Named Entity Recognition with Support Vector Machines

J O E L M I C K E L I N

KTH Computer Science
and Communication

Master of Science Thesis
Stockholm, Sweden 2013

Named Entity Recognition with Support Vector Machines

J O E L M I C K E L I N

DD221X, Master's Thesis in Computer Science (30 ECTS credits)
Degree Progr. in Engineering Physics 300 credits
Master Programme in Mathematics 120 credits
Royal Institute of Technology year 2013
Supervisor at CSC was Viggo Kann
Examiner was Anders Lansner

TRITA-CSC-E 2013:011
ISRN-KTH/CSC/E--13/011--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.kth.se/csc

Abstract

This report describes a degree project in Computer Science, the aim of which was to construct a system for Named Entity Recognition in Swedish texts of names of people, locations and organizations, as well as expressions for time. This system was constructed from the part-of-speech tagger *Granska* and the Support Vector Machine system *SVMlin*. The completed system was trained to recognize Named Entities by analyzing patterns in training corpora consisting of lists of example words belonging to each category. The system was initially trained to recognize patterns based on individual characters in words, but was later rewritten to recognize other characteristics of individual words such as the types of characters the words contained. When evaluating the system, it was determined that no incarnation of the system managed to perform satisfactorily when tested to recognize Named Entities of the aforementioned categories. A possible reason for this is that three of the categories, i.e. names of people, names of locations and names of organizations have few or no distinguishing features between them, which might warrant more research. The system proved apt when tested with solving the related problem of distinguishing email addresses from other named entities, indicating that the system might be of use in some cases of Named Entity Recognition.

Sammanfattning

Igenkänning av Named Entities med hjälp av supportvektormaskiner

Denna rapport beskriver ett examensarbete inom datalogi, målet med vilket var att konstruera ett system för igenkänning i svensk text av Named Entities för personnamn, platsnamn och namn på organisationer, samt tidsangivelser. Systemet konstruerades utgående från part-of-speech-taggaren *Granska* samt supportvektormaskinsystemet *SVMlin*. Det färdiga systemet tränades att känna igen Named Entities genom att analysera mönster i träningscorpora bestående av listor på exempelord tillhörande varje kategori. Systemet tränades först att känna igen mönster baserade på enskilda tecken i ord, men skrevs sedan om för att känna igen andra karakteristika hos enskilda ord såsom vilka slags tecken de innehåller. När systemet evaluerades framkom att ingen version av det fungerade tillfredsställande när det testades att känna igen Named Entities av ovan nämnda kategorier. En möjlig orsak till detta kan vara att tre av kategorierna, personnamn, platsnamn och namn på organisationer har få eller inga inneboende skillnader sinsemellan, vilket kan bli grund till mer forskning. Systemet visade sig dugligt när det prövades att lösa det relaterade problemet att särskilja mailadresser från andra Named Entities, vilket kan tyda på att systemet kan användas för viss typ av igenkänning av Named Entities.

Foreword

This report describes my degree project at KTH. The project is the final portion of my studies for a Master of Science in Engineering. It has been an interesting and challenging project, forcing me to read up on subjects of language parsing and machine learning; subjects with which I had little to no previous experience. I found the project to be a wonderful learning opportunity, as well as an opportunity for me to be creative and come up with interesting ideas. I would like to thank CSC for holding excellent classes during my studies and the opportunity to do this project, as well as my supervisor, Professor Viggo Kann for his excellent advice and criticism of my work. I want to thank Doctor Martin Hassel and Professor Hercules Dalianis of Stockholm University for providing me with the source code for the SweNam system. Doctor Hassel deserves further thanks for generously providing me with training and test corpora for my system. I also wish to thank Professor Dimitrios Kokkinakis of Gothenburg University for his aid with tagging my example texts with his system, thus providing invaluable material to compare my results with. My fellow students at Engineering Physics at KTH deserve thanks for all the laughs and silliness over the years, which have made my studies that much more enjoyable. Finally, I want to thank my boyfriend Robert, the love of my life, who provided trans-Atlantic support when I needed it the most, and who also helped proof read my text.

Contents

Chapter 1

Introduction

This project was carried out in cooperation with the School of Computer Science and Communication, CSC, at KTH, Stockholm. The purpose of the project was to create a system for *Named Entity Recognition* of texts written in Swedish.

1.1 Named Entity Recognition

1.1.1 Definition

Named Entity Recognition refers to a form of information extraction, namely the process of parsing a written text, and classifying the textual elements (called *Named Entities*) therefrom into a predefined set of categories. One category of named entities might be 'companies', a category which might include strings like 'Exxon Mobil', 'Apple', 'Google' and 'British Petroleum'. Another category might be 'Japanese family names', which would include strings like 'Tanaka', 'Kanno' and 'Watanabe'. Numerical data such as percentages and time expressions are also commonly regarded as named entities. There are two main approaches to attack this problem: Either one defines grammatical rules for each category and tries to find matches to these rules within the input texts, or one uses a statistical model based on machine learning. The former approach generally requires in-depth knowledge of linguistics, while the latter approach requires a large training corpus to train the system with [13].

1.1.2 History

The problem of Named Entity Recognition is a subset of the more general problem of information extraction, and the histories of the two are thus closely tied. The history of automatic information extraction goes back to the mid-eighties when early commercial systems for information extraction began to see the light of day. One notable example was the JASPER system used by the news agency Reuters for the purpose of extracting financial data in order to maintain a realtime flow of financial news [1]. Research into information extraction started gaining momentum in 1987 with the first *Message Understanding Conference*. These conferences, sponsored by the Defense Advanced Research Projects Agency (DARPA)

agency of the United States of America, had as their aim to fuel the development of better approaches to information extraction. At each of these conferences systems for information extraction would be formally evaluated by being tested to extract information from texts on a given topic. New categories of information to be extracted were added for each conference, with the sixth conference adding the task of Named Entity Recognition [11]. At the seventh and final Message Understanding Conference in 1997, a large-scale evaluation of systems for Named Entity Recogniton in English language texts was held, which showed that NER systems could yield results comparable with those of a human annotator [20]. Annual conferences for evaluating NER systems held since then have included the NIST sponsored *Automatic Content Extraction Program* held from 2000 until 2008 as well as its successor, the *Text Analysis Conference* which has been held since 2008 [3]. The languages for which Named Entity Recognizers have been implemented have been very diverse, including Arabic [22], Japanese [12], Swedish [7] and Czech [18] in addition to English. The types of textual contexts for which Named Entity Recognizers have been implemented have been just as diverse, ranging from financial texts [10] to clinical data [27]. Aside from the rule based approach to Named Entity Recognition, there have been many implementations of Named Entity Recognition with the help of methods like Bayesian networks [19], decision trees [12], artificial neural networks [22] and genetic algorithms [25]. This project is concerned with Named Entity Recognition with the help of *Support Vector Machines*. The Support Vector Machine is a non-probabilistic binary classifier, originally proposed in 1995 [6], which has been used succesfully for Named Entity Recognition in various contexts ever since [9, 18, 20].

1.2 My system

The goal of my project was to construct a system that takes a text written in Swedish as input and outputs a version of the same where the Named Entities corresponding to times, dates, organizations, locations and names of people are marked with XML tags. There are two problems inherent in the process of designing a system for Named Entity Recognition. The first is how the system should extract the Named Entities from the input texts, and the second is how the system should go about categorizing them. My system is based on two other systems: the rule-based Part-of-Speech tagger *Granska* and the Support Vector Machine system *SVMlin*. Granska is a Part-of-Speech tagger for the Swedish language developed at the Department of Numerical Analysis and Computer Science at KTH. It takes input texts and assigns very specific tags to linguistic elements within them [16]. By analyzing these tags, it is easy to see which elements are candidates for being Named Entities. These elements are then evaluated by SVMlin [26], which uses efficient statistical models for categorizing the elements. The rationale for merging two systems rather than build them from scratch was that the latter would require a significant amount of time, as well as a skillset I did not have.

