

Named Entity Extraction from Text Messages

Tobias Ek (tobias.ek@telia.com)
Camilla Kirkegaard (kirkegaard@ejp.se)

Examensarbete för 30 hp
Institutionen för datavetenskap, Naturvetenskapliga fakulteten, Lunds Universitet

Thesis for a diploma in computer science, 30 ECTS credits
Department of computer science, Faculty of science, Lund University

Abstract

The primary goal of this Master's thesis was to investigate the possibility to have a viable *named entity recognition* (NER) system for cellular text messages that would be appropriate for commercial use. Most NER system deals with text that is structured, formal, well written, with a good grammatical structure, and with few spelling errors. Many text messages lack these qualities and have instead a short handed language, mixed language, and emoticons.

We wanted to examine if it was possible to incorporate the techniques and tools required into a cellular telephone and to have the system analyze the incoming text messages within a reasonable response time.

The *named entity* focus was on locations, names, dates, times, and telephone numbers with the idea that extraction of these entities could be utilized by other applications such as meeting details for calendar, location look-up via Google map, or contact information for the cellular telephone's record of contacts.

We visualized the results using the short message service (SMS) application which, if the user chooses to, can send the extracted information to the appropriate application. We reached an F-score of 86% for strict matches and 89% for partial matches.

Sammanfattning

Målet med denna magisteruppsats har varit att undersöka möjligheterna till att ha en fungerande *named entity recognition* (NER) system för textmeddelanden som är användbart för kommersiellt användande. De flesta existerande NER systemen idag hanterar text som är strukturerad, formell, välskriven, med korrekt grammatik och med endast ett fåtal stavfel. Många textmeddelanden saknar dessa kvaliteter och har istället ett kortfattat språk, med slang, låneord och smileys.

Vi ville vi undersöka om det gick att bygga in den teknik och verktyg som behövdes i en mobiltelefon och att låta systemet analysera inkommande textmeddelanden inom en rimlig svarstid.

Vårt fokus på vilka entiteter vi ville extrahera låg på platser, datum, tidstuttryck, telefonnummer samt platser. Målet var att dessa entiteter sedan kunde användas av andra applikationer som t.ex. den inbyggda kalenderapplikationen, mobilens telefonbok eller Google maps.

Vi visualiserade detta med en short message service (SMS) applikation som på användarens förfrågan kan skicka vidare den extraherade informationen till en lämplig applikation. Vi uppnådde ett F-score på 86% för *strict matches* och 89% för *partial matches*.

Contents

1	Acknowledgement	4
2	Introduction	5
2.1	Goal	6
2.2	Problem Formulation	7
3	Background	8
3.1	Project	8
3.2	Named Entity Recognition	9
3.3	Corpus	9
3.3.1	Gathering a Corpus collection	9
3.4	Linear Classifier	10
3.4.1	LIBSVM or LIBLINEAR?	11
3.4.2	Solver Types	11
3.4.3	Support Vector Machine for Data Classification	12
3.4.4	Logistic Regression for Data Classification	12
4	Method	13
4.1	Overview	13
4.1.1	Existing Applications, Standards, and Formats	13
4.2	Tokenization	15
4.3	Annotation	15
4.3.1	Regular Expressions	15
4.3.2	Annotation with IOB2	16
4.3.3	Bootstrapping to Save Time	16
4.4	Finding Features	17
4.4.1	Part of Speech	19
4.4.2	Classifier using Regular Expressions	21
4.4.3	Lists of Common Tokens	21
4.5	A Classifier Window gives Context to a Token	22

5	Analysis	25
5.1	Detection of Named Entities in a Text Message	25
5.2	Finding Date, Telephone number, and Time	25
5.3	Finding Name and Location	26
5.3.1	Names	26
5.3.2	Locations	27
5.4	Useful Context	27
5.5	Corpus Collection	28
5.6	Text Issues	28
5.6.1	Informal Text	28
5.6.2	Language	29
5.7	Tools to use in Android	30
5.8	Post-processing	30
6	Evaluation	32
6.1	Comparison	32
6.2	Evaluation tools	32
6.2.1	Confusion Matrix	33
6.3	F-score	33
6.4	Testing	34
6.4.1	Cross Validation	35
6.5	Classifier vs. Regular Expressions	36
7	Result	38
7.1	LIBLINEAR's Solver Type	38
7.2	F-score on November test set	38
7.2.1	Cross Validation	40
7.3	Techniques	41
7.3.1	Evaluation of Features Sets	41
7.4	How well does other NER system perform?	42
8	Conclusion	43
8.1	Problems	43
8.2	Accomplishments	43
8.3	Future Work	44
9	Appendix	49
9.1	Android Environment	49
9.2	SNEEX	50

Chapter 1

Acknowledgement

We want to thank Pierre Nugues, our mentor at Lund University for invaluable guidance throughout our project. Thanks to him we have gone from being ignorant to feeling comfortable dealing with natural language processing techniques and especially *named entity recognition*.

We want to thank Håkan Jonsson, our mentor at Sony Ericsson for providing us with a Master's thesis, supplying us with an constant influx of ideas and directions to work with and setting up the work conditions and every arrangements to help us carry out our Master's thesis.

Also we would like to thank Christofer Bach and Johan Gunnarsson, our fellow students, for their contribution with an text message corpus to work with but also for their insight, advice, and help. Erik Pettersson for sharing his programming expertise when it was needed the most.

Chapter 2

Introduction

Cellular telephones are today constantly handling information, foremost either by sending or receiving it. But until recently that information have not been utilized by the cellular telephone itself or the applications within it. As cellular phones or actually smart phones increases in markets shares and computing power, they are becoming more commonplacred in our everyday life, there is a lot of new research and application opportunities emerging.

What if the cellular phones started to take advantage of the information that is generated to aide the users in everyday mundane task? One of such example with unused information would be text messages between users. If the application could semantically understand the text message, it could give the users the option to extract details from the message and forward it to another application that could make sense of it. For instance if the text message concerns scheduling a meeting, it might contains booking details. Those details can be extracted and sent to a calendar application that creates a event in the calendar automatically. Such convenience would be greatly appreciated by any user if it worked properly. To make such an application possible you need to detect those details in a text message and that is where *named entity recognition* can be helpful.

Sony Ericsson is interested in an application that could extract information from cellular text messages and if it could be devised. We were provided with a description of it. Sony Ericsson wanted, first of all, to see if there existed techniques good enough to handle information extraction inside a cellular phone. This included doing an overview of the problems in the area and suggesting possible solutions. A first step in such a research was to detect telephone numbers, dates, times, and names in text messages. It also included constructing a method on how to evaluate the result and to compare

it with existing standards. Finally it should be concluded how usable the examined techniques are, by ideally constructing an fully working prototype and suggest where further research and work are needed or which techniques could be applicable.

2.1 Goal

The goal of this Master's thesis was to develop an application that could run inside a cellular telephone that provide results with high accuracy in real-time. It also had to perform results that are in line with what could be expected from a mid or high-end computer.

The final application had to be a proof of concept that was able to recognize enough of the entities that provides information about a meeting. Then it had to prompt the user in a non intrusive way that there exists an opportunity to book the meeting from the conversation into the calendar. If the user would like to go ahead and book the meeting, then the application had to automatically supply the entries to the calendar with the meeting details it could extract from the text message.

Since a cellular telephone in today's standards is in all regards just a small but compact computer, we except to run into several limitations such as hardware issues like computing power, available memory, and how to conserve battery usage.

We also had to consider the non-hardware issues like data traffic, it might have been that we could reach high performance but only if it was linked with high transfer usage. The performance association with resource cost in the above mentioned challenges is a balance act. We needed to find a suitable middle-ground with respect to the standards of today's cellular phones even though these problems might have been alleviated with future technology advances they are nevertheless problems at the moment.

Besides the obvious commercial use for a company such as Sony Ericsson to have another killer application bundled to their cellular phones application armory, there was above that a scientific value. Part of the problem was to investigate if it was at all possible to achieve sufficient performance to make proper detection of meetings and if the smart phones have finally reach the capacity to be able to do an analysis in real-time using natural language processing techniques.

2.2 Problem Formulation

Is it possible to develop an application for cellular phones that can grasp enough context to detect if there has been meeting arrangements between two users? We have divided the research to the following sub areas, which we will investigate and discuss during this project.

- Detection and recognition of named entities in text messages
 - Finding relevant entities
 - Existing techniques to extract named entities
 - Appropriate techniques for us to utilize
 - Utilizing techniques
 - Limitations of existing techniques and how they be circumvented
- Evaluation of results
 - Evaluation of performance
 - Factors to measure
 - Comparison between our result and leading applications
- Corpus collection
 - Size of a sufficiently large corpus
 - Gathering a corpus of sufficient size
 - Hand annotation of a large corpus
 - Use of a semi-supervised method to go from a small hand-annotated corpus to a large automated-annotated corpus
- Structure of text messages
 - Structure of a one-on-one conversation
 - Adapting approaches to text messages

Chapter 3

Background

3.1 Project

It is worth mentioning that this Master's thesis started out as a project [8] for the course *Language Processing and Computational Linguistics* [20] at the department of Computer Science, Lund University.

