<table>
<thead>
<tr>
<th></th>
<th># abstracts</th>
<th># sentences</th>
<th>#words</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Training Set</strong></td>
<td>2,000</td>
<td>20,546 (10.27/abs)</td>
<td>472,006 (236.00/abs) (22.97/sen)</td>
</tr>
<tr>
<td><strong>Test Set</strong></td>
<td>404</td>
<td>4,260 (10.54/abs)</td>
<td>96,780 (239.55/abs) (22.72/sen)</td>
</tr>
<tr>
<td>1990-1999</td>
<td>106</td>
<td>1,115 (10.52/abs)</td>
<td>25,080 (236.60/abs) (22.49/sen)</td>
</tr>
<tr>
<td>2000-2001</td>
<td>130</td>
<td>1,452 (11.17/abs)</td>
<td>33,380 (256.77/abs) (22.99/sen)</td>
</tr>
<tr>
<td>S/1998-2001</td>
<td>206</td>
<td>2,270 (11.02/abs)</td>
<td>51,957 (252.22/abs) (22.89/sen)</td>
</tr>
</tbody>
</table>

### Table B.2 – Absolute and Relative Frequencies for Named Entities Within Each Set

<table>
<thead>
<tr>
<th></th>
<th>protein</th>
<th>DNA</th>
<th>RNA</th>
<th>cell_type</th>
<th>cell_line</th>
<th>All NEs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Training Set</strong></td>
<td>30,269 (15.1)</td>
<td>9,533 (4.8)</td>
<td>951 (0.5)</td>
<td>6,718 (3.4)</td>
<td>3,830 (1.9)</td>
<td>51,301 (25.7)</td>
</tr>
<tr>
<td><strong>Test Set</strong></td>
<td>5,067 (12.5)</td>
<td>1,056 (2.6)</td>
<td>118 (0.5)</td>
<td>1,921 (4.8)</td>
<td>500 (1.2)</td>
<td>8,662 (21.4)</td>
</tr>
<tr>
<td>1978-1989</td>
<td>609 (5.9)</td>
<td>112 (1.1)</td>
<td>1 (0.0)</td>
<td>392 (3.8)</td>
<td>176 (1.7)</td>
<td>1,290 (12.4)</td>
</tr>
<tr>
<td>1990-1999</td>
<td>1,420 (13.4)</td>
<td>385 (3.6)</td>
<td>49 (0.5)</td>
<td>459 (4.3)</td>
<td>168 (1.6)</td>
<td>2,481 (23.4)</td>
</tr>
<tr>
<td>2000-2001</td>
<td>2,180 (16.8)</td>
<td>411 (3.2)</td>
<td>52 (0.4)</td>
<td>714 (5.5)</td>
<td>144 (1.1)</td>
<td>3,501 (26.9)</td>
</tr>
<tr>
<td>S/1998-2001</td>
<td>3,186 (15.5)</td>
<td>588 (2.9)</td>
<td>70 (0.3)</td>
<td>1,138 (5.5)</td>
<td>170 (0.8)</td>
<td>5,152 (25.0)</td>
</tr>
</tbody>
</table>

### B.2 CoNLL-02 Dataset

The CoNLL-02 shared task is a language-independent named entity recognition challenge presented as part of the 6th Workshop on Computational Language Learning. The task concentrates on four types of general named entities: persons, locations, organizations, and names of miscellaneous entities (Tjong Kim Sang 2002a). It offers

---

14 Ibid.
datasets for two European languages: Spanish and Dutch. The named entities are assumed to be non-recursive and non-overlapping.

The Spanish data is a collection of news wire articles from May 2000, made available by the Spanish EFE News Agency. The data contains words and entity tags only appearing on separate lines. The training dataset contains 273,037 lines, the development dataset contains 54,837 lines, and the test dataset contains 53,049 lines.

The Dutch data consist of four editions of the Belgian newspaper “De Morgen” from the year 2000, and contain words, entity tags and part-of-speech tags. The training dataset contains 218,737 lines, the development dataset contains 40,656 lines, and the test dataset contains 74,189 lines.

Named entities are tagged using either a B-XXX tag denoting the beginning of an entity (for e.g., B-PER denotes beginning a person’s name), or an I-XXX tag which indicates a continuation of the same entity started with the corresponding B-XXX tag. Words tagged with O are outside of named entities.