1.3 Evaluation of my system

When evaluating systems of this kind, it is common to evaluate their performance relative to a human tagger. I also wanted to see how well the system did in comparison with rule-based Named Entity Recognition systems. Thus, I have performed the evaluation as follows: I have chosen twenty news articles from the *KTH News Corpus* (to which I was kindly given access by Dr. Martin Hassel), which is a database of archived news articles at KTH. I have then gone through these texts by hand and added a corresponding tag to each Named Entity found therein. After that, I have let three NER systems tag the texts: My own system, a rule-based system developed by Professor Kokkinakis of Gothenburg University, and the rule-based system SweNam developed by professor Hercules Dalianis and Erik Åström at KTH [7]. Based on how many of my manual tags each system reproduced, I could compute a measure of the system's quality, a so-called *F-score*. The F-score is a measure of both how many correct tags a system produces and how precise it is (precise in this case being defined as tagging only the elements we expect the system to tag).

Chapter 2

Theory

In this chapter I will describe the theory of the two systems I have merged, Granska and SVMlin. The former is a part-of-speech tagger, while the latter is a linear Support Vector Machine system. I used the Granska system at the suggestion of my supervisor, and it turned out to be more than apt at extracting Named Entities from texts. I chose to use an SVM system for the categorization of Named Entities since it has proven, at least for English, to yield performance results comparable with those of human taggers [20] and has also yielded favorable results in some contexts where Swedish texts are involved [27]. I chose SVMlin in particular since it was licensed under GPL, making it possible to integrate with Granska. I will also describe the theory of evaluation of systems of this kind and define a measure of the quality of the performance of any such system.

2.1 Part-of-speech-tagging

Part-of-speech tagging refers to the process of, for each atomic linguistic element in an example text (also known as *corpus*), assigning a marker or tag reflecting its syntactic role. By atomic linguistic elements, one means not only single words, but also punctuation. The first phase of any part-of-speech tagger is that of *tokenization*, in which punctuation is separated from words in the text (taking into account that punctuation may sometimes be a part of the word structure, as in abbreviations). This produces a string of words, which the system must then tag, choosing tags from a given *tagset*. A syntactic tagset contains markers for linguistic concepts such as *verb*, *noun*, *conjunction*, *interjection*, *pronoun* and the like. To give an example of tagging, suppose that a part-of-speech tagger gets the Swedish input string:
Jag åt en glass igår.
A tagged output string might look something like:
Jag <PRONOUN> *åt* <VERB/PASTTENSE> *en* <COUNTER> *glass* <NOUN> *igår* <TIME> . <PUNCTUATION>

Tagging would be a trivial task were it not for the problem of *ambiguity*. Ambiguity refers to the situation where it is possible to assign more than one tag to a given word. To take another example from Swedish, consider the word *få*.

Whether this word refers to an adjective (meaning *few*) or a verb (meaning *get* or *get to/be allowed to*) depends entirely on context. One of the challenges of any part-of-speech tagger is to resolve, or *disambiguate*, such ambiguities by taking context into account [13]. Taggers fall into two categories: *probabilistic taggers* and *rule based taggers*. Probabilistic taggers use probabilities computed from examining a large training corpus in order to assign the most probable tag to ambiguous elements. Rule based taggers use a complex set of grammar rules in order to assign the most probable tag. Granska is a hybrid system of a probabilistic Hidden Markov tagger and a rule based tagger [8].

2.2 Hidden Markov Model taggers

A *Hidden Markov Model* is based on the concept of *Bayesian Inference*, which dates back to the work of Bayes in the 18th century. The idea of Bayesian Inference is that it is possible to update the probability of a given hypothesis as new evidence is learned. The concept of Bayesian Inference is centered around *Bayes' Rule*:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \tag{2.1}$$

$P(x|y)$, also known as the *posterior*, is the probability of an event x occuring given that one has observed y. This is the probability of a certain hypothesis given the available evidence. $P(y|x)$ is known as the *likelihood*. It is the probability of observing an event y given that hypothesis x is true. $P(x)$ is the *prior*, the probability of hypothesis x being true given no evidence at all. In the case of part-of-speech taggers, x is a sequence of possible tags for an observed sequence of words y. If the tagger can maxmize the posterior function $P(x|y)$, it can make a guess as to the most probable tag sequence for the word string y. How, then, can the tagger compute the posterior? Firstly, the calculations are made easier by the fact that the denominator $P(y)$ never changes for a given string of words y. Hence, it can be ignored, and the function which is to be maximized is the product of the prior and likelihood, namely $P(y|x)P(x)$. $P(x)$ can be approximated as

$$P(x) \approx \prod_{i=1}^{n} P(x_i|x_{i-1}) \tag{2.2}$$

where x_i refers to the tag at index i in the sequence x. The likelihood $P(y|x)$ can be approximated as

$$P(y|x) \approx \prod_{i=1}^{n} P(y_i|x_i) \tag{2.3}$$

where x_i refers to the tag at index i in the sequence x and y_i refers to the tag at index i in the sequence y. These probabilities can be estimated by examining a large training corpus. A very basic way of doing so is to estimate $P(t_i|t_{i-1})$ as

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})} \tag{2.4}$$

where $C(t_{i-1}, t_i)$ is the number of times the tag t_i occurs after t_{i-1} in the corpus, and $C(t_{i-1})$ is the number of occurences of the tag t_{i-1} in the training corpus. In the same manner, one can estimate $P(w_i|t_i)$ as

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)} \tag{2.5}$$

where $C(t_i, w_i)$ is the number of times that the word w_i has the tag t_i in the training corpus. Computing the posterior this way is known as a *Markov Chain*.

There is, however, a further hindrance where part-of-speech tagging is concerned which makes Markov Chains inapplicable: If one wishes to compute the maximal posterior for a given word in a string of words, one needs to have knowledge of the correct assignment of tags for all the previous words in that same string (something which is only possible if all of those words are unambiguous). In a *Hidden Markov Model* or *HMM*, there is a distinction between *observed events* such as a sequence of words, and *hidden events*, such as an unknown sequence of tags. It can be formally described as a weighted automaton, containing the following components:

- $Q = \{q_1, \ldots, q_i, \ldots, q_N\}$, a set of N states and special states ,q_0 and q_F, which are start and end states for the automaton, respectively
- A transition probability matrix A, where $A_{i,j}$ is the probability of moving from state i to state j ($\sum_j A_{i,j} = 1 \forall i$)
- A sequence of observations o
- $B = b_i(o_t)$, a set of probabilities where $b_i(o_t)$ refers to the probability of getting the observation o_t in state i

To this, we add two assumptions. The first is the *Markov Assumption*, which states that the probability of ending up in a particular state depends only on the previous state, or more formally:

$$P(q_i|q_1 \ldots q_{i-1}) = P(q_i|q_{i-1}) \tag{2.6}$$

The second is the assumption of *output independence*, that any observation depends only on the current state in which it is observed. More formally:

$$P(o_k|q_1 \ldots q_T, o_1 \ldots o_T) = P(o_k|q_k) \tag{2.7}$$

This model is of course well suited to part-of-speech tagging. Suppose that the set of states is defined as the possible tags while the set of observations is the set of possible observed words. Then, A and B are simply the sets $P(t_i|t_{i-1})$ and $P(w_i|t_i)$, respectively.