The main purposes for that project was to create an initial framework for this thesis, which consisted of gathering a text message corpus or look at existing corpora as a viable alternative. We were to find *named entities* with regular expression, finding ways to evaluate performance. Further on, we were to compare against a state-of-the-art program and comparing different annotating techniques. Also we had to develop ways to visualize the extracted information and as well reflecting on the difficulties that arouse.

We learned that:

- A corpus that is suitable and sufficiently large is not easy to obtain. We found no available corpus that suited our purposes.
- Regular expressions have their limits and needs to be supplemented with lists of locations and names.
- A state-of-the-art comparison needs to be automated. Comparing against programs which hide their inner workings is impractical. Better to embrace evaluation that are used at conference challenges such as CoNLL [21].
- How to use F-score to measure performance in the *named entity recognition* domain.

3.2 Named Entity Recognition

Named entity recognition (NER) is the process of finding mentions of things that belongs to certain categories in a running text. In our case we are looking for following named entities:

Table 3.1: Examples of *named entity* categories.

Categories	Examples
Date	10-09-22 or 22/09/10
Time	12:34 or 9 pm
Telephone number	073-1234567 or +464612345
Name	Tobias or Torsten Andersson
Location	Lund or stationen

According to the article “A survey of named entity recognition and classification” [18] the impact of different textual genre and domains has been neglected in NER literature i.e. how well a NER system adapts to new domain. When testing a NER system on different domains there was a significant drop in performance on every system. The paper concluded that every domain can be reasonably supported but porting a system to a new domain remained a major challenge.

3.3 Corpus

A corpus (plural: corpora) is a large and structured set of texts in one or more languages. To be able to develop a software that recognizes named entities in text, we need plenty text for training and testing with. Ideally you want text that is similar to the text the system will have to deal with in run time, that means we want a huge amount of authentic text messages.

3.3.1 Gathering a Corpus collection

Our corpus mainly consist of text messages in Swedish but mixed with some English and German. It is mostly Swedish since the study is performed in Sweden and it is based on real text message with Swedish participants in an international setting.

We needed to gather a large corpus to have reliable result. A size about 100,000 words (i.e. *tokens*) was sufficient but more would give a more accurate result. [7] We instantly faced the problem with retrieving a sufficient amount of text messages since they are somehow considered by many users

to be private. The majority of the users was hesitating to sharing their text messages, if there was not an option to exclude certain private text messages. Strides were made to protect the users privacy and integrity, they also had to agree on a *terms and conditions* where they were informed that their contributions was used for certain services and research purposes. [14]

But even then, there was simply no compelling reason for the users to contribute with their text messages for solely research purposes. This is a common problem facing many corpus gathers and is the leading reason that there are only a few corpora around and even less that are annotated.

Researchers has been forced to buy corpora from others or in some manner gather one themselves which often prolong their work process. When annotation was lacking they had resorted to hire people to manually annotate by hand or device an open contests with prizes for any annotators who like to participate. Without a reward system it is hard to make people inclined to contribute with their text messages.

Our solution was to collaborate with two other Master's thesis students at Sony Ericsson whose Master's thesis [6] consisted of figuring out trends in text messages. The resulting application of that thesis was an Android application that displays current trends for the user. If the users want to use the application they would have to volunteer their own incoming and outgoing text messages which would in return be used for calculating future trends. We had full access to their corpus which was build up by 11 participants (students and employees from Sony Ericsson) and we were partly helped out with annotating it by Sony Ericsson.

3.4 Linear Classifier

A *linear classifier* tries to identify what class an object belongs to. First the classifier is trained with a corpus of tokens with known classes (i.e. gold annotated). The classifier calculates weights for each feature in correlation to each class. This can be seen as a probability of an object belonging to a certain class when having those specific characteristics. These weights are then saved in a probability matrix (i.e. *a model*), and will be used when the classifier is going to classify unseen tokens.

Unseen tokens lack the information of what class they belong to (i.e. it is not gold annotated). The classifier calculates a probability of a token belonging to each class. The class with the highest value is provided as the predicted class. Figure 3.1 visualizes linear separation for a problem with two classes.

For a two-class classification problem, one can visualize the

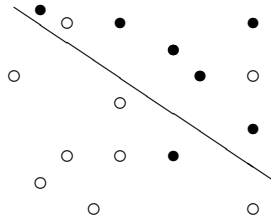


Figure 3.1: Linear separation

operation of a linear classifier as splitting a high-dimensional input space with a hyperplane: all points on one side of the hyperplane are classified as "yes", while the others are classified as "no". [5].

A linear classifier is considered to be fast when the feature set of each object is small. It handles multidimensional classifications well. Linear classifiers are usually divided into two categories of models, *generative* and *discriminative models*. We used the latter. Discriminative models includes both algorithms for logistic regression and for *support vector machine* (SVM).

3.4.1 LIBSVM or LIBLINEAR?

LIBSVM, released in 2000, has been a commonly used classifier for different areas of classification problem. Earthquake predictions, predictions within the medical area, business intelligence, and as well *named entity* classifications.

LIBLINEAR was released in 2007 and gained a lot of attention. It is a evolution of LIBSVM [12] with almost the same performance but is a factor of 100 times faster in certain cases [25]. We chose to use LIBLINEAR [9] since it uses a state of the art SVM. Other reasons was that it has a Java API and the Android software development kit is in Java. This made an integration possible. It is also open source so we could later on make adoptions if needed.

3.4.2 Solver Types

The LIBLINEAR package has six different solver types, two based on logistic regression and four based on different kinds of SVM's. SVM's are relatively new and tried out to solve a lot of old and new problems while logistic regression is traditional statistical tool.

3.4.3 Support Vector Machine for Data Classification

A *support vector machine* (SVM) is a popular technique for data classification.

“Each instance in the training set contains one target value (i.e. the class labels) and several attributes (i.e. the features or observed variables). The goal of SVM is to produce a model (based on the training data) which predicts the target values of the test data given only the test data attributes.” [12].

The SVM makes a matrix (i.e. a model) over the correlation between certain features and classes that occur together. Later on, that model can be used to predict the classes of tokens in a previously unseen corpus. Different features will be given different weights, as some token features gives a more certain prediction than others. The relation: *included in a list of cities* might be a greater give away than *preceded with to* as might also be applicable for times *17 to 19* or in a compositions of words like *to start with*.

3.4.4 Logistic Regression for Data Classification

Regression analysis is a field of mathematical statistics that is well explored and has been used for many years. The goal is to find a mathematical function that has the same characteristics as the data at hand. Given a set of observations, one can use regression analysis to find a model that best fits the observation data. This model is then used to predict future data using interpolation. Logistic regression compared to linear regression has a benefit of returning a value between zero and one, i.e a probability for a special case.[5].

Chapter 4

Method

4.1 Overview

The flowchart in (Figure 4.1) shows an overview of the major steps and components in our NER system and how it works on the cellular phone. It all starts with an SMS corpus which gets tokenized, each token gets its part-of-speech tag and regular expressions are used to catch the simpler annotations. This alleviate the following step that is to a manual annotation of the whole corpus, which should to be as correctly as possible.

After a conversion to a format that is suitable for the classifier, we feed that data into the classifier which returns a model on how it thinks classification should be done according to whatever rules (features) we instructed it with, which is usually called training the classifier.

The part-of-speech model used in the part-of-speech tagging goes through a similar process. Both these models are intended to be used later on for classifications.

The cellular telephone device have to do a similar process to prepare the text message for classification, it have to be tokenized, part-of-speech tagged, and prepared for the classifier since it wants to classify on pieces that looks likes the data it was training on. That is followed by some post-processing and finally the text messages is marked up with is proper named entities.