<table>
<thead>
<tr>
<th>Table B.3 – Number of Named Entities in the CoNLL-02 Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Spanish Datasets</strong></td>
</tr>
<tr>
<td>LOC</td>
</tr>
<tr>
<td>Training</td>
</tr>
<tr>
<td>4913</td>
</tr>
<tr>
<td>Development</td>
</tr>
<tr>
<td>984</td>
</tr>
<tr>
<td>Test</td>
</tr>
<tr>
<td>1084</td>
</tr>
</tbody>
</table>

**B.3 CoNLL-03 Dataset**

The CoNLL-03 shared task is another language-independent named entity recognition challenge offered as part of the 7th Conference on Natural Language Learning (Tjong Kim Sang and De Meulder 2003). The task concentrates on four general named entity types: persons, locations, organizations, and names of miscellaneous entities. It
offers datasets for two different European languages than those offered during the CoNLL-02 task, namely English and German. The data for each language consists of four data files for training, development, and test in addition to a large file with unannotated data. The task required using a machine learning approach that incorporates using the unannotated data file in the learning process.

The English data was taken from the Reuters Corpus, which consists of Reuters news stories from 1996 and 1997. The text for the German data was extracted from the 1992 German newspaper Frankfurter Rundschau. Table B.4 and Table B.5 summarize the data contents for each language. The English unannotated data file contains 17 million tokens and the German unannotated data file contains 14 million tokens (German).

The data files contain one word per line with empty lines representing sentence boundaries. Each line contains four fields: the word, its part-of-speech tag, its chunk tag and its named entity tag. Words tagged with O are outside of named entities and the I-XXX tag is used for words inside a named entity of type XXX. Whenever two entities of type XXX are immediately next to each other, the first word of the second entity is tagged B-XXX. The named entity annotation style is different than that used for the CoNLL-02 challenge task, where B-XXX tags were present for each named entity in the corpus and not just to split two consecutive entities.

The CoNLL-03 task data requires the Reuters corpus as the data files contain only references to the Reuters corpus and the scripts needed to extract, tokenize and annotate the data. In addition to the data files, the CoNLL-03 task provides gazetteers (reference lists) of known person names, locations, organizations, and miscellaneous entities.
Table B.4 – Basic Statistics for the CoNLL-03 Dataset\textsuperscript{15}

<table>
<thead>
<tr>
<th></th>
<th>English Datasets</th>
<th>German Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Articles</td>
<td>Sentences</td>
</tr>
<tr>
<td>Training</td>
<td>946</td>
<td>14,987</td>
</tr>
<tr>
<td>Development</td>
<td>216</td>
<td>3,466</td>
</tr>
<tr>
<td>Test</td>
<td>231</td>
<td>3,684</td>
</tr>
</tbody>
</table>

Table B.5 – Number of Named Entities in the CoNLL-03 Dataset\textsuperscript{16}

<table>
<thead>
<tr>
<th></th>
<th>English Datasets</th>
<th>German Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LOC</td>
<td>MISC</td>
</tr>
<tr>
<td>Training</td>
<td>7140</td>
<td>3438</td>
</tr>
<tr>
<td>Development</td>
<td>1837</td>
<td>922</td>
</tr>
<tr>
<td>Test</td>
<td>1668</td>
<td>702</td>
</tr>
</tbody>
</table>


\textsuperscript{16} Ibid.
Appendix C

Biomedical Entities in the GENIA Corpus

The following information about the GENIA corpus is extracted from the GENIA Project website at http://www-tsujii.is.s.u-tokyo.ac.jp/GENIA/home/wiki.cgi, and a technical report describing the GENIA ontology (Kim et al. 2006). “The GENIA project seeks to automatically extract useful information from texts written by scientists to help overcome the problems caused by information overload” (The GENIA Project 2002-2006).

Annotating (and identifying) biomedical named entities is not as simple as identifying general entities such as people names or locations. Biological terms in the GENIA corpus are semantically defined as the terms identifiable with any terminal concepts in GENIA ontology. Yet syntactically, they are not simply defined. Terms including biological entity names are mostly general nouns and they can appear in text with a variety of specifiers or qualifiers. To cite a couple of annotation challenges, biological named entities may appear nested, such as the example of IL-2 gene and IL-2 gene transcription, or appear in coordinated clauses, for example, ‘CD2 and CD25 receptors’ refers to two terms, CD2 receptors and CD25 receptors, but CD2 receptors doesn’t appear in the text.

C.1 The Current GENIA Ontology

The GENIA ontology (Kim et al. 2006), summarized in Table C.1, is a taxonomy of some entities involved in biological reactions concerning transcription factors in human blood cells. It was developed as the semantic classification used in the GENIA corpus. The five entities used for the NER task are highlighted.
### Table C.1 – The Current GENIA Ontology