What we need in order to be able to assign tag sequences to strings of words is, given an HMM, an algorithm for producing the most probable sequence of hidden states when taking the HMM and our observations as inputs. This is a process known as *decoding* and a common algorithm for accomplishing this is a dynamic programming algorithm known as the *Viterbi Algorithm*. The algorithm

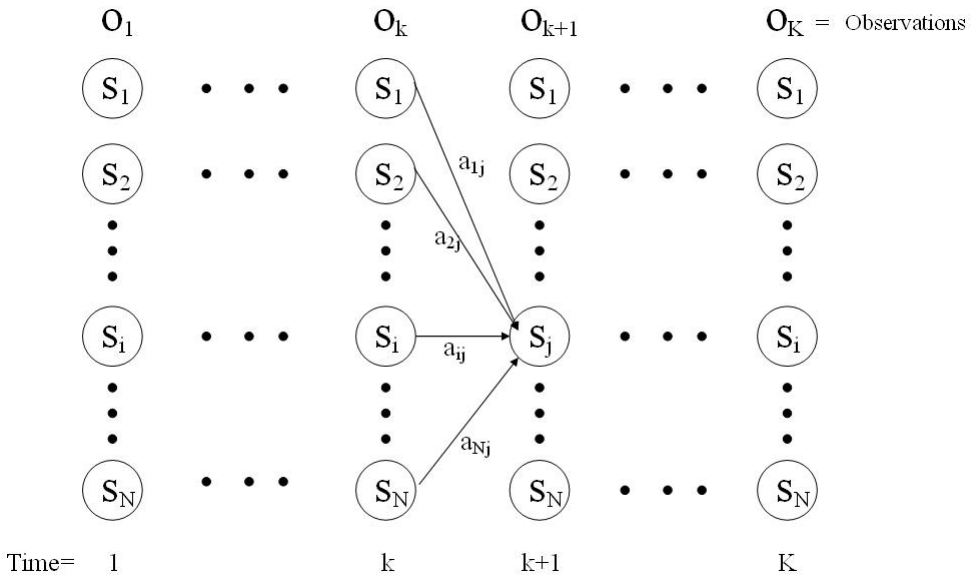

Figure 2.1: *Example of an HMM trellis. The weighted arrows represent probabilities of passing between states. Source: [29].*

can, in brief, be summarized as that it takes as input an HMM, with state set q, transitional matrix A and observation likelihoods B, as well as a set of observations o. It then builds a special graph, known as a *trellis* (illustrated in Fig. 2.1), where each node represents the quantity $v_t(j)$, the Viterbi propability of being in state i after having seen t observations. $v_t(j)$ is computed as:

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(j) A_{i,j} b_j(o_t) \tag{2.8}$$

In other words, the trellis is filled out from left to right. This means, of course, that the complexity of the algorithm is linear in the number of observations. It is thus very efficient and well suited for practical applications [13].

2.3 Support Vector Machines

SVMlin, the system used for classification of the named entities, is a *linear Support Vector Machine* [26]. This section describes how such a system works.

A Support Vector Machine, or SVM for short, is a system which is trained to classify input data into one of two categories. The SVM is trained on a large training corpus containing marked examples of the two categories. Each training example is represented by a point plotted into a hyperspace. The SVM then attempts to draw a hyperplane between the two categories, splitting the points as evenly as possible. For the two dimensional case, this is illustrated in Fig. 2.2.

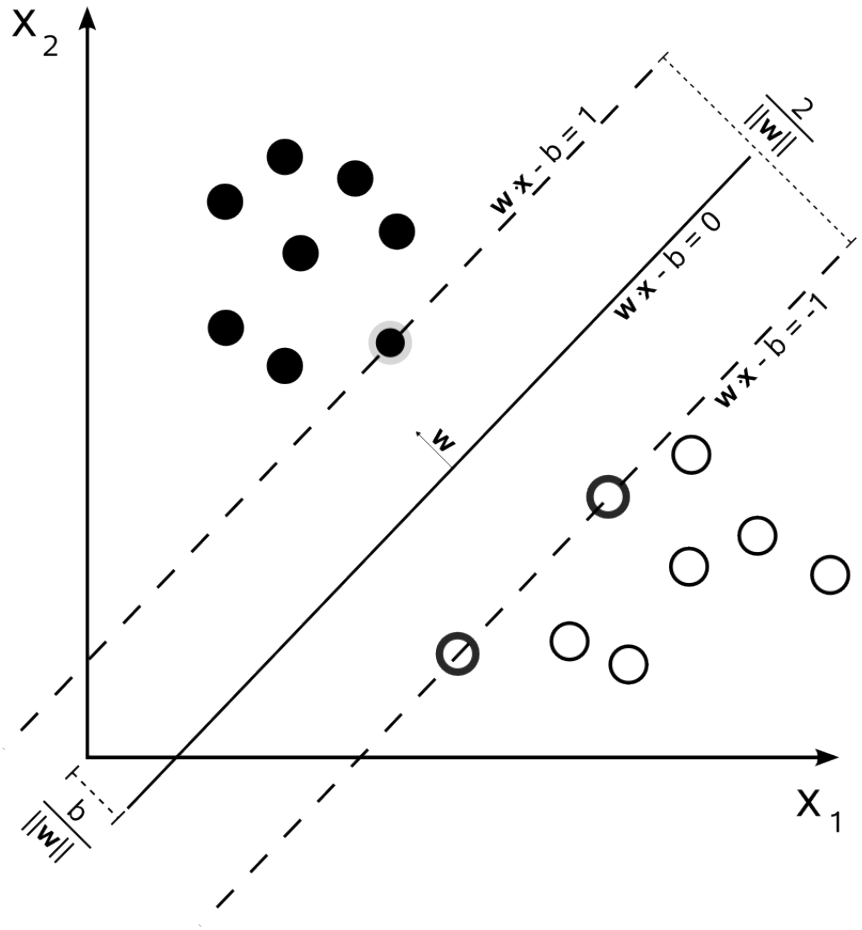

Figure 2.2: *A two dimensional SVM, with the hyperplanes representing the margin drawn as dashed lines. Source: [30].*

2.3.1 Formal description of Support Vector Machines

Suppose that there is training data for the SVM in the form of n k-dimensional real vectors $\mathbf{x_i}$ and integers y_i, where y_i is either 1 or -1. Whether y_i is positive or negative indicates the category for the vector i. The aim of the training phase of the SVM is to plot the vectors in a k-dimensional hyperspace and draw a hyperplane which as evenly as possible separates points from the two categories.

Suppose that this hyperplane has normal vector $\mathbf{w}$. Then the hyperplane can be written as the points $\mathbf{x}$ satisfying

$$\mathbf{w} \cdot \mathbf{x} - b = 0 \tag{2.9}$$

where $\frac{b}{\|\mathbf{w}\|}$ is the offset of the hyperplane from the origin along $\mathbf{w}$. We wish to choose this hyperplane as to maximize the margin between the points representing the two categories. Imagine two hyperplanes lying at the 'border' of two regions in each of which there are only points of either category. These two hyperplanes are perpendicular to $\mathbf{w}$ and cut through the outermost training data points in their respective regions. Two such planes can be seen illustrated as dashed lines in Fig. 2.1. Wanting to maximize the margin between the points representing the two categories is the same thing as wanting to keep these two hyperplanes as far

apart as possible. The training data points which end up on the dashed lines in Fig. 2.1 are called *support vectors*, hence the name Support Vector Machine. The hyperplanes can be described by the equations

$$\mathbf{w} \cdot \mathbf{x} - b = 1 \tag{2.10}$$

and

$$\mathbf{w} \cdot \mathbf{x} - b = -1 \tag{2.11}$$

The distance between the two is $\frac{2}{||\mathbf{w}||}$. Since the SVM wants to maximize the margin, we need to minimize $||\mathbf{w}||$. It also does not want to extend the margin indefinitely, since it does not want training data points within the margin. Thus, the following constraints are addded to the problem:

$$\mathbf{w} \cdot \mathbf{x_i} - b \geq 1 \tag{2.12}$$

for $\mathbf{x_i}$ in the first category, and

$$\mathbf{w} \cdot \mathbf{x_i} - b \leq -1 \tag{2.13}$$

for $\mathbf{x_i}$ in the second category. This can be rewritten as the optimization problem of minimizing $||\mathbf{w}||$ subject to

$$(\mathbf{w} \cdot \mathbf{x_i} - b)y_i \geq 1, (1 \leq i \leq n) \tag{2.14}$$

If one replaces $||\mathbf{w}||$ with $\frac{||\mathbf{w}||}{2}$, one can use Lagrange Multipliers to rewrite this optimization problem into the following quadratic optimization problem:

$$\min_{\mathbf{w},b} \max_{\alpha \geq 0} \left\{ \frac{||\mathbf{w}||^2}{2} - \sum_{i=1}^{n} \alpha_i((\mathbf{w} \cdot \mathbf{x_i} - b)y_i - 1) \right\}, \alpha_i \geq 0 \tag{2.15}$$

where the α_i are Lagrange multipliers [21]. Data sets which are possible to divide in two are called *linearly separable*. Depending on how the data is arranged, this may not be possible. It is, however, possible to use an alternative model involving a *soft margin* [6]. The soft margin model allows for a minimal number of mislabeled examples. This is done by introducing slack variables ξ_i for each training data vector $\mathbf{x}_i$. The function to be minimized, $\frac{||w||^2}{2}$ is modified by adding a term representing the slack variables. This can be done in several ways, but a common way is to introduce a linear function, so that the problem is to minimize:

$$\frac{||w||^2}{2} + C \sum_{i=1}^{n} \xi_i \tag{2.16}$$

for some constant C. To this minimization, the following modified constraint is added:

$$(\mathbf{w} \cdot \mathbf{x_i} - b)y_i \geq 1 - \xi_i, (1 \leq i \leq n) \tag{2.17}$$

By using Lagrange Multipliers as before, the problem can be rewritten as:

$$\min_{\mathbf{w},\xi,b} \max_{\alpha,\beta} \left\{ \frac{||w||^2}{2} + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} (\alpha_i(y_i(\mathbf{w} \cdot \mathbf{x_i} - b) - 1 + \xi_i) - \beta_i \xi_i) \right\} \tag{2.18}$$

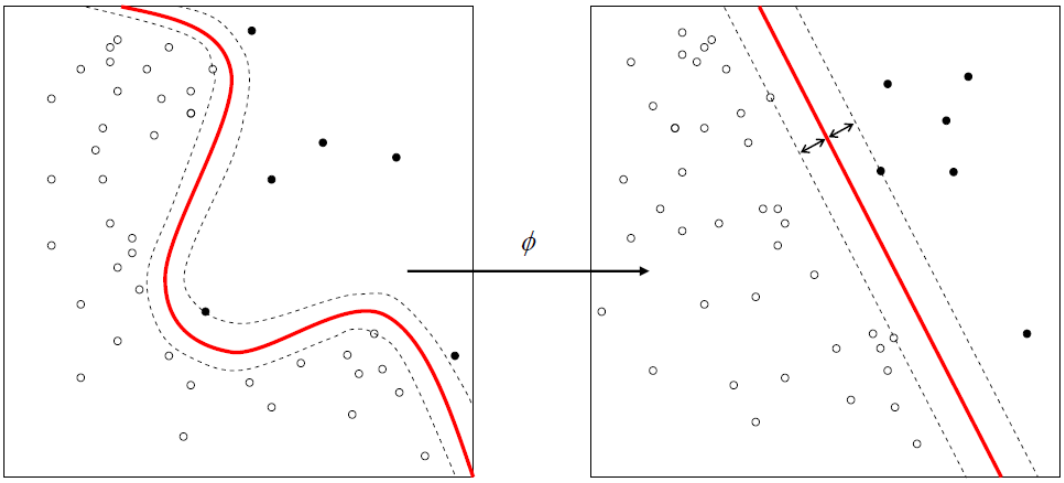

Figure 2.3: *The mapping ϕ of input data from the input space into an infinitely dimensional Hilbert Space in a non-linear SVM classifier. Source: [31].*

for $\alpha_i, \beta_i \geq 0$ [6]. To get rid of the slack variables, one can also rewrite this problem into its *dual form*:

$$\max_{\alpha} \left\{ \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x_i} \cdot \mathbf{x_j} \right\} \tag{2.19}$$

subject to the constraints

$$0 \leq \alpha_i \leq C, (1 \leq i \leq n) \tag{2.20}$$

and

$$\sum_{i=1}^{n} \alpha_i y_i = 0, (1 \leq i \leq n) \tag{2.21}$$

The system used in my project, SVMlin, is a linear classifier as above [26].

It is worth mentioning that there are also *non-linear classifiers*. While linear classifiers require that the data be linearly separable, or nearly so, in order to give qualitative results, the input data is transformed as to become linearly separable. In a non-linear classifier, the input vectors $\mathbf{x_i}$ are transformed as to lie in an infinitely dimensional Hilbert Space where it is always possible to linearly separate the two data categories [5]. An example of a non-linear system is *SVMlight* [15]. The mapping process of a non-linear classifier is seen in Fig. 2.3. Previous research shows that the input data of most text classification problems is linearly separable [23, 14], which is why non-linear classifiers were not tested in this project.

2.4 Generating feature vectors

When classifying any set of objects with an SVM system, one must first translate said objects to vector form. This is done by defining *feature vectors*, the coordinates of which are numeric representations of certain features inherent to the

objects in question. The objects can then be translated to feature vectors and thus be processed by the SVM system. A very simple example of this would be a bitmap image translated into a vector consisting of the color values of its pixels. Feature vectors can be designed entirely manually, or generated by an automatic process. A common way of doing the latter is to give a set of potential features as input to an optimizing function and determining which subset of these gives optimal performance. It is common to assign some score corresponding to quality to each subset, and then use the subset which obtains the highest such score, where good scores are usually obtained for feature vectors with as low dimensionality as possible. Exhaustive trial and error of all subsets is of course impractical, since the time complexity of such a search grows as $2^{|S|}$ for a set of potential features S. A popular way of overcoming this is to assign each subset a score which allows for the problem of finding an optimal subset to be rewritten as an optimization problem. One such method is known as *Minimum-redundancy-maximum-relevance* or *mRMR* selection. It is defined as follows: suppose we have a set of potential features S of some category of object c. For each subset S' of S we define the measure

$$D(S') = \frac{1}{|S'|} \sum_{f_i \in S'} I(f_i, c) \tag{2.22}$$

where f_i are features in S' and $I(f_i, c)$ is the *mutual information* between feature f_i and the category of object c. From statistics, it is known that the mutual information between two random variables is defined as

$$I(X; Y) = \sum_{x \in X, y \in Y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \tag{2.23}$$

for the discrete case, where $p(x)$ and $p(y)$ are the probability distribution functions for X and Y, and $p(x, y)$ is the joint probability distribution function for X and Y. We also define the following measure for each subset S':

$$R(S') = \frac{1}{|S'|^2} \sum_{f_i, f_j \in S'} I(f_i, f_j) \tag{2.24}$$

where $I(f_i, f_j)$ is the mutual information between feature f_i and feature f_j. The measure $D(S')$ represents *relevant* features while the measure $R(S')$ represents *redundant* features. The machine learning process is sped up when we the feature vectors *relevant* features as possible while eliminating as much *redundant* information as possible. To give a measure for this, we introduce the *mRMR* criterion, which says that optimality is reached for a feature subset F such that

$$D(F) - R(F) = \max_{S' \subset S} [D(S') - R(S')] \tag{2.25}$$

If variables x_i are introduced, this problem can be reduced to the optimization problem

$$D(F) - R(F) = \max_{x \in \{0,1\}^n} \left[\frac{\sum_{i=1}^n I(f_i, c) x_i}{\sum_{i=1}^n x_i} - \frac{\sum_{i,j=1}^n I(f_i, f_j) x_i x_j}{(\sum_{i=1}^n x_i)^2} \right] \tag{2.26}$$

given that $|S| = n$. The feature set F will then be uniquely determined by the variables x_i. If x_i is set to 1 then f_i will be in F. If x_i is set to 0 then f_i will be absent from F [24]. There are many other methods of automatically constructing feature vectors with a minimal number of redundant features; everything from greedy algorithms [28] to genetic programming [2] can be used for this purpose.