4.1.1 Existing Applications, Standards, and Formats

An important area in the background work is to investigate what research that has already been done in the area and what applications that uses partially or entirely the functionality that might help us. Whenever possible we have opted to use existing foundations such as standards, formats, tools,

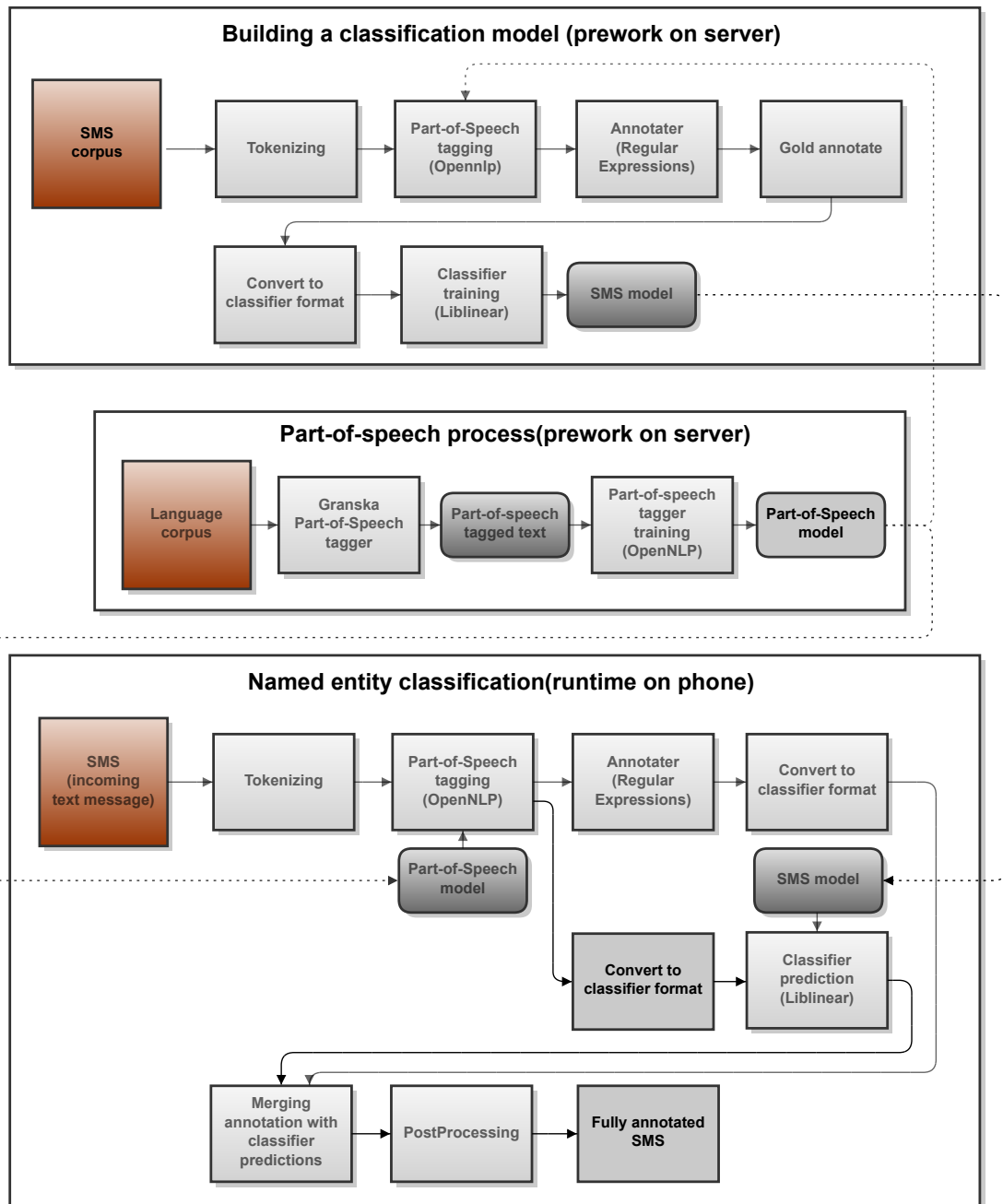


Figure 4.1: Overview of the major steps and components in our NER system.

and so on. It is preferable to build upon something that already exist to save time and energy that could be well spent on the main issue instead.

We decided to approach the *named entity recognition* (NER) problem by learning from other NER system developers. One of our most useful resources turned out to be CoNLL [21], specifically the *CoNLL 2003 shared task* where different teams was given the task of “Language-Independent Named Entity Recognition (II)”. In this task the participants had to find names of persons, organizations, locations, times, and quantities in given texts. Each word had been appended by the following *dynamic tags*, part of speech, syntactic chunk, and a *named entity tag*.

The common denominator by most teams participating in the CoNLL 2003 shared task was that they were;

- Using lists (of names or places etc)
- Taking advantage of the *dynamic tags* (such as part of speech 4.4.1)
- Taking advantage of the IOB2 annotation 4.3.2)
- Combining and linking different NER system together utilizing the strength of each system

4.2 Tokenization

The first step is to break up raw text into meaningful and manageable tokens. This is necessary because we want in subsequent task to mark up each token with at least two pieces of *dynamic tags*. A token is quite often a word delimited by space, but there exist many cases of when that is not the case.

How to split text into tokens is not trivial and straightforward as just splitting on whitespace and punctuation. Splitting on every punctuation will not do either since it breaks up ellipsis (e.g. [...]), common emoticon (e.g. :D), URLs and email addresses which could have some semantic value. Tokens consist either of one or two words, one or several punctuation characters, e.g. as an emoticon.

4.3 Annotation

4.3.1 Regular Expressions

A simple, yet powerful, way to find certain patterns in a text is to create *regular expressions* (regex) that can be used for string matching. A regex

like `.*borg` would match *Helsingborg* and *Vänersborg*. `(S|s)ofie` would match *Sofie* or *sofie*.

The greatest advantage of regexps to us is the number format matching. We can match common date format like 2010-06-12, times like 12:34 or telephone numbers which also follows a certain pattern of numbers with or without dashes and plus signs e.g. +46 073-12 34 567. Regexps may look a bit cryptic but is a very concise and precise way to match a pattern.

4.3.2 Annotation with IOB2

The text will then be divided into tokens and each token needs to be annotated with the correct tag. IOB2 format is dividing text into chunks of tokens that are beginning (B), inside (I) or outside (O) a *named entity*.

In our corpus most tokens, around 90%, are not considered to be named entities and are thereby denoted to belong to the class *O*. The tokens that are considered to be named entities belongs to one of the five *named entity* classes presented in Table 4.1. An example of the IOB2 annotation can be seen in Table 4.2

Table 4.1: Named entities with corresponding IOB2 format.

Named Entity	Example
DAT	Date
TIM	Time
PHO	Telephone number
PER	Name
LOC	Location

4.3.3 Bootstrapping to Save Time

Once the the corpus is tokenized, every token needs to get annotated with a correct IOB2 tag. Annotation is a time consuming and a costly process that requires a good deal of human effort. Ideally you would want a supervised process like a small team to annotate the same part of the corpus so that high accuracy and quality can be achieved. Annotators are not perfect, their skill to correctly label entities varies and they will sometimes have different opinions of how to annotate so a consensus have to be reached for each entity.

Even with such safeguards the annotation will not often be able to reach 100% accuracy, instead, it will linger closer to 97% [19]. We do not have a

Table 4.2: IOB2 annotation example.

Token	IOB2 tag
Can	O
we	O
meet	O
at	O
Grand	B-LOC
Hotel	I-LOC
at	O
17.30	B-TIM
?	O

annotator team, so we looked at different methods to bootstrap the annotation.

We took a smaller part of the corpus that we ran our own regular expressions to detect and label entities as an initial step. Regular expressions are great for detecting entities which follow certain patterns for instance dates which follow certain formats. But the regular expressions are insufficient for entities that differentiate too much from these simple patterns and it is impractical to device regular expressions that could match everything.

The following step was to correct the entity labels by hand. This is something we want to avoid as stated earlier, but it serves a purpose. We do not want a supervised process as it is time consuming and unsupervised is not clearly going to work if it is only regular expressions that annotates. But a semi-supervised process with regular expressions and hand annotation on a smaller part of corpus can be part of the *dynamic tags* that will be fed to a machine-learning tool such as a classifier. This machine-learning tool will then hopefully be able to predict the complex patterns needed to determine named entities with an acceptable accuracy.

4.4 Finding Features

A *feature* is, in this context, a characteristic of a token. All features of a token might be possible clues to the system of whether or not this token is a *named entity*.

Description of some of our features:

Part of speech: Lets the classifier draw its own conclusion how POS is related to current token.

Preposition before current token: If the preceding token is a preposition like “at” could indicate that current token is a location.

Data derived from training set: More about them in Dynamic lists [4.4.3](#).

Contains features: Most of these are based on regular expressions (digit, times, phones, letter)

Prefix and suffix: In Swedish many locations (towns, squares, parks) ends with certain suffixes.

Capitalization: Usually a strong indicator of locations and names, not so common in text messages though.

Preposition, verb, noun: Lets the classifier draw its own conclusion of how a specific POS is related to current token. Intuitively, these POS should be covered by the POS feature but this feature brings more weight to these specific POS.

Only letters or digits: Helps separate times and telephone numbers from the others.

Word length: Entities are more likely be in certain length, for instance overall locations has longer word length than names.

End of sentence E.g. names are often at the beginning or in the end of a sentence like in “Hi Alex”, “cya later /Bart”.

All our features are binary with the Boolean attribute true or false. If a specific token fulfills the characteristics of a feature, the feature is set to true for this token; otherwise false. These features will then indicate which category a particular token could belong to. If the token consists of digits, it could mean that the token belongs to either the telephone or time category but could just as well belong to neither.

Unless you are only looking at cases that are simple classifications this Boolean indicator alone would end up with ambiguous and uncertain results. So you need several features to distinguish what category a token belongs to and the classifier have to be trained with data where the correct category has been annotated.

As every feature is binary, we need to represent our features as such. That means a feature as the *POS* feature has a feature representing every POS class (about 110 in total) or the *which common word is it?* feature which takes more than 2,000 words in consideration has a feature representing each and every one of those words, but in contrast the *is it a common word?* feature only needs one feature.