<table>
<thead>
<tr>
<th>Source</th>
<th>Natural</th>
<th>Artificial</th>
</tr>
</thead>
<tbody>
<tr>
<td>organism</td>
<td>multi-cell organism</td>
<td>mono-cell organism</td>
</tr>
<tr>
<td>body part</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tissue</td>
<td></td>
<td>other (natural source)</td>
</tr>
<tr>
<td>cell type</td>
<td></td>
<td>cell line</td>
</tr>
<tr>
<td>cell component</td>
<td></td>
<td></td>
</tr>
<tr>
<td>other (natural source)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>source</td>
<td></td>
<td></td>
</tr>
<tr>
<td>substance</td>
<td>compound</td>
<td>organic</td>
</tr>
<tr>
<td>protein</td>
<td></td>
<td></td>
</tr>
<tr>
<td>amino acid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>peptide</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA family or group</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nucleic acid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNA family or group</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNA family or group</td>
<td></td>
<td></td>
</tr>
<tr>
<td>polynucleotide</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nucleotide</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lipid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>carbohydrate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>steroidal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>organic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>compound</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inorganic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
C.2 Entity Definitions from the GENIA Ontology

The five named entities extracted in the NER biomedical experiments are a subset of the GENIA ontology. The NER task entities are: protein, DNA, RNA, cell type, and cell line. The definitions come from the GENIA Project website (The GENIA Project 2002-2006) and the technical report describing the GENIA ontology (Kim et al. 2006).

1. **Protein**: Proteins include protein groups, families, molecules, complexes, and substructures. Corresponds to the Proteins category of MeSH.

   “Linear POLYPEPTIDES that are synthesized on RIBOSOMES and may be further modified, crosslinked, cleaved, or assembled into complex proteins with several subunits. The specific sequence of AMINO ACIDS determines the shape the polypeptide will take, during PROTEIN FOLDING, and the function of the protein.”

   - **protein family or group**: A family or a group of proteins, e.g., STATs
   - **protein complex**: A protein complex e.g., RNA polymerase II. The class includes conjugated proteins such as lipoproteins and glycoproteins.
   - **individual protein molecule**: An individual member of a group of non-complex proteins, e.g., STAT1, STAT2, STAT3, or a (non-complex) protein not regarded as a member of a particular group.
   - **subunit of protein complex**: A monomer in a complex, e.g., RNA polymerase II alpha subunit.
   - **substructure of protein**: A secondary structure or a combination of secondary structures, e.g. leucine-zipper, zinc-finger, alpha-helix, beta-sheet, helix-loop-helix.
• **domain or region of protein:** A tertiary structure that is supposed to have a particular function, e.g., SH2, SH3.

2. **DNA:** DNAs include DNA groups, families, molecules, domains, and regions. Corresponds to DNA category of MeSH.

   “A deoxyribonucleotide polymer that is the primary genetic material of all cells. Eukaryotic and prokaryotic organisms normally contain DNA in a double-stranded state, yet several important biological processes transiently involve single-stranded regions. DNA, which consists of a polysugar-phosphate backbone possessing projections of purines (adenine and guanine) and pyrimidines (thymine and cytosine), forms a double helix that is held together by hydrogen bonds between these purines and pyrimidines (adenine to thymine and guanine to cytosine).”

• **DNA family or group:** A family or a group of DNAs, e.g., myc family genes, rel family genes

• **individual DNA molecule:** An individual member of a family or a group of DNAs, e.g., AP-1/c-jun expression vector, AP2 cDNA

• **domain or region of DNA:** A substructure of DNA molecule which is supposed to have a particular function, such as a gene, e.g., c-jun gene, promoter region, Sp1 site, CA repeat. This class also includes a base sequence that has a particular function.

3. **RNA:** RNAs include RNA groups, families, molecules, domains, and regions. Corresponds to RNA category of MeSH.

   “A polynucleotide consisting essentially of chains with a repeating backbone of phosphate and ribose units to which nitrogenous bases are attached. RNA is unique among biological macromolecules in that it can encode genetic information, serve as an abundant structural component of cells, and also
possesses catalytic activity. (Rieger et al., Glossary of Genetics: Classical and Molecular, 5th ed)”

- **RNA family or group:** A family or a group of RNAs, e.g., tRNAs, viral RNA, HIV mRNA
- **individual RNA molecule:** An individual molecule of RNA, e.g., globlin mRNA, Oct-T1 transcript
- **domain or region of RNA:** A domain or a region of RNA, e.g., polyA site, alternative splicing site

4. **Cell type:** A cell type, e.g., T-lymphocyte, T cell, astrocyte, fibroblast. Corresponds to the Cells category of MeSH excluding the Cell Structure sub-category which is separated out into Cell_component.

“The fundamental, structural, and functional units or subunits of living organisms. They are composed of CYTOPLASM containing various ORGANELLES and a CELL MEMBRANE boundary.”

5. **Cell line:** The class includes cell strains and established cell cultures, e.g., HeLa cell, NIH 3T3, lymphoma line, human bone marrow culture. It corresponds to the Cells, Cultured category of MeSH.

“The fundamental, structural, and functional units or subunits of living organisms. They are composed of CYTOPLASM containing various ORGANELLES and a CELL MEMBRANE boundary.”