2.5 Theory of evaluation for Named Entity Recognizers

When evaluating all kinds of language parsers, a common method of evaluation is to compare the output of the parser with the output of a human performing the same task. Depending on the similarity between the two, one can then compute a score called an *F-score*. In this section, this measure is defined.

In the context of language parsers, two important concepts related to the F-score are *precision* and *recall*. A general definition of precision and recall is:

$$\text{precision} = \frac{|\text{relevant elements} \cap \text{retrieved elements}|}{|\text{retrieved elements}|} \tag{2.27}$$

$$\text{recall} = \frac{|\text{relevant elements} \cap \text{retrieved elements}|}{|\text{total available relevant elements}|} \tag{2.28}$$

High precision means that many of the found and tagged elements are tagged correctly, while high recall means that many of the NER elements in the text corpus used for evaluation are found and tagged correctly. In the context of Named Entity Recognizers, the retrieved elements are the produced annotations, while the relevant elements are the correct annotations. Precision can thus be seen as a measure of the degree to which the output produced contains the desired output, while recall can be seen as a measure of the degree to which the desired output is produced at all. The F-score will be defined as a weighted mean between precision and recall. A concept which must be introduced before the definition of the F-score is the *harmonic mean*. The harmonic mean is a special mean for a set of n non-zero reals a_i and is defined as:

$$HarmonicMean(\{a_i\}) = \frac{n}{\prod_i \frac{1}{a_i}} \tag{2.29}$$

The F-score is defined as a harmonic mean. If one abbreviates precision to P and recall to R, the F-score is simply the harmonic mean of αP and $(1 - \alpha)R$. The F-score F is thus defined:

$$F = \frac{1}{\frac{1}{\alpha P} \times \frac{1}{(1-\alpha)R}} \tag{2.30}$$

If β is defined as $\beta^2 = \frac{1-\alpha}{\alpha}$, F can be rewritten as a function of β:

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{2.31}$$

β is then a weight one can put to different values to favor either precision or recall. $\beta > 1$ favors recall, while $\beta < 1$ favors precision. $\beta = 1$ gives the F-score F_1:

$$F_1 = \frac{2PR}{P + R} \tag{2.32}$$

F_1 gives equal weight to precision and recall. The F-score has become praxis when evaluating information retrieval systems, as it is a measure which summarizes the quality of the output of the system. It is common to want to give equal weight to precision and recall when evaluating, and thus use F_1 [13].

Chapter 3

Method

This chapter describes the methods used for training and evaluating the system. The system was trained on a large training corpus consisting of words from three of the four categories of named entities in question, i.e. names of people, names of places and names of organizations. Since the Granska part of the system was already capable of finding instances of the fourth category, expressions for time, the SVM part of the system needed not be trained to recognize them. This is described in more detail in the chapter *System design*.

3.1 Training

In order to create feature vectors, a simple C program was written which takes long lists of instances of each category as input. The input given was a list of 500,000 Swedish first and last names, as well as lists of thousands of names of places and organizations. These lists were generously provided by Dr. Martin Hassel of Stockholm University. The program generated, for each entry in the lists given, a corresponding feature vector. All of these feature vectors were stored (in a format readable by the *SVMlin* system) in three files: *nameModelInput*, *locationModelInput* and *organizationModelInput*. These three files were identical. Also generated by the program were the three files *nameModelInputFlags*, *locationModelInputFlags* and *organizationModelInputFlags*. These latter three files contained long lists of flags, where each flag could be either 1 or -1. *nameModelInputFlags* contained one flag for each feature vector in *nameModel*. The flags in *nameModelInputFlags* corresponding to feature vectors generated from names of people were all set to 1, while the other flags were set to -1. The flags in *locationModelInputFlags* and *organizationModelInputFlags* were set analogously: The flags in *locationModelInputFlags* corresponding to feature vectors generated from names of places were all set to 1, as were the flags in *organizationModelInputFlags* corresponding to feature vectors generated from names of organizations, while all other flags in the two files were set to -1. The training function of *SVMlin* generates an SVM model from input consisting of a list of feature vectors and a list of corresponding flags, where each flag determines which of the two categories each feature vector belongs to. Thus, the files *nameModelInput* and *nameModelInputFlags* were given as input to *SVMlin*, which generated the SVM

model file *nameModel*. In the same way, *SVMlin* generated the model file *locationModel* from the input files *locationModelInput* and *locationModelInputFlags*, and the model file *organizationModel* from the input files *organizationModelInput* and *organizationnModelInputFlags*. It should be noted that the feature vectors are designed manually, with no feature selection taking place. The reason for this is that it did not seem practical to perform optimization on feature vectors with a low number of features, where none of the features depend on any parameters. The features are described in detail in the *Results* chapter.

3.2 Evaluation

A natural and standard method of evaluating Named Entity Recognizers and language parsers in general which has been used extensively to evaluate other systems in the past, is to compare the results given by the automated system with the results produced by a human annotator [20]. This method was also used to evaluate the performance of one of the systems I was comparing my system to, SweNam [7]. Thus, me and my supervisor agreed that it is an applicable method for evaluating the performance of my system. Thus, the evaluation of my system went as follows:

1. Twenty news articles were chosen from the KTH News Corpus. The process of choosing was this: Included in the KTH News Corpus, where most articles were stored in HTML form, were 100 articles which had been converted to plaintext. These 100 articles were given indices from 0 to 99 by me. A random number generator [1] was used to generate twenty indices between 0 and 99, and the articles with those indices were chosen to be included. These articles were then manually annotated by me. The annotations were XML tags of the kinds <TIME>, <LOCATION>, <NAME> and <ORGANIZATION>. These classes were chosen due to their use by the SweNam system as well as due to their abundance and varied use throughout the news articles.

2. The original news articles were tagged by my system, the rule-based system SweNam and a rule-based Named Entity tagger developed by Professor Kokkinakis [17].

3. The respective outputs from the three systems were assigned F-scores, which were then compared.

The tag <NAME> was added to names of people, whether a first name, last name or a first name followed by a last name, such as *Anna*, *Bertilsson* or *Bo Lundgren*. The tag <LOCATION> was added to the name of any specific place in the physical world, such as *Stockholm*, *Catalonia* or *Equatorial Guinea*. The tag <ORGANIZATION> was added to the name of any organization or governmental agency such as *Skattemyndigheten* or *Google*. The tag <TIME> was added to any specific reference to time, such as *år 2000*, *den 29:e juni* or 1989-07-12

[1] More specifically, the Unix command 'echo $(($RANDOM%100))'.

04:30. My system was not adapted to tag vague or relative references to time such as *nästa söndag* or *ibland* (meaning *next Sunday* and *sometimes*, respectively). It turned out that both SweNam and Professor Kokkinakis' system tagged some of the latter type of time references. I thought it most fair not to consider those taggings to be incorrect but to simply ignore them when counting the number of correct and incorrect tags. Two evaluations were performed, with a period in between in which I improved my system. When evaluating the output from each system, I first considered using the tool *AutoEval* which is built and maintained by the *Human Language Technology Group* at CSC, KTH [4], but later concluded that a manual counting would make it easier to properly judge which tags were correct and incorrect.