So as far as our classifier is concerned, our system contains around 43,000 such features. But as far we are concerned there are only slightly more than 30 features. Most of the features we ended up using are showed in Table 4.3.

Most of these features are self-explanatory. Some look at the token at hand (digits, letters, prefixes, suffixes). Some look at the part-of-speech tag associated with the token or previous tokens. The *contain features* uses either regular expressions or lists. A few features tries to bring in some context and looks at previous words or if the token is at the start or end of a sentence.

Finding features that might be relevant and how to express them is more or less a never ending work in larger systems. Essentially you come up with some features by looking at what features other NER system uses and then you use your own knowledge about your data to create new ones. When having a sufficient amount of features you are making a lot of experiments with different combinations of features to finally conclude if that particular feature set is improving performance or not. You have to follow up the development with some regression tests to validate yours or others conjectures and see if they do indeed improve performance for your system. We only used those features that we saw had a positive impact on the F-score.

4.4.1 Part of Speech

The different characteristics of tokens might give us clues of which *named entity* the token belongs to. The *Part of speech* (POS) is one of those characteristic. POS are noun, verb, adverb, adjective, preposition, pronoun, conjunction, interjection, and article. Things that we all learn early in our school years but never really knew what they where good for. The *Natural Language Processing* field has found a real use for them and have distinguished them further to about 50-150 POS that are commonly used.

We use the POS as a tell-tale if a word is a entity and which category it then could belong to.

“Each part of speech explains not what the word is, but how the word is used. In fact, the same word can be a noun in one sentence and a verb or adjective in the next.” [4].

A word could be a location if it is a noun but it is seldom a location if it has any other kind of POS, e.g. a verb. Another tell-tale can be the POS of the words around the particular word that we are currently looking at. One of those patterns could be that the previous token POS tag is preposition e.g. *inside Starbucks*.

To simplify, if we detect a pattern, let's say a group of three words that has the following POS; verb, preposition, and a noun. They occur very frequently together and the word that has POS noun is annotated as a location. Then the classifier can deduce or make an educated guess when it encounters a similar example that the word with the noun, is a location.

Part of Speech Tagging

A POS tagger takes a sequence of words and tries to assign each word its proper POS tag according to the context of the word. We took a small part of the Stockholm Umeå Corpus [23], which consists of 100,000 Swedish words. We ran that corpus through the Swedish POS tagger Granska [15]. Table 4.4 shows an example of output from Granska.

The outcome of the Granska's tagging was used to finally train a different POS-tagger called OpenNLP [2]. OpenNLP creates a model which can be used later on for POS tagging within the cellular phone. The reason for this inconsistency is that Granska can not operate within Android and OpenNLP have to be trained to be able to tag in Swedish, since it lacks a model for Swedish. Linking taggers in this manner brings unwanted side effects, using the output from one tagger to another causes degeneration, which gets worse on each iteration and restricting the size of training corpus for the POS tagger means it performs suboptimal.

Another issue is that the size of the trained model has to be reasonable small to keep the memory usage low for the Android application, since the default implementation on phones is that no application may allocate more than 16 MB memory. With OpenNLP, we can dictate the model size to our liking with using a small sized training data. Unfortunately the backside is that the tagger will be less general and provide less accurate results.

We could have a better equipped POS tagger for training models and to use in the evaluation, since these parts were done on a desktop computer, but we wanted real results that would correspond with the performance within the cellular phone. We also wanted to have the POS tagging in the training part and the prediction part to be in agreement on how to tag since any discrepancies would actually worsen performance. So we made sure that we ran OpenNLP on the training as well on the testing part so the classifier trained with the same "substandard" tagging to not risk escalate any error

rate further.

4.4.2 Classifier using Regular Expressions

The classifier uses regexps to extract features. Such regexps covers simple cases of dates, times, telephone numbers, and suffixes. A second reason for having regexps for the classifier was to make the classifier be able to stand alone so the annotation with regular expressions could be removed or detached if necessary later on.

4.4.3 Lists of Common Tokens

Static Lists

When looking for names and locations in text, it might increase performance having static lists of the most common names and locations. We compiled lists to use in feature extraction as well as for regular expression matching. Lists are easily collected from the Internet but they are not always that suitable for the purpose.

We populated our own lists in some cases and other times we gathered them from various sources such as Statistiska centralbyrån (SCB) [22]. From this list we pulled first names, last names, and Swedish locations. If you put in some effort, you can create a web crawler to collect useful data, for instance from Wikipedia [5] and have huge lists in no time. We did not. There is an increased risk of false positive hits with larger lists. Selective smaller lists with relevant content should in most cases be a wiser choice. Table 4.5 shows our static lists.

Comprehensive lists that cover most global locations, would make this problem even worse. A workaround could be to keep the list to that specific region the user is located at or look at the *dynamic tags* that is connected to the word suspected to be a location, *dynamic tags* such as POS for instance if the word “*bara*” (only) is an adverb or a city name. If it is not capitalized then it is not likely to be a location. If you just try to match lists against a specific word you will end up with noise and ambiguity.

Dynamic Lists

In addition to the static lists of locations and names, we also used dynamic lists that were derived automatically from the training data. A frequent word list that contains words that occurred more than, e.g. two times in the training corpus.

To be able to use information as common word patterns, we use lists of extracted unigrams, bigrams, and trigrams. Also lists of words and part of speech that precedes a *named entity* are used. During development an individual cutoff frequency has been determined to each list. With an aim to keep the lists as small as possible without infringing on performance. The dynamic lists are only used by the classifier.

4.5 A Classifier Window gives Context to a Token

Since context is often critical to predict which category a token belongs to, we are in some cases interested in the preceding and succeeding tokens of a token. We include the feature data from the surrounding tokens for the current token. In the example below where the token W_n is to be classified and the two previous part of speech tags and the current token are deemed important for a feature.

Tokens:	W_{n-3}	W_{n-2}	W_{n-1}	W_n	W_{n+1}	W_{n+2}	W_{n+3}
POS tags:	T_{n-3}	T_{n-2}	T_{n-1}	T_n	T_{n+1}	T_{n+2}	T_{n+3}

We are looking at the most at three tokens preceding and succeeding with their corresponding tags for a total window of seven tokens on those features where it makes sense.

If you expand the window too far out it starts to introduce noise and will decrease overall performance. We found that a window of maximum seven was ideal for our purposes. For most features, a smaller window size was better, typically the classifier worked best when looking at the current token plus one before or after.

Table 4.3: The major features in use by our NER system.

Feature	Example
Part of Speech tags	noun, verb
Contains common word derived from training set	
Which common word it is	with, car, orange
Contains a location suffix	-harbor, park
Contains a digit	12:34, 1234
Contains a denominator such as	,,;:/ ()-
Contains common names	Maria
Contains names derived from training set	
Contains locations	New York
Contains locations derived from training set	
Contains dates	august, Monday
Contains times	12:00, 10 pm
Contains telephone numbers	046-123456
Only letters	wordwithoutdigits
Only digits	1234567890
Is all in uppercase	SEMC, XML, NLP
Initial letter is uppercase	Summer
Initial letter is lowercase	summer
Preposition	to, from, by
Preposition + token + end of sentence	'to Lund.', 'by him?'
Verb	drive, see
Verb using a word list of small selection of verbs	coming, eat, driving
Noun	phone, train, book
Is end of sentence	!..?
Length of the word	
Useful unigrams	at, regards, hi
Useful bigrams	'welcome to'
Useful trigrams	'wanna meet at'
Useful part-of-speech unigrams	preposition, noun

Table 4.4: Output example from Granska.

Detta är en magisteruppsats i datavetenskap.
 ‘This is a Master’s thesis in computer science.’

Detta	<pn.neu.sin.def.sub/obj>
är	<vb.prs.akt.kop>
en	<dt.utr.sin.ind>
magisteruppsats	<nn.utr.sin.ind.nom>
i	<pp>
datavetenskap	<nn.utr.sin.ind.nom>
.	<mad>

Table 4.5: Static lists (from various sources) .

Content	List size	Example
First names	200	Johanna, Martin
Last names	100	Björklund, Lundqvist
Relation names	14	Faster, Mamma
Swedish towns	2,000	Lund, Stockholm
International cities	150	New York, Tokyo
Point-of-interest	15	Dagis, Systemet
Months	32	July, Dec
Words related to days	60	Valborg, Imorgon

Chapter 5

Analysis

In this chapter we try to analyze our methods, draw conclusions, and highlight our problems during the different phases of our work.

5.1 Detection of Named Entities in a Text Message

Sony Ericsson is interested in finding several different types of named entities in text messages: telephone numbers, dates, time expressions, names, and locations. A first division of entities is done by grouping together date, time, and telephone number in one group having in mind that they mostly consists of numbers and might respond to the same techniques. The other group is locations and names, both of them consists of letters and use similar matching techniques.