According to (Kim et al. 2006), the name of this class has been changed from Cell_line, as it covers not only cell lines but also other cultured cell types including clone or hybrid cells.
C.3 Sample Annotated Biomedical Text

The sample MEDLINE abstracts are extracted from the GENIA corpus. The five named entity types are highlighted in different colors according to the following legend:

Entity Types: protein, DNA, RNA, cell_line, cell_type

We present the sample abstracts in this format for better visibility. When consecutive words are highlighted, they represent one named entity and should be classified as such. Note that named entities may appear in different shapes and forms.

Sample #1: MEDLINE Abstract# 89078997

Octamer-binding proteins from B or HeLa cells stimulate transcription of the immunoglobulin heavy-chain promoter in vitro. The B-cell-type specificity of the immunoglobulin (Ig) heavy-chain and light-chain promoters is mediated by an octanucleotide (OCTA) element, ATGCAAAT, that is also a functional component of other RNA polymerase II promoters, such as snRNA and histone H2B promoters. Two nuclear proteins that bind specifically and with high affinity to the OCTA element have been identified. NF-A1 is present in a variety of cell types, whereas the presence of NF-A2 is essentially confined to B cells, leading to the hypothesis that NF-A2 activates cell-type-specific transcription of the Ig promoter and NF-A1 mediates the other responses of the OCTA element. Extracts of the B-cell line, BJA-B, contain high levels of NF-A2 and specifically transcribe Ig promoters. In contrast, extracts from HeLa cells transcribed the Ig promoter poorly. Surprisingly, addition of either affinity-enriched NF-A2 or NF-A1 to either a HeLa extract or a partially purified reaction system specifically stimulates the Ig promoter. This suggests that the constitutive OCTA-binding factor NF-A1 can activate transcription of the Ig promoter and that B-cell-specific transcription of this promoter, at least in vitro, is partially due to a quantitative difference in the amount of OCTA-binding protein. Because NF-A1 can stimulate Ig transcription, the inability of this factor to activate in vivo the Ig promoter to the same degree as the snRNA promoters probably reflects a difference in the context of the OCTA element in these two types of promoters.
Sample #2: MEDLINE Abstract# 98339383

Peripheral blood T cells and monocytes and B cell lines derived from patients with lupus express estrogen receptor transcripts similar to those of normal cells. OBJECTIVE: To identify and characterize estrogen receptor (ER) transcripts expressed in immune cells of patients with systemic lupus erythematosus (SLE) and healthy donors. METHODS: Peripheral blood monocytes and T cells were prepared from patients with SLE (n=6) and healthy donors (n=8). T cells were separated into CD4 and CD8. Some monocytes and T cells were stimulated with estradiol, PMA, and ionomycin. Epstein-Barr virus-transformed B cell lines (n=7) and B cell hybridomas (n=2) established from patients with SLE and a healthy individual were used as a B cell source. These cells were examined for ER mRNA by reverse transcription nested polymerase chain reaction. Amplified CDNA were sequenced by standard methods. RESULTS: In all cells tested, ER mRNA was expressed without prior in vitro stimulation. Partial sequences from exons 1-8 were nearly identical to the published sequence of the human ER mRNA. There were no notable differences in the ER transcripts between patients and healthy controls. Variant receptor transcripts lacking exon 5 or exon 7, which encodes the hormone binding domain, were identified in the majority of the cells. Precise deletion of the exons suggests that they are alternatively spliced transcripts. Whether the detected transcripts are translated into functional receptor proteins remains to be determined. In vitro stimulation did not affect ER mRNA expression. The presence of variants did not correlate with disease activity or medication. CONCLUSION: Monocytes, T cells, and B cells in patients express transcripts of the normal wild type ER and the hormone binding domain variants in vivo.

Sample #3: MEDLINE Abstract# 98342108

Human white blood cells and hair follicles are good sources of mRNA for the pterin carbinolamine dehydratase/dimerization cofactor of HNF1 for mutation detection. Pterin carbinolamine dehydratase/dimerization cofactor of HNF1 (PCD/DCoH) is a protein that has a dual function. It is a pterin 4alpha-carbinolamine dehydratase that is involved in the
regeneration of the cofactor tetrahydrobiopterin during the phenylalanine hydroxylase-catalyzed hydroxylation of phenylalanine. In addition, it is the dimerization cofactor of HNF1 that is able to activate the transcriptional activity of HNF1. Deficiencies in the gene for this dual functional protein result in hyperphenylalaninemia. Here we report for the first time that the PCD/DCoH mRNA is present in human white blood cells and hair follicles. Taking advantage of this finding, a sensitive, rapid and convenient method for screening mutations occurring in the coding region of this gene has been described.
Works Cited


Additional References