Chapter 4

System design

This chapter summarizes the design of the system. For clarity, this explained in steps corresponding with the process by which the Granska code was altered to incorporate SVM functionality and to tag named entities instead of parts of speech. The chapter ends with pseudocode which is intended to give an overview of the system functionality.

4.1 Modifying the output of Granska

The first step in modifying the functionality of the *Granska* system was for me to discover which function in the *Granska* code generated the textual output of the program. This was key to obtaining control over the functionality of *Granska*; if I could learn to modify its output, I would also be able to use that output for my own purposes. I eventually discovered that the output was generated in a function which I have labeled **Print**() in my pseudocode. To this function was fed C++ objects called *Words*. Each *Word* corresponded to an atomic text element from the input file, and contained three objects of note: a *string*, corresponding to the text element, a *tag* corresponding to the part-of-speech tag assigned to the element by *Granska*, and a *token*, which remained from the tokenization phase in *Granska*. Based on the contents of each *Word*, output data was fed to an output stream *out*. By finding the **Print**() function, I could feed whatever I wanted to *out*, and thus modify the program output.

4.2 Sifting out candidate elements for Named Entity Recognition

After studying the output of *Granska*, I could determine two things:

1. The elements I wanted to tag with <TIME> almost always received the *Granska* tokens 'TOKEN_DATE','TOKEN_YEAR' or 'TOKEN_TIME'. I could thus easily sift them out without having to resort to SVM methods.

2. The elements I wanted to tag with <NAME>, <LOCATION> or <ORGANIZATION> nearly always received the *Granska* tags 'pm.nom' or 'pm.gen'

(corresponding to Minimal Phrase: Nominative Case and Minimal Phrase: Genitive Case, respectively). I would thus only have to feed such elements to an SVM function.

Now that this was done, all that remained would be to invent a way to efficiently categorize the candidate elements.

4.3 Categorizing the candidate elements

I then wrote a function for creating feature vectors based on the *string* of each *Word*. In my pseudocode, I call this function **featurevec**(string *s*). Each featurevector was then fed to a function which tried to match it to several SVM models with the help of the code from *SVMlin*. In pseudocode, I call this function **SVMMatch**(featurevector *f*, SVMModel *M*), where the SVMModel objects have been constructed before the tagging started by feeding training data to the *SVMlin* system. Depending on whether the feature vector is matched to a model or not, the corresponding *string* is tagged or not tagged. The *string* is then fed to the output stream *out*.

A note on feature vector design

The feature vectors in question went through several revisions, all of which are elaborated upon in the *Results* chapter. The final version of the system uses feacture vectors which take into account such things as whether or not the word contains special characters, numbers, dashes or certain separators.

4.4 Pseudocode

Pseudocode for the **Print**() function is seen on the next page. The actual code was implemented in C code, built upon Granska's C++ code.

void **Print**()

```
outputstream out;
for Word w in getGranskaWord(){
  string tag="";
  string endtag="";
    if (w.token==("TOKEN_YEAR" or "TOKEN_DATE" or "TOKEN_TIME"))
        tag="<TIME>"; endtag="</TIME>";
    else if (w.word==("pm.nom" or "pm.gen")){
        featurvector f=featurevec(w.string);
        if (SVMMatch(f,nameModel)==true)
          tag="<NAME>"; endtag="</NAME>";
        else if (SVMMatch(f,organizationModel)==true)
          tag="<ORGANIZATION>"; endtag="</ORGANIZATION>";
        else if (SVMMatch(f,locationModel)==true)
          tag="<LOCATION>"; endtag="</LOCATION>";
       }
    out << tag << string << endtag;
}
return;
```

Chapter 5

Results

This chapter contains tables of the results of the first and second evaluations of the system, along with brief descriptions of how the system design looked when the evaluations were performed. The important difference between how the different versions of the system looked is how the feature vectors it uses were constructed. Therefore, the system descriptions in this chapter will only focus on that, as the full algorithm is described at greater length in the *System Design* chapter.

5.1 The first evaluation

At the time of the first evaluation, the system employed very primitive feature vectors, that is feature vectors based on the individual characters in each word. This first incarnation of feature vectors were not expected to yield very favorable results, but were merely used to show that the system was working. The function **featurevec1**($s(w)$) constructing these feature vectors took as input a word w converted to a C-style string s. The function **featurevec1**($s(w)$) was defined as

$$\textbf{featurevec1}(s(w)) = f \tag{5.1}$$

where f is a C-style string defined as

$$f_i = (\textbf{int})s_i \tag{5.2}$$

that is, the entry at index i in f is simply the element at index i in s, cast as an integer in C. These vectors did indeed yield very poor performance results for the system. The system obtained a precision of 0.23 and a recall of 0.20, yielding an F-score of

$$F_{fvec1} = \frac{2 \cdot 0.23 \cdot 0.20}{0.23 + 0.20} \approx 0.21 \tag{5.3}$$

with the F-score defined as in the *Theory* chapter. By contrast, the SweNam system obtained a precision of 0.75 and a recall of 0.45, yielding an F-score of

$$F_{SweNam} = \frac{2 \cdot 0.75 \cdot 0.45}{0.75 + 0.45} \approx 0.56 \tag{5.4}$$

while Kokkinakis' system obtained a precision of 0.95 and a recall of 0.88, yielding an F-score of

$$F_{Kokkinakis} = \frac{2 \cdot 0.95 \cdot 0.88}{0.95 + 0.88} \approx 0.91 \tag{5.5}$$

These results are summarized in the table below.

	Precision	Recall	F-score
My system	0.23	0.20	0.21
SweNam	0.75	0.45	0.56
Kokkinakis' system	0.95	0.88	0.91

Table 5.1: *The precision, recall and F-score of the first version of my system when used to find names of people, locations and organizations, with the corresponding numbers for SweNam and Kokkinakis' system provided for comparison*

5.2 The second evaluation

With the results of the first versions of feature vectors being so poor in comparison with rule based systems, I consulted with my supervisor and we agreed to next use features not based on individual characters, but rather on characteristics of whole words. The function **featurevec2**$(s(w), L)$ used in the second evaluation took as input not only a C-style string s of a word w, but also a very large language corpus L, where L contains many example texts of the language from which the Named Entities were intended to be extracted. The function **featurevec2**$(s(w), L)$ was defined as

$$\textbf{featurevec2}(s(w), L) = f \tag{5.6}$$

where f is a vector of length 11, defined as

$$f_1 = \begin{cases} 1 \text{ if } s \text{ contains special characters} \\ 0 \text{ if } s \text{ does not contain special characters} \end{cases} \tag{5.7}$$

$$f_2 = \begin{cases} 1 \text{ if } s \text{ contains only capital letters} \\ 0 \text{ if } s \text{ does not only contain capital letters} \end{cases} \tag{5.8}$$

$$f_3 = \begin{cases} 1 \text{ if } s \text{ contains capital letters} \\ 0 \text{ if } s \text{ does not contain capital letters} \end{cases} \tag{5.9}$$

$$f_4 = \begin{cases} 1 \text{ if } s \text{ contains numbers} \\ 0 \text{ if } s \text{ does not contain numbers} \end{cases} \tag{5.10}$$

$$f_5 = \begin{cases} 1 \text{ if } s \text{ begins with a number} \\ 0 \text{ if } s \text{ does not begin with a number} \end{cases} \tag{5.11}$$

$$f_6 = \begin{cases} 1 \text{ if } s \text{ contains only numbers and dashes} \\ 0 \text{ if } s \text{ does not only contain numbers and dashes} \end{cases} \tag{5.12}$$

$$f_7 = \begin{cases} 1 \text{ if } s \text{ contains dashes} \\ 0 \text{ if } s \text{ does not contain dashes} \end{cases} \tag{5.13}$$

$$f_8 = \begin{cases} 1 \text{ if } s \text{ contains colons} \\ 0 \text{ if } s \text{ does not contain colons} \end{cases} \tag{5.14}$$

$$f_9 = \begin{cases} 1 \text{ if } s \text{ contains forward slashes} \\ 0 \text{ if } s \text{ does not contain forward slashes} \end{cases} \tag{5.15}$$

$$f_{10} = \frac{\text{The number of occurrences of } s \text{ in } L}{\text{The total number of words in } L} \tag{5.16}$$

$$f_{11} = \begin{cases} \text{The average position in a sentence of } s \text{ in } L \text{ if } s \text{ occurs in } L \\ 0 \text{ if } s \text{ does not occur in } L \end{cases} \tag{5.17}$$