We might add that when annotating we were quite strict. A token could be misspelled or compounded in such a way the it would be missed by any NER system (e.g. “12januri”) but to the annotator it is still clear what the users intention was, so it gets annotated. The effect will be that the classifier gets better data to build the model on, but if such an example would only be seen in the test set it would lower the F-score.

5.2 Finding Date, Telephone number, and Time

It works quite well finding telephone numbers, dates, and time expressions, with regular expressions. At least it works well as long as the entities are more or less well formed. When users are using their non standard ways of expressing these entities it gets difficult. The regular expressions quickly

reaches their limit, it is possible to continue developing them but at a point any improvement comes with increasing complexity and manageability.

Date: When users expresses dates like “I’m having vacation week 30-31” or “Can we meet the 7th?”.

Telephone number: Weird number formats like “++49 (0)9445 111” or in the text message “Call 222”. 222 in this case is a short number to reach the cellular telephone’s answering machine.

Time: When users express time as; *930* or *1234*, then it is as likely to be an amount, number of a street or a code as it would be a time expression.

5.3 Finding Name and Location

For both names and locations the strategy was to first match the tokens against lists of known locations and names and to have something to start out with. For instance, features that checked for tokens that started out with capital letters are used and also features that takes in information about the surrounding tokens.

The likelihood of a token being a location or name increases if the last or next token is one. For both classes a good clue is to look at the part of speech tag. Locations and names are mostly having the *nouns* or *proper name* tags. There lies an ambiguity with names and locations that is not easily solved, point of interest locations, such as restaurants or coffee shops, are often named after a person.

5.3.1 Names

Using lists to match names works quite well but the system that depends on regular expressions only matches against the token and not the context. The hard case here is that some words are ambiguous so the system will undoubtedly make some mistakes. Examples are given below:

Name	Other meaning
Per	“1,20 kr <i>per</i> minute”.
Inga	None/zero
Bo	Staying/living at.
Max	Short for <i>maximum</i>

To the system based on regexps it is also hard to detect nicknames as “darling”, “sweetie” etc., and names not included in the list will mostly be missed.

The classifier system will not have the same problem even though it utilizes the same lists. Even though it finds a strong correlation between the lists and what is actually a name it does not rely exclusively on one feature. Instead it tries to use several features and therefore has a context to base its decision on.

5.3.2 Locations

Locations are a bit tricky, or in fact, might be the hardest one. Some locations are straightforward to detect, as correctly spelled city names. Those we can detect with a list of common city names and regular expressions. The hardest part is to detect the *I-LOC*, meaning inside locations like *York City* in “New York City”.

*“Tieto welcomes you to Android Seminar tomorrow Wednesday
May 5. Ideon, **Scheelevägen 17, Lund**. Registration and
breakfast from 08.30.”*

Here *Ideon* should be classified as B-LOC and the bold marked tokens should be denoted as I-LOC. Especially punctuation tokens as “,” are hard to catch both when using regular expressions or when depending on a classifier technique. The streets numbers are fairly hard to find as well. It is generally quite difficult to find common characteristics of tokens being inside a named location entity. These are the reasons why this *named entity* class is also the one that gives us the lowest F-score.

5.4 Useful Context

As we had an unsupervised regular expression annotation preceding the classifier, we had one extra layer of information besides the word and it is the part of speech tag. The extra layers of information, POS and IOB2 tags, are visualized in Table 5.1. We thought that we could take advantage of this knowledge by looking at the on current, preceding, and succeeding IOB2 annotation (the window stretching from C_{n-2} to C_{n+2}). Thinking that it might be easier to classify the current token if there was something to go on, instead of a blank slate of IOB2 information. As it happened it did not, the classifier found such strong indications from the IOB2 annotation that it

Table 5.1: Context window for a token

Tokens:	W_{n-3}	W_{n-2}	W_{n-1}	W_n	W_{n+1}	W_{n+2}	W_{n+3}
POS Tags:	T_{n-3}	T_{n-2}	T_{n-1}	T_n	T_{n+1}	T_{n+2}	T_{n+3}
IOB2 Tags:	C_{n-3}	C_{n-2}	C_{n-1}	C_n	C_{n+1}	C_{n+2}	C_{n+3}

shut down all its own attempts to find its own annotation and was satisfied with just correcting mislabeled IOB2 annotations.

That triggered a separation of the annotation with regular expressions and the classifier and lead to our merging idea that is covered in the Classifier vs. Regular Expressions Section 6.5.

We still feel that perhaps there is some way for the classifier to utilize this information but we did not pursue it any further.

5.5 Corpus Collection

Corpus collection is something that has to be planned for in advance. The actual work cannot start until you have a corpus to develop against. In some categories of corpus, as formal text, gold annotated corpora already exists and are readily available. In some other categories, there is none. It is time consuming to gold annotate a corpus and bootstrapping methods should be taken into consideration.

5.6 Text Issues

5.6.1 Informal Text

Formal text is text that is intended to be read by many readers. Typically published in a book, on a website or in a newspaper. The advantage with working with such a text is in the aspect of correctness and thereby predictability. Corpora like the SUC [23] or Penn Treebank [3] corpus are published texts and show a high correctness in spelling, grammar, and use of standardized language.

Informal text, as cellular phones text messages, offers a greater challenge. Text messages often include slang, a special set of abbreviations like “CU” (see you) . It frequently includes emoticons that tries to reduce the inher-

ent ambiguity in text message so the intentions of the sender is properly conveyed or just simply to express emotions. For instance the difference between *you are stupid* and *you are stupid ;-)* are to most people nowadays obvious. Capital letters are something most users are not bothered with since it involves extra keystrokes on the cellular phone. Another problem is that a text messages quite often are fragments of a real sentences.

It is also important to point out that text messages are *conversation-based* which is a big difference from formal text. The context is shared by the two users and it gives less demands on complete descriptions. Even though text messages often show mistakes like the ones shown above, it is mostly obvious to the receiver what the message is about. People tend to accept this “incorrect” language having the other advantages in mind, like speed and convenience.

A text message can be seen as a part of a conversation that is performed on multiple medias, through live conversation, telephone calls, email, IM (instant messaging), and through cellular telephone text messages. This means that a text message might be a fragment of the full story and that pieces of information are left out. “Ok, See you there.” Might mean “I accept to see you at the buss stop Bankgatan at 8.30 as I suggested in my last text message.”.

So, in general usage, the text issues are not a problem to users. But in our case, when trying to extract information, this can be a real obstacle. We can not rely on rules like that a location starts with a capital letter like most other NER system. [17]

A POS tagger is more likely to give a erroneous interpretation of words if the grammar or spelling is incorrect. It is almost impossible to create rules that catch malformed date or time occurrences. The conclusion of all this is that we will expect a lower performance (F-score) than the current state of the art.

5.6.2 Language

Our system had as one of it is goal to take care of both Swedish and English text messages and even though POS classification scheme does not differ much between the two languages, there are differences such as which class a word belongs to. So the POS taggers are not language independent which forces us to match each language to a specific POS tagger or choose a POS tagger that is open to be trained for several languages. Either way this requires a mechanism that can detect and classify what language a text belongs to.

To decide what language a text message is in, we intended to use a straight

forward technique that involves a list of the 1,000 most common words in each language and will give the text message a language score depending on how many words on each list that was encountered. The language that got the highest score, will then be used as a determiner of which subsystem that should be used to extract information. Using the English system on a text message written in English will theoretically reach a higher F-score since it will take into consideration English formats of date, time, etc.

5.7 Tools to use in Android

We evaluated several different learning classifier systems for POS tagging. We had quite strict requirements to fulfill. It had to be written in Java so that it could be used in Android, preferably open-source if we would have unexpected issues later on and could not use more than 16 MB memory in runtime. Finally it could be trained to work for most languages. We were almost forced to write a simplified POS tagger on our own but eventually out of 15-20 candidates we found OpenNLP. To our knowledge there exists no software in this category that has been designed to be used with cell phones in mind.

5.8 Post-processing

After the classifier was done, we did some after work on the classification. We checked and corrected if there were any obvious errors as an *inside* tag without a preceding *begin* tag or two adjacent *inside* of different entities. We also noticed some patterns, like when there was two name entities in a sequence both being *Begin*, e.g. *B-PER B-PER*, they were likely to be first name and last name and therefore the first entity should be set as *B-PER* and the preceding as *I-PER*. The rules for this is shown in Table 5.2. Sometimes other NE labels were mixed in the mentioned way, Table 5.3 shows guidelines of how to deal with them. Bear in mind that, not all of these rules was in place or was applied for every NE, these are just examples.

During development when our classifier did not perform at its best, this increased the performance. Against some test set, the error checking and correcting rules helped on other test set it even lowered performance significant. We ended up not using post-processing on cross validation or on the application prototype.

Table 5.2: Post-processing of IOB2 tags of same NE, e.g. LOC.

Original		Changes	
O	O		
O	B-LOC		
O	I-LOC	O	B-LOC
B-LOC	O		
B-LOC	B-LOC	B-LOC	I-LOC
B-LOC	I-LOC		
I-LOC	O		
I-LOC	B	B-LOC	I-LOC
I-LOC	I-LOC	B-LOC	I-LOC

Table 5.3: Post-processing examples of IOB2 tags with different NE.

