Applications of Named Entity Recognition in Customer Relationship Management Systems

Farbod Saraf Jadidian

September 2014

Dissertation submitted in partial fulfilment for the degree of
Master of Science in Information Science

Computer Science Department
Radboud Univers

Attestation

I understand the nature of plagiarism, and I am aware of the University's policy on this.

I certify that this dissertation reports original work by me during my University project except for the following (*adjust according to the circumstances*):

- The theory review in Section 2.1 was partially taken from[2].
- The technology review in Section 2.3 was partially taken from TUD Palladian Overview[1].
- The code discussed in Section 5 and Appendix was developed by OpenNLP Group[2] and was used in accordance with the licence supplied.
- The training data used in Section 5 was provided by NLP-GEO project[3].

Signature Date

[1] http://palladian.ws/documentation/
[2] http://opennlp.apache.org/
[3] https://code.google.com/p/nlp-geo/

Acknowledgements

I would like to take this opportunity to express my appreciation to everyone who supported me throughout this master thesis project. I am sincerely grateful for their knowledge sharing, invaluable guidance, constructive criticism and friendly advice during the project.

It gives me a great pleasure in acknowledging the support and help of my supervisor Professor Th. van der Weide. Without his guidance and persistent help, this thesis project would not have been possible. I am also grateful to P. van Bommel for accepting my request to be the second reviewer for this project.

I would also like to thank my project external guide Mr. de Rooij from the company Soluso B.V. and all the people from OpenNLP team who provided me with the facilities and resources being required for conducting my thesis project.

At the end, I am indebted to my parents and close friends who supported me by preparing a proper atmosphere for me to able to focus on the project and manage to finish it with my desirable result on time.

Table of Contents

List of Figures

1 Introduction

Natural language processing refers to human-computer interaction in terms of Linguistic; in simple words, understanding of the human language by computer or the natural language generation by the machines. Dealing with natural language processing has been always known as one of the complex fields in computer science. As time goes by, more progress are being made to improve the performances of natural language processing systems. However achieving the nearly human performance in NLP application is not pragmatic due to numerous reasons such as ambiguities in Languages, slangs, sarcasms and so on. There is a wide range of fields in Natural Language Processing such as Parsing, Speech recognition, Machine translation, Information retrieval, Part-of-speech tagging and so on. In this thesis, the focus is on Named entity recognition, finding a solution to detect and extract entities such as persons, locations, organizations, dates in order to automatize several tasks in a CRM application, namely creating pre-filled forms, an agenda item, saving contacts information, making summary of an email or a report and so on.

Due to the complexity of the field, introducing a new algorithm or making a system from scratch would not be a solution with regards to the limitations of the master thesis. Therefore, the first step was checking state-of-art of the field to explore all the existing algorithms and techniques as well as their implementations as toolkits, software, packages, and libraries which is elaborated in the second chapter. Analysis of the system requirements was also a major step which was conducted in the third chapter. With regards to different aspects of requirements ranging from performance to licensing issue and technology limitations, out of all available systems OpenNLP is nominated, a Java-based NLP library which is distributed under Apache 2.0 License. This toolkit offers the most common NLP tasks, such as part-of-speech tagging, coreference resolution and so on. In the fourth chapter, more details about OpenNLP is elaborated, including the introduction to the package, instruction for training a model and modifying the code with regards to the requirements. To check the performance, a model is trained out of the training data which is annotated for entity “Location”. Using the trained model, the accuracy of the recognition for location entitles were evaluated, 0.84 for precision, 0.45 for recall and 0.59 for F-measure. The high precision indicates that the 84% of all recognized entities were actual locations which means the output of the model is reliable enough for using in the targeted system. However, the recall of 0.45 means not all the entities were recognized, indicating that the amount of annotated data for training the model was less which can be solved by collecting and annotating more data from available resources. The overall performance was almost 0.6 which is acceptable for the provided amount of training data to the system.

In the end, the modification of system to improve the performance is discussed. The NER engine of OpenNLP has been implemented using maximum entropy algorithm. A bootstrapping method is based on running the model and modifying the result recursively to remove the undesired known entities and to add ignored entities to the system for boosting the performance.