When implemented, these new feature vectors gave the system a precision of 0.22 and a recall of 0.20, giving an F-score of

$$F_{fvec1} = \frac{2 \cdot 0.22 \cdot 0.20}{0.22 + 0.20} \approx 0.20 \tag{5.18}$$

The corpus L used in this test consisted of the entire KTH News Corpus. These results are summarized in the table below.

	Precision	Recall	F-score
My system	0.22	0.20	0.20
SweNam	0.75	0.45	0.56
Kokkinakis' system	0.95	0.88	0.91

Table 5.2: *The precision, recall and F-score of the second version of my system when used to find names of people, locations and organizations, with the corresponding numbers for SweNam and Kokkinakis' system provided for comparison*

5.3 Retraining the system to recognize email addresses

Since the more sophisticated feature vectors used in the second evaluation did not improve the performance of the system, I had the idea that this might be due to the categories of named entities tagged being too similar and thus hard to differentiate in a text without either prior knowledge or taking into account the entire textual context, which would explain why the rule based systems performed better. This idea will be further elaborated upon in the *Discussion* chapter. I wanted to test if my system would be more useful when categorizing named entities with obvious differences between them. After consulting my supervisor I decided to retrain my system to differentiate email addresses from other named entities. For this purpose, I made a third incarnation of the feature vectors in which the entries depending on a large text corpus L were removed, since they did not seem to improve the result by much yet increased the time complexity of the system greatly. I also added an entry for whether or not the word contains the @ character and one for whether or not the word contains dots. The function **featurevec3**$(s(w))$ is defined as

$$\textbf{featurevec3}(s(w)) = f \tag{5.19}$$

where f is a vector of length 11, defined as

$$f_1 = \begin{cases} 1 \text{ if } s \text{ contains special characters} \\ 0 \text{ if } s \text{ does not contain special characters} \end{cases} \tag{5.20}$$

$$f_2 = \begin{cases} 1 \text{ if } s \text{ contains only capital letters} \\ 0 \text{ if } s \text{ does not only contain capital letters} \end{cases} \tag{5.21}$$

$$f_3 = \begin{cases} 1 \text{ if } s \text{ contains capital letters} \\ 0 \text{ if } s \text{ does not contain capital letters} \end{cases} \tag{5.22}$$

$$f_4 = \begin{cases} 1 \text{ if } s \text{ contains numbers} \\ 0 \text{ if } s \text{ does not contain numbers} \end{cases} \tag{5.23}$$

$$f_5 = \begin{cases} 1 \text{ if } s \text{ begins with a number} \\ 0 \text{ if } s \text{ does not begin with a number} \end{cases} \tag{5.24}$$

$$f_6 = \begin{cases} 1 \text{ if } s \text{ contains only numbers and dashes} \\ 0 \text{ if } s \text{ does not only contain numbers and dashes} \end{cases} \tag{5.25}$$

$$f_7 = \begin{cases} 1 \text{ if } s \text{ contains dashes} \\ 0 \text{ if } s \text{ does not contain dashes} \end{cases} \tag{5.26}$$

$$f_8 = \begin{cases} 1 \text{ if } s \text{ contains colons} \\ 0 \text{ if } s \text{ does not contain colons} \end{cases} \tag{5.27}$$

$$f_9 = \begin{cases} 1 \text{ if } s \text{ contains forward slashes} \\ 0 \text{ if } s \text{ does not contain forward slashes} \end{cases} \tag{5.28}$$

$$f_{10} = \begin{cases} 1 \text{ if } s \text{ contains @} \\ 0 \text{ if } s \text{ does not contain @} \end{cases} \tag{5.29}$$

$$f_{11} = \begin{cases} 1 \text{ if } s \text{ contains dots} \\ 0 \text{ if } s \text{ does not contain dots} \end{cases} \tag{5.30}$$

To the lists of entities described in the *Method* chapter was added a list of 500,000 email addresses obtained from the Internet, and a new model file named *mailModel* was added to the previous three. I added two email addresses to random positions in each of the twenty news articles I had previously tested the system on (as mentioned in the *Method* chapter) and calculated an F-score based on how correct the system was in tagging these addresses using the new feature vectors. This gave a recall of 0.90 and a precision of 1, yielding the F-score of

$$F_{email} = \frac{2 \cdot 0.90 \cdot 1}{0.90 + 1} \approx 0.95 \tag{5.31}$$

I wanted to see if web addresses, having a similar structure to email addresses, could also be found by this approach. Hence, I changed ten of the email addresses in the articles to web addresses instead, to see what effect this had on the system performance. This gave a recall of 0.85 and a precision of 0.89, giving an F-score of

$$F_{email/web} = \frac{2 \cdot 0.85 \cdot 0.94}{0.85 + 0.94} \approx 0.89 \tag{5.32}$$

These results are summarized in the table below.

	Precision	Recall	F-score
When finding email addresses	1.00	0.90	0.95
When also finding web addresses	0.94	0.85	0.89

Table 5.3: *The precision, recall and F-score of the system when used to find email addresses as well as a combination of email addresses and web addresses*

Chapter 6

Discussion

This chapter contains conclusions drawn from the process of designing and evaluating my system, as well as suggestions for further research and uses for the system. A hypothesis is presented in an attempt to explain the reason why the system could not adequately differentiate between the names of people, locations and organizations but had little trouble differentiating email addresses from other Named Entities. Further research is called for in order to investigate whether this hypothesis is valid.

6.1 The problem of categorizing names of people, locations and organizations and the need for context

The failure of my system to obtain results comparable to those of rule-based systems when attempting to differentiate between the names of people, places and organizations is in my opinion an interesting one in that it might be a result of an inherent lack of distinguishing features between these three categories of Named Entities. I myself had a very hard time when trying to come up with features which might properly reflect the differences between them. My hypothesis is that they are all extremely similar; they are mostly composed of alphabetical characters only, with the exception of the odd hyphen here and there, as well as some company names containing several capital letters or sometimes even numbers and some names of people being variations of one another. Mostly they begin with a capital letter, and there is no grammatical restriction on where they appear in a sentence. Furthermore, the frequency with which they appear in a given text depends to a large extent on the author. Any person, location or organization once named in a text can, if the author chooses to write the text that way, be referred back to without being named again. The problem is further complicated due to the fact that a given entity might belong to several categories. There is, for example, no law forbidding a person to name a newly founded company after him- or herself, and there are also several geographical places named after famous people worldwide. As the excellent performance of the rule-based systems show, however, it is not at all impossible to program a computer to differentiate between these three

categories with a performance comparable to that of a human, and I believe that the key to the success of such systems is in fact that they emulate human behavior by taking context into account. Imagine, for example, how a small child or someone else with a very limited exposure to the world would tell the difference. If there is indeed no inherent difference between the categories themselves, this person would have no choice but to take context into account, which is exactly what a rule-based system also does. Previous research shows that support vector machines can be successful in finding these three categories provided they either take context into account or compare found entities with lists of words [9, 18]. It did not occur to me to design my system this way, as I thought such an approach to be too similar to a rule-based system. I would welcome further research in order to determine whether one really has to resort to context based methods in order to tell the difference between the three aforementioned categories, or whether there there is in fact a way to tell the difference statistically. Research into how the human brain determines the difference between Named Entities with no obvious general differences between them might be useful for this purpose.