Original		Changes	
B-LOC	B-PER		
B-DAT	I-PHO	B-DAT	B-PHO
I-PER	B-TIM		
I-TIM	I-DAT	I-TIM	B-DAT

Chapter 6

Evaluation

6.1 Comparison

An important section of this report has to be the way we compare the performance of our techniques and thereby our application with the ones that exists on the market. As this area is new on the cellular telephone market, we have to compare with larger system which are run on desktop computers. It should be remarked that this comparison might be unfair in the sense that a computer does not have the same limitations such as a small cellular phone's processing power, its restricted use of memory or limited response time.

In our opinion a result as good as the ones currently available on the market, for desktop computers, would be an absolutely satisfying result.

We will have to develop our own test program that can evaluate different measurements. We need to compare the result as well as comparing them for specific areas of difficulties. The latter we think is essential. Since it gives guidelines during development of where more work is needed and also it shows the strengths of our system.

6.2 Evaluation tools

We evaluated the standard metric in the field of our application, using F-score as it is of the more prominent standards. Over time we found a need of more detailed information about our programs strengths and weaknesses. We then calculate F-score for each IOB-tag (B-LOC, I-LOC etc.), for each *named entity* class, and also how many actual correct classifications we did. We printed all the missed entities to be able to find patterns that we could explore further. When developing we were helped by this nuanced feedback by the system.

Table 6.1: Sample confusion matrix showing partial matches

		P R E D I C T					
		TIM	PHO	DAT	LOC	PER	OUT
G	TIM	23	0	0	0	0	42
	PHO	3	6	0	0	0	6
O	DAT	8	0	6	2	0	72
L	LOC	2	0	2	6	0	107
D	PER	0	0	0	0	10	69
	OUT	13	3	5	11	0	2599

6.2.1 Confusion Matrix

First of all we calculate a *confusion matrix* showed in Table 6.1 to see how the result are allocated. The first column shows what class the token should have been classified to and the first row shows what the token was predicted to be, with our system. The confusion matrix gives a good insight of any weak spots in the system. E.g. it can show if there is any entity type that is misinterpreted for another entity type. The numbers in gray cells are tokens that are correctly tagged, i.e. they are having the same correct tag as they has been predicted as. It is also obvious that most tokens are not named entities, by having 2,599 tokens being correctly tagged as OUT (i.e. not a *named entity*).

6.3 F-score

In statistics, the F-score (also called F-measure) is a measure of accuracy. To evaluate our results we use the F-score because it is used by CoNLL [21] and thereby became a standard within this field of research. We saw that this method is still the common way to deal with the evaluation of systems. To calculate an F-score one first need to compute the precision and recall, we need these definition to start with:

- True Positives (TP), correctly predicted, is an entity.
- True Negatives (TN), correctly unpredicted, is **not** an entity.
- False Positive (FP), incorrectly predicted, is **not** an entity.
- False Negative (FN), incorrectly unpredicted, is an entity.

$$Precision = \frac{TP}{TP + FP} \quad (6.1)$$

$$Recall = \frac{TP}{TP + FN} \quad (6.2)$$

$$F\text{-score} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \quad (6.3)$$

Precision answers the question: *How good are our findings?*. Recall answers the question: *How many of the occurrences did we find?*. The F-score is the harmonic mean of precision and recall. The F-score that is generated, spans in the range of 0 to 1 where 0 is worst and 1 implies 100% correctness.

When developing our system we made a conscious choice to put emphasis on precision instead of recall. It would be possible to reach the same F-score but having the balance the other way around. A high precision mean that we prefer to miss some entities sometimes but we are quite sure that the ones we do present are correct.

6.4 Testing

To perform a evaluation the corpus is divided into two pieces, training set, and testing set. The training set is based on the gold corpus while the test set has been annotated solely by regular expressions i.e. devoid of gold annotation. It is a representative of text used as input in a real system. The training set is used to train the model while the test set is used for evaluation and error analysis. Most of the corpus should go to the training set, 80-90% is typically used.

The test set could then be further divided into a developer set and a test set. The error analysis and feature development is performed on the development set. The test set is not actually looked at and is only used as a final evaluation or to give performance hints during the development. The reason behind this is that you will otherwise probably, consciously or unconsciously, be swayed to adapt your features to the test set, but a model based on these features won't generalize particular well to any new data. This issue is called over fitting, which can be avoided by using cross-validation.

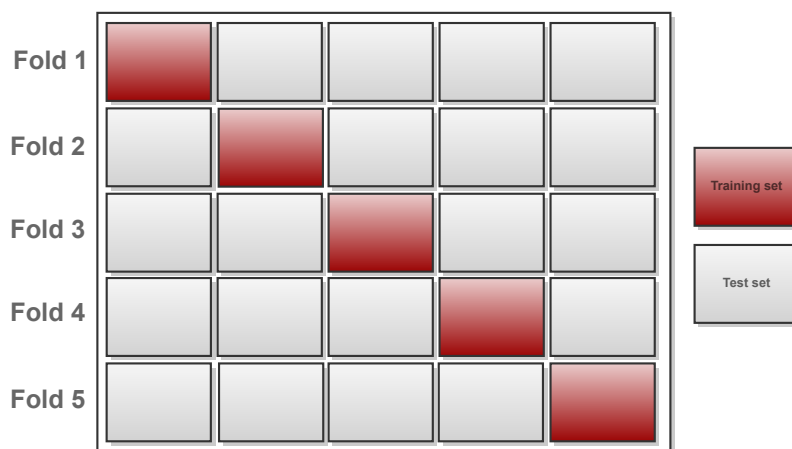


Figure 6.1: Example of a 5-fold cross validation, red is training set, gray is test set for each row.

6.4.1 Cross Validation

A cross validation is a technique that is used to estimate the results of how a system that is predicting data will perform during real conditions [16]. In one adaption of cross-validation you slice the corpus into N pieces, on which you perform multiple evaluation. Each piece gets to acts as the test set while the other $N - 1$ pieces takes the role as the training set. See Figure 6.4.1 for example.

We decided on using a test set with 2,000 tokens (214 text messages) for feature development and error analysis which Sony Ericsson gathered during our project and was composed by a unknown but very few participants. This test set will be referred as *November test set*. The training set has 60,000 tokens (4,446 text messages) and was composed by 11 participants. We did not use a development test set and ran into the risk of having results that would be inaccurate and over-optimistic. We, by this reason, performed a cross-validation on the training set, a 5-fold (1/5 test, 4/5 training) where we did a splitting on the text messages to keep as much context as possible intact. Which basically means we had two different ways of testing performance.

6.5 Classifier vs. Regular Expressions

With a F-score close to 74% (counting named entities only) we reached a threshold. We realized that the classifier trusted the regular expressions (regexp) too much. The classifier depended so much on the regexps that it did not add any more rules based on the correlations it had extracted from the training corpus. At that time we compared the results from the classifier and the regexps and saw that even though the F-score were rather similar, there were differences.

Table 6.2: Classifier vs. regular expressions, star (*) show which system that performed best at that particular IOB2 tag and the difference shows with how much.

NE	Classifier	Regexp	Difference
B-TIM		*	2 %
I-TIM		*	36 %
B-PHO		*	5 %
I-PHO		*	15 %
B-DAT		*	9 %
I-DAT		*	14 %
B-LOC		*	4 %
I-LOC	*		30 %
B-PER	*		3 %
I-PER	*		71 %

The classifier and the regexp found a great amount of matches that were the same, but showed clearly that their strengths were on different areas. To use their different areas of expertise we decided to separate the systems into two, one based on the classifier and the other on the regexp. Then we compared their classifications of each token as can be seen in Table 6.2.

We developed an algorithm for which system to trust, based on trial and error, that gave the best performance that can be seen in Table 6.3

We were successful with improving the results with this method. Their separate and combined performance can be seen in Table 6.4

Table 6.3: The algorithm

- If neither of the systems tagged the token as a NE, it was tagged as Out.
- If only one system tagged the token as a NE, that tag was chosen.
- If the classifier tagged the token as I-PER or B-PER, that tag was chosen.
- If both systems tagged the token, and it was not tagged as PER, the regexp annotation was chosen.

Table 6.4: F-score results for classifier, regexp and their combined end result.

System	F-score
Regexp	76,76 %
Classifier	77.73%
Combined	84.19%

Chapter 7

Result

7.1 LIBLINEAR’s Solver Type

During our development process, we examined all of LIBLINEAR’s solver types. We concluded that in our system the solver type that uses logistic regression gives a little better result than the ones based on a *support vector machine*. Even though time complexity of SVM is $O(N^2)$ and with logistic regression $O(N^3)$, this is not an issue for us since the time consuming phase occurs when training the model. The model is trained on a desktop computer and is then transferred to the cellular telephone. The F-score for the different techniques can be seen in Table 7.1.