2 State-of-The-Art

The first step in a proper research is to find and examine all existing technologies specifically the State-of-The-Art, helping the researcher to find the most appropriate techniques which are not only fulfil the theoretical aspects of the problem but also it is practical enough to implement using existing resources. In this chapter, all the findings regarding both theoretical and technical aspect of State-of-the-Art are summarised. There is a wide range of literatures introducing new techniques, concepts and solutions. In upcoming parts, first all theoretical aspects of state of the arts are introduced, followed by the state-of-the-art techniques for the evaluation of NER systems. Last part of this chapter is covering all existing implementation of those techniques which is available (either freely or commercial) in the area of Natural Language Processing.

2.1 Theoretical aspects: Survey of Named Entity Recognition (NER)

Named Entity Recognition field has its roots back in the days in 1991 when Lisa F. Rau represent his first research papers at the 7th IEEE Conference of Artificial Applications, recognizing and extracting "company names". Most of those old methods were relied on handcrafted and heuristic rules. After investigating a variety of papers and surveys in the field of named entity recognition and classification**Error! Reference source not found.**, the conclusion has been made that the whole existing techniques can be categorised in three major fields below:

Hand crafted rule-based algorithms, Feature Space and Machine Learning techniques. However, machine learning techniques and feature space are much more new compared to Rule-based algorithms, based on the system requirements it is possible to either go in one direction or combine different methods to achieve a highest performance. Upcoming sub-chapters describes more details about all mentioned techniques.

2.1.1 Handcrafted Rule-based Algorithms

As it is mentioned, handcrafted rule-based algorithms are the oldest techniques in this fields. In general, all works and results by different scientists in this area can be seen as three major factors of 'Entity type', 'Domain' and 'Language factors'. Some of them worked on specific topics such as "Enemex" recognizing names of Type "Persons", "Locations" and "Organizations" whereas there are some papers regarding Open Domain covering a wide range of types. Some other works in this field was done for specific textual genre, for example, in 2005 E.Minkov et al. designed a system for email documents. However, it sounds practical to convert a system from a specific domain to another domain but it requires a lot of time and effort since it is a complicated task. In terms of language factor, most of the works are done in English but also there are some other scientists and specialists which tried to broad the same features in other languages such as Dutch, German, Chinese and etc.

2.1.2 Feature Space

Features are being considered as attributes which representing the characters of words being used by different algorithms. For example, a simple feature vector can be a Boolean attribute which stands for showing whether the word is started with capital or small letter. Although a wide range of features stands for representing different aspects of words, in most cases they are not merely enough to solve the real world NER problem. However, when they are coupled with some machine learning techniques they contribute to best performance which is the one of the state-of-the-are techniques using as solutions to NER problems. Based on the survey of named entity recognition and classification the most used feature in NER can be divided into three categories:

2.1.2.1 Word-level features

As the name implies, Word-level features are describing the character composition of words. There is a wide range of word-level feature, out of which the most common features in NER is listed below:

- Digit patterns

  This pattern specifically stands for digits such as dates, percentage, amount and so on. For example, if a two-digit numbers followed by a dash, two more digits, another dash and four-digit number (xx-xx-xxxx) the system can learn to use this pattern for recognizing a date.

- Common word ending (Morphology)

  It is mostly related to the origin of the words and affixes. For example a system can learn to recognize words followed by "land" as candidates for countries such as Nederland, Scotland, England and so on.

- Function over words

  There is a wide variety of useful features which is extractable from words out of applying functions. One of the most common way of implementing function over words is to use n-grams. In the upcoming technical chapter, there is a detailed information about n-grams.

- Part-of-Speech

  In general, POS refers to lexical categorizing of a word in the sentence (i.e. subject, object, verb and so on). This kind of lexical features might help to recognize a pattern especially when it combines with machine learning methods. However, for most of NER solution POS as feature will not improve the result but still in some cases it might be necessary to take it into account.

- Patterns and summarized patterns

  Patterns features for the first time presented by M. Collins in 2002. Using this kind of features helps to map word(s) to set(s) of characters in order to summarize them. For example, a summarized pattern can map all the capital

letters to X, small letters to x, digits to 0 and Special characters to _; then the result would be:

Heyendaalseweg 135, 6525 AJ Nijmegen = Xxxxxxxxxxxxx 000_ 0000 XX Xxxxxxxx

2.1.2.2 Documents and Corpus Features

One of the best source of features is a large and proper amount of corpus. While going through and processing the whole corpus a lot of features can be extracted from single or multi-words, based on the relation between words and sentences and Meta information lying under the corpus which is hard to recognize without using statistics. Here are some extractable features from documents and corpus:

- Entity coreference and alias

  One of the difficulties in detecting the name entities is when a same word is occurred across the documents in the various makeups. Furthermore, Aliases might lead to more difficulties since they represent a same name in another manner. For instance in all these words share the same meaning which represent the company: corporation, Ltd., Corp, company, B.V. and etc. Recognizing coreference and alias seems like the same as entity recognition in terms of difficulties; however, using combination of some techniques such as machine learning as well as world level feature might lead to solve this problem much easier. There are more solution for solving the problem of entity reference such as semantic tagging, using heuristic rules and so on.