6.2 The usefulness of my system

The failure of my system to solve the initial problem of being able to categorize the original three categories does not, in my opinion, mean that the system is useless for further research. As the success with differentiating email and web addresses from other Named Entities shows, the system might have use in Named Entity Recognition provided that there is an inherent difference between the categories looked for and that the feature vectors used are so defined as to properly reflect these differences. The recognition of email and web addresses is of course an extremely simple example, especially as the same results could have been obtained with regular expressions as well, but it serves as a proof of concept on which further research could be built. The success of text recognition by the use of SVM systems in other contexts [20, 27, 9, 18, 14] makes it probable that further research with my approach would be a worthwhile pursuit. I invite anyone who might be interested to improve and test the capabilities of my system with other categories and feature vectors than those used by me in this project.

6.3 Other statistical approaches the system could be compared with

The scope of this project was only to compare an SVM based system with two strictly rule-based systems. These two approaches are naturally not the only ones which might be suitable to use for Named Entity Recognition. Named Entity Recognition based on other methods such as decision trees, artificial neuron networks, Bayesian networks and genetic algorithms have been successfully implemented in the past [12, 22, 19, 25]. I would think it interesting to see how well systems based on these algorithms are at finding the three categories of named en-

tities my system had trouble differentiating between. I would welcome any such research.

6.4 In summary

I interpret the results of my evaluations such that the feature vectors used in the system seem to be the limiting factor for the quality of the Named Entity Recognition. The extraction of Named Entities works all but flawlessly with the help of *Granska*, but it is key for the differentiation between categories to yield favorable results that the feature vectors accurately reflect the inherent differences between categories of Named Entities. I welcome more research on this, and invite others to use my system for that purpose, as well as to compare the performance of the system to that of other machine learning approaches such as artifical neuron networks and decision trees. I also think that research into whether there are sets of categories with no discernible differences between them might yield interesting results.

Chapter 7

References

Bibliography

[1] P. Andersen, P. Hayes, A. Huettner et al. Automatic Extraction of Facts from Press Releases to Generate News Stories, *Proceeding of the Third Conference on Applied Natural Language Processing*, 1992

[2] M. Aurnhammer, Evolving Texture Features by Genetic Programming, *Proceedings of the 2007 EvoWorkshops 2007 on EvoCoMnet*, EvoFIN, EvoIASP,EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog: Applications of Evolutionary Computing, 2007

[3] *Automatic Content Extraction 2008 Evaluation Plan (ACE08)*, NIST, 2008

[4] J. Bigert, L. Ericson, A. Solis AutoEval and Missplel: Two Generic Tools for Automatic Evaluation *In proceedings of Nodalida, Reykjavik, Iceland*, May 2003

[5] B. Boser, I. Guyon, V. Vapnik A training algorithm for optimal margin classifiers *5th Annual ACM Workshop on COLT*, pages 144–152, Pittsburgh, PA, 1992

[6] C. Cortes, V. Vapnik, Support-Vector Networks, *Machine Learning*, 20, 1995

[7] H. Dalianis, E. Åström SweNam - A Swedish Named Entity Recognizer *IPLab*, 2001

[8] R. Domeij, O. Knutsson, J. Carlberger, V. Kann, Granska - an efficient hybrid system for Swedish grammar checking, *NoDaLiDa 99*, 49-56, 1999

[9] A. Ekbal, S. Bandyopadhyay, Named Entity Recognition using Support Vector Machine: A Language Independent Approach, *International Journal of Electrical and Electronics Engineering* 4:2, 2010

[10] D. Farmakiotou, V. Karkalestis, J. Koutsias et al. Rule-Based Named Entity Recognition For Greek Financial Texts, *Proceedings of the Workshop on Computational Lexicography and Multimedia Dictionaries*, 2000

[11] R. Grishman, B. Sundheim, Message Understanding Conference - 6: A Brief History, *Proceedings of the 16th International Conference on Computational Linguistics*, I, pages 466–471, Copenhagen, 1996

[12] H. Isozaki, Japanese named entity recognition based on a simple rule generator and decision tree learning, *ACL '01 Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, Stroudsburg, Philadelphia, USA, 2001

[13] D. Jurafsky, J. Martin *Speech and Language Processing, Second Edition* Prentice Hall, New Jersey, 2009

[14] T. Joachims, Text categorization with support vector machines: Learning with many relevant features, *Proceedings of the European Conference on Machine Learning*, pages 137–142, Berlin, 1998

[15] T. Joachims Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning, *B. Schölkopf and C. Burges and A. Smola (ed.), MIT-Press*, 1999

[16] O. Knutsson, J. Bigert, V. Kann, A robust shallow parser for Swedish *NoDaLiDa 03*, Reykjavik, Iceland, May 2003

[17] D. Kokkinakis, Reducing the Effect of Name Explosion, *Proceedings of the LREC Workshop: Beyond Named Entity Recognition, Semantic labelling for NLP tasks. Fourth Language Resources and Evaluation Conference (LREC)*. Pp. 1-6. Lissabon, Portugal, 2004

[18] J. Kravalová, Z. Žabokrtský, Czech Named Entity Corpus and SVM-based Recognizer, *ACL-IJCNLP 2009 workshop: Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration*, pages 194-201, Suntec, Singapore, 2009

[19] A. Louis, A. De Waal, C. Venter, Named entity recognition in a South African context, *SAICSIT '06 Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, pages 170-179, 2006

[20] E. Marsh, D. Perzanowski, MUC-7 Evaluation of IE Technology: Overview of Results, *MUC-7*, April 1998

[21] S. Marsland, *Machine Learning, an Algorithmic Perspective, 1st edition*, Chapman & Hall, London, 2009

[22] N.F. Mohammed, N. Omar, Arabic Named Entity Recognition Using Artificial Neural Network, *J. Comput. Sci.*, 8, pages 1285-1293, 2012

[23] P.Y. Pawar, S.H. Gawande, A Comparative Study on Different Types of Approaches to Text Categorization, *International Journal of Machine Learning and Computing*, Vol. 2, No. 4, August 2012

[24] H.C. Peng, F. Long, C. Ding, Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 27, No. 8, pp. 1226–1238, 2005

[25] S. Saha, U.K. Sikdar, M. Hasanuzzaman, *22nd IEEE International Conference on Tools with Artificial Intelligence*, pages 354-355, October 2010

[26] V. Sindhwani, P. Niyogi, M. Belkin, SVMlin: Fast Linear SVM Solvers for Supervised and Semi-supervised Learning *Workshop on Machine Learning Open Source Software, Neural Information Processing Systems (NIPS)*, 2006

[27] M. Skeppstedt Negation detection in Swedish clinical text: An adaption of NegEx to Swedish *Journal of Biomedical Semantics*, 2011

[28] T. Zhang, On the Consistency of Feature Selection using Greedy Least Squares Regression, *Journal of Machine Learning Research* 10, pages 555-568, 2009

Figure sources

[29] http://bioinfopakistan.ucoz.com/_nw/1/04621520.jpg

[30] http://en.wikipedia.org/wiki/File:Svm_max_sep_hyperplane_with_margin.png

[31] http://en.wikipedia.org/wiki/File:Kernel_Machine.png

TRITA-CSC-E 2013:011
ISRN-KTH/CSC/E--13/011-SE
ISSN-1653-5715

www.kth.se