7.2 F-score on November test set

We calculated the F-score at several steps to be able to get a fuller picture of where the strengths and weaknesses of our system lies. Since the annotation is sparse which means there is a overwhelming number of the O (outside NE) class. The category *O* is excluded from most tables and the F-score calculations, since it is not a *named entity* like the other tags.

In this section we present how well our system performed on our *November*

Table 7.1: Solver types of LIBLINEAR

Technique	Strict matches	Partial matches
Logistic regression	77.04 %	84.52 %
Support vector machine	74.15 %	81.61 %

Table 7.2: Confusion matrix showing strict matches on November test set.

		P R E D I C T E D										
		B-TIM	I-TIM	B-PHO	I-PHO	B-DAT	I-DAT	B-LOC	I-LOC	B-PER	I-PER	OUT
G O L D	B-TIM	41	1	0	0	0	0	0	0	0	0	1
	I-TIM	1	17	0	0	0	0	1	0	0	0	3
	B-PHO	0	0	11	0	0	0	0	0	0	0	0
	I-PHO	0	0	0	4	0	0	0	0	0	0	0
	B-DAT	1	0	0	0	41	7	0	0	0	0	0
	I-DAT	0	0	0	0	0	39	0	0	0	0	0
	B-LOC	0	0	0	0	0	1	56	1	0	0	22
	I-LOC	1	0	0	0	0	0	3	12	0	1	20
	B-PER	0	0	0	0	0	0	0	0	52	0	12
	I-PER	0	0	0	0	0	0	0	0	0	12	3
	OUT	1	0	1	0	3	0	1	1	2	1	2623

test set. First a **confusion matrix** in Table 7.2, showing all predictions and miss predictions.

Each NE class has its own F-score calculated and then every class is further separated into *Begin* and *Inside* according to the IOB2 standard. These sub classes also gets their F-score calculated.

Strict F-score means that a predict is only accepted if the prediction is in the right sub NE category as well. It differ between *Begin* and *Inside* which can be seen in Table 7.3. As can be seen, the hardest *named entity* to detect was *Location*, especially a token that is inside a location phrase. E.g. “Grand **Hotel**”.

Table 7.3: Strict matches on all labels by November test set.

Tag	False pos	False neg	True pos	F-score
B-TIM	4	2	41	93.18
I-TIM	1	5	17	85
B-PHO	1	0	11	95.65
I-PHO	0	0	4	100
B-DAT	3	8	41	88.17
I-DAT	8	0	39	90.7
B-LOC	5	24	56	79.43
I-LOC	2	25	12	47.06
B-PER	2	12	52	88.14
I-PER	2	3	12	82.76
F-score				84.19
O	61	10	2623	98.66

Partial F-score means that a predict is accepted if the prediction is in the right NE category. It does *not* differ between *Begin* and *Inside* within an entity class (such as B-LOC and I-LOC). In this case, a last name that

is predicted as a first name is correct since it is in fact a name. In Table 7.4 partial matches are calculated.

Table 7.4: Partial matches on main labels by November test set.

Tag	False pos	False neg	True pos	F-score
TIM	3	5	60	93.75
PHO	1	0	15	96.77
DAT	4	1	87	97.21
LOC	3	45	72	75
PER	1	15	64	88.89
F-score				88.43

7.2.1 Cross Validation

Table 7.5: Cross validation results. Since the corpus was split on text messages, the amount of tokens differs between the five test sets.

Tokens in set	Strict matches	Partial matches
11754	85,94	89,41
10250	87,31	90,02
8413	84,23	87,27
7057	86,77	89,20
6427	86,12	89,50
	86,09	89,12

If you compare the result in the cross validation Table 7.5 and the *November test set* Table 7.3 and Table 7.4, you will notice that *November test set* has a slightly lower F-score then the cross-validation which contradicts what has been previously said, that the cross validation score is usually lower than the F-score. Our reasoning for the better F-score on cross-validation is that the *November test set* differs from the training set, the *November test set* is homogeneous since it is based on very few participants while the training set has more of a heterogeneous nature. In Table 7.5 can be seen that: **Strict matches** reaches an F-score of 86%. **Partial matches** has a bit higher result of: 89%.

7.3 Techniques

Here we present what performance that was reached by using different techniques that we studied and tried out.

7.3.1 Evaluation of Features Sets

We have been developing and trying out many different feature sets and will here provide you with an overview of how good each approach is by itself.

Table 7.6: F-score of different feature sets (all results are expressed in percent).

Tag	POS	Regexp	List	Other	All
TIM	43	67	43	22	87
PHO	43	64	67	72	87
DAT	31	72	50	0	94
LOC	17	57	48	3	72
PER	22	65	81	0	87
Mean	28	65	56	10	85

POS Tagger

When using only features based on the POS tags of each token and other tokens within the context window the F-score for the named entities are shown in Table 7.6. It can be seen that the POS features gives well distributed result but a system relying on POS tags only is not that impressive.

Regular Expressions

Regexp are straight forward and performs quite good even when being without support from other kinds of features. They are especially good with numbers. The classifier responds well on the clues given by the regexps and they are easy to combine with other sets of features.

Lists

Having lists of names and locations obviously gives a boost to locations and names. But also clever lists like unigram, bigram, and trigram lists tweaks the performance. If using n-gram lists, it is important to use some *cut off* value to keep the system general. One does not want the system to “overact” on occurrences it only saw a few times.

Other Features

Other features gives a boost to a already working system. They might increase the result a bit. But this category is the one where one has to be extra careful to not produce features that will be over fit to a corpus.

7.4 How well does other NER system perform?

An important factor in evaluating our system is obviously to be able to compare it with other systems that are doing similar information extraction. Even though we did rigorous testing against our own data, its helpful to know how we perform compared to other NER systems. So we gathered figures given in various articles when we did our research. It is not a fair comparison and we do not claim they all are state-of-the-art. The systems have all different conditions as languages, training and test data, trying to recognize different kinds of named entities, and a few systems are a bit dated.

There is really no comparison to speak of unless the systems use the same training and test data, nevertheless we think Table 7.7 gives some insight and something to relate to, but take it with a grain of salt.

Table 7.7: Various NER Systems, FS (context-sensitive finite-state grammars), CG (rule based system on constraint grammar), ME (maximum entropy), MBL (memory-based learning), gazetteers (another word for lists)

System	Language	F-score
Winner of Conll 2003 FIJZ03 [10]	English	89%
Winner of Conll 2003 FIJZ03 [10]	German	72%
SweNam [11]	Swedish	51%
The Icelandic NameFinder system [24]	Icelandic	79%
The Swedish FS system, with gazetteers [13]	Swedish	92%
The Danish CG system, with gazetteers [13]	Danish	88%
The Danish FS system, with gazetteers [13]	Danish	64%
The Norwegian MBL system, with gazetteers [13]	Norwegian	68%
The Norwegian ME system, with gazetteers [13]	Norwegian	61%
The Norwegian CG system, with gazetteers [13]	Norwegian	61%

Chapter 8

Conclusion

8.1 Problems

The problems with developing a NER system for cellular phones has several issues:

- Cell telephone limitations of processing power, storage, and memory.
- The poor language use in text messages impairs the POS-taggers ability to decide what word class a token belongs to.
- The nature of text message as being short and lacking information impairs the ability to extract information at a high level.
- The lack of corpora based on text messages, limits the possibility of training a good model, that is well rounded and can deal with most cases.

8.2 Accomplishments

During this Master's thesis we have accomplished:

1. Gathered an suitable corpus of real text messages. Came up with an unsupervised annotation scheme (regular expressions) to alleviate the manual annotation burden.
2. Dealt with all issues concerning: what to recognize, how to recognize, different approaches, how to evaluate performance, and the difference from text messages with "normal" text.

3. Developed a viable NER system for text messages with a strict F-score of 86% and partial F-score of 89%.
4. Proof of concept, by incorporating a NER system into a cellular telephone we showed that cellular phones are ready for natural language processing technology and we show some of their many practical uses with an SMS application.

8.3 Future Work

We would like to see more applications taking advantage of a NER system. We have proven that it is feasible and have practical uses. A NER system like ours does not have to be attached an SMS application. It could be used just as well for e-mails or even an dialer application if it performs a Speech-to-text transcription of the telephone calls. Such transcript can be analyzed just as any other text and once the user is finished with a telephone call she can ask the application to extract telephone numbers, booking details etc. that was mentioned in that call.

There are still improvements that can be done. You can expand with more *dynamic tags* to work with for example chunking on phrases, lemma of the words such as we did with part of speech. There is a hidden potential inside the telephone as you can track conversations between specific users (prolonged context). But more importantly, to utilize any information that is already present in the cellular telephone, i.e. in other applications, phone books, or call logs for telephone or name references, social networking applications, social games based on locations, Global Positioning System (GPS) lookup where the user is at the moment to include a more detailed list of locations for that neighborhood.

A prolonged context would likely be very helpful, linking conversation between users and finding context that transcend a single text message. I.e. an appointment that is agreed on is typically negotiated over several messages where key information like the final *where* and *when* are placed in different text messages.

There were a lot of features for the classifier that we did not get around to implement and test, statistics derived from training data, prefix/suffix of 1-4 characters of each word, functional words etc, the easiest way to figure out good features to have is to collect them from papers or shared task contest as CoNLL and see what works for your system.

Interaction is better than algorithms, no system will ever be perfect. You can arrange a application that will learn how to better predict from the users. For instance, if a user wants to make an appointment with information extracted from a text message, the system provides a partial location match *hotel* from “Grand hotel“ and the user adds the missing *Grand* part when he forwards the booking meetings to the calender. We now got a supervised system with an ever increasing gold corpus, that is if the user is willing to upload his or hers text messages.

Expansion and adaptation for several languages, right now it requires a little work to make the system totally language independent, but it should essentially be finding text message corpus for training the classifier, finding a text that is POS tagged to train the POS tagger, constructing list and regular expression for each language. You probably would also need a language detection tool to make the proper branching in a early stage.

A fully fledged POS tagger that is capable to run inside a cellular telephone or perhaps a larger corpus to train the POS tagger would be sufficient.

Confidence predictions from the classifier might be useful either to cut off uncertain predications to increase precision if that would turn out to be an issue for the users or if the classifier says the token has a low probability to be a location but it is not certain we can then double check with an on-line location lookup service.

Bibliography

- [1] Android Developers. <http://developer.android.com/reference/android/content/Intent.html>,.
- [2] OpenNLP POS-Tagger: (A package of java-based NLP tools). <http://opennlp.sourceforge.net/>,.
- [3] The Penn Treebank Project. <http://www.cis.upenn.edu/~treebank/>,.
- [4] The Writing Centre (University of Ottawa). <http://www.writingcentre.uottawa.ca/hypergrammar/partsp.html>,.
- [5] Wikipedia. <http://www.wikipedia.org>, year = "2010",.
- [6] Christofer Bach and Johan Gunnarsson. Extraction of trends in SMS text. Master's thesis, Lund University, June 2010.
- [7] Daniel M. Bikel, Scott Miller, Richard Schwartz, and Ralph Weischedel. Nymble: High-performance learning name-finder. In *In Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 194–201, 1997.
- [8] Tobias Ek and Camilla Kirkegaard. Detection of phone numbers, dates, time and names in handset text messages. http://fileadmin.cs.lth.se/cs/Education/EDA171/Reports/2009/tobias_camilla.pdf,.
- [9] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.
- [10] Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. Named entity recognition through classifier combination. In Walter Daelemans and Miles Osborne, editors, *Proceedings of CoNLL-2003*, pages 168–171. Edmonton, Canada, 2003.

- [11] Hercules Dalianis, Erik Astrom and Erik Åström. SweNam-A Swedish Named Entity recognizer Its construction, training and evaluation, 2001.
- [12] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University, 2003.
- [13] J.B. Johannessen, K. Hagen, A. Haaland, A.B. Jonsdottir, A. Noklestad, D. Kokkinakis, P. Meurer, E. Bick, and D. Haltrup. Named entity recognition for the mainland Scandinavian languages. *Literary and Linguistic Computing*, 2005.
- [14] Håkan Jonsson, Pierre Nugues, Christofer Bach, and Johan Gunnarsson. Text mining of personal communication. In *Understanding the technical and privacy related challenges. In 14th International Conference on Intelligence in Next Generation Networks (ICIN)*. Berlin, Germany, 2010.
- [15] Kann, Viggo. Granska Tagger: (A Part-of-Speech Tagger for Swedish). <http://www.csc.kth.se/tcs/humanlang/tools.html>, 2009.
- [16] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. pages 1137–1143. Morgan Kaufmann, 1995.
- [17] Lisa Larsson and Magnus Danielsson. Name extraction in car accident reports for Swedish. Technical report, LTH, Department of Computer science, Lund, January 2004.
- [18] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Linguisticae Investigationes*, 30(1):3–26, January 2007. Publisher: John Benjamins Publishing Company.
- [19] E.M. NRL and D.P. NRL. MUC-7 EVALUATION OF IE TECHNOLOGY: Overview of Results. 1998.
- [20] Pierre M. Nugues. *An Introduction to Language Processing with Perl and Prolog: An Outline of Theories, Implementation, and Application with Special Consideration of English, French, and German (Cognitive Technologies)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
- [21] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2003*, pages 142–147, 2003.

- [22] Statistiska Centralbyrån. Statistiska Centralbyrån (SCB). <http://www.scb.se>,.
- [23] The Stockholm Umeå Corpus. The Stockholm Umeå Corpus(SUC). <http://www.ling.su.se/staff/sofia/suc/suc.html>,.
- [24] A. Tryggvason. Named Entity Recognition for Icelandic Research Report.
- [25] Yin wen Chang, Cho jui Hsieh, Kai wei Chang, Michael Ringgaard, and Chih jen Lin. Low-degree polynomial mapping of data for svm.

Chapter 9

Appendix

9.1 Android Environment

The software to be developed will run on cellular phones with the Android operative system. Android gives a working development framework with some basic functionalities. The operative system is also *open source* which means that it is available for everyone to download and to continue to build on or to build new applications from scratch. It is easy for the user to download applications and install, as well as uninstalling them.

“An *Intent* in Android is an abstract description of an operation to be performed. [...] it is most significant use is in the launching of activities, where it can be thought of as the glue between activities. It is basically a passive data structure holding an abstract description of an action to be performed” [1]. We will use these intents to let our software communicate some of the built in applications in Android, for instance letting the Multimedia Messaging Service (MMS) application send an intent to the Calender application with information that is extracted from a text message to be used to schedule a meeting.

Developing software for Android cellular phones also comes with restrictions compared with software used on desktop computers. The cellular telephone itself offers a less powerful processor and less memory compared to a desktop computer. For instance, a application can not usually allocate more than 16 MB memory. The Android operative system limits the response time to 5 seconds for its applications before it assumes that the application has crashed. The software can by these reason not be too computation heavy or too memory reliant.

9.2 SNEEX

When performing this project at Sony Ericsson a by-product was an application able to perform *named entity* extraction from a cellular telephone text messages. Sony Ericsson named our application *SNEEX* a name based on the words **S**ms **N**amed **E**ntity **E**Xtraction. SNEEX is to be seen as a proof of concept that demonstrate functionality rather than a of the shelf product. It is a functionality, built into the existing MMS application that is included in the Android framework.

At the time that this report is sent to printing, SNEEX is extracting *named entities* from text messages and then if the user choses to, by clicking on the text message, it presents which entities it was able to extract.

We aim to finish up SNEEX before ending our project on Sony Ericsson so that the extracted information is converted into a formats that is accepted by other applications. Those applications will be Google Maps, the record of contacts, and the calendar application. All of them included in the Android Framework and thereby present in all Android cellular telephones.

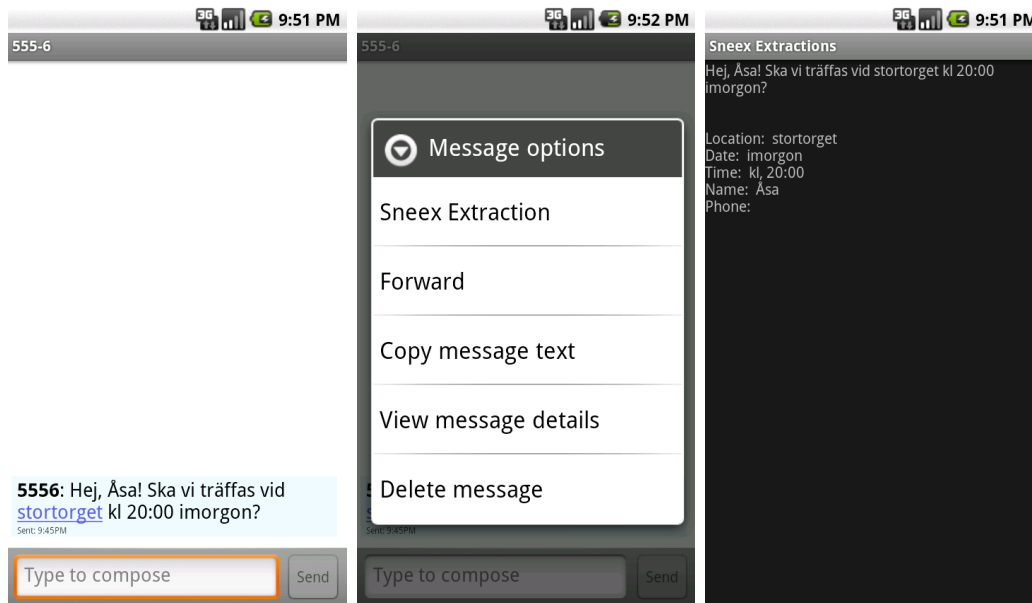


Figure 9.1: Images from the Android prototype SNEEX. First image displays a incoming text message with the location *stortorget* highlighted. Second image displays a context menu with *SneexExtraction*. Third image displays the extracted information from the text message.