- Meta information

  Meta information can play an efficient role in detecting named entities since most of them can be used directly. For example URL of a document can bring some useful information regarding the content of the document. Another example can be title of a research paper, providing a clue for the context of document. Or even more simple and clear example is the title of an email, indicating the greeting followed by a name of a person and in some cases contact information such as name of companies. There are more and more example in this area such as XML section, tables, figures and so on.

- Local syntax

  Using the position of a word in sentence, paragraph or even in document might provide useful information. Furthermore, another application of local syntax is using Enumeration, which refers to having set of a groups of words which are related to each other. For example, November, August and July all are in the same categorization.

- Multi-word units

Processing large amount of corpus can help the system to extract features for detecting multi-word entities. Out of statistical methods, in 2004, Da Silva managed to define some useful feature functions over multi-word entities in large co. For example, he defined the threshold on selecting a multi-word unit as named entity candidate.

2.1.2.3 Dictionary Feature

Dictionary or list look up feature refers to using lists (dictionaries, lookup tables or etc.) for recognizing entities base on their probabilities of being targeted entities. The dictionaries are created out of processing a large amount of corpus to calculate the probability of a word being a specific entity once it is detected in the context. For example, when the word "Nijmegen" is appeared in a text there is a high likelihood that it refers to the city.

There is a wide variety of lists in literature. Based on a survey of named entity recognition and classification **Error! Reference source not found.**, three major categories of lists are:

- List of entities
  It is just a big list of all entities covering areas of First name, Last name, Organization, Airline, Government, Educational, Celebrity, Continent, Astral body, Country, City, State and so on.
- List of entity cues
  A lot of entities can be recognised by identifying words which mostly they are coupled with. It can be pre-fix, person title, location typical word, post-nominal letter and so on. For example when Ltd or Corp. are coupled with a word there is a high chance that the word is an organization name.
- General dictionaries:
  General dictionaries are pre-existing lists containing common nouns which can help a lot in recognizing entities. For example, when first letter of the first word of a sentence is capitalised it can lead to some disambiguates whether it is an entity or it is only capitalized because it is located at the first of sentence. Referring to a general dictionary can solve this type of issue to an acceptable extent.

Exact matching with pre-existing dictionaries would not provide a satisfactory results. However, there are several ways and techniques in order to achieve more flexibility and better results using lists:

- Accessing lists via Soundex algorithm is one of the way to increase the flexibility. Soundex is an algorithm for mapping names into their sound using their English pronunciation. Soundex code algorithm use the first letter of the word as well as three digits repressing the sound. For example, Radboud University has been stated in two ways in different web sites. In some corpus it is appeared as "Radboud University" while in

some others it is “Radboud Universiteit” (Dutch version). However, University and Universiteit has a same Soundex code (U516). Therefore we are able to embed this technique to the old one to detect both words in corpus.

- Most of the time words are not stating as their origins and roots. Therefore, normalizing words and removing derivational and inflectional suffixes from them before matching helps to have more correct detections. For example, “Prof.” can be normalized to “Professor” in order to be detected as a cue for a name.
- One of the most effective solution is using a fuzzy-matched technique. It check the lists using edit-distance factors. It check the distance of each words to the words in the list and if the edit-distance of the word with one of the words in the lists was less than a threshold then it will nominate that word as potential candidate.

As it is already mentioned, using feature space merely might not be enough to solve NER problems. However, when they are coupled with machine learning the system achieve highest performances. In next chapter, different techniques of machine learning for NER systems are described.

2.1.3 Machine Learning Techniques

In contrast to early systems which were based on hard-coded rule base system, nowadays most of advance named entity recognition systems are using machine learning techniques to boost the performance. Machine learning techniques gives the system ability to learn based on input and induce the rules for detecting entities. In general, three sub categories of unsupervised, semi-supervised and supervised learnings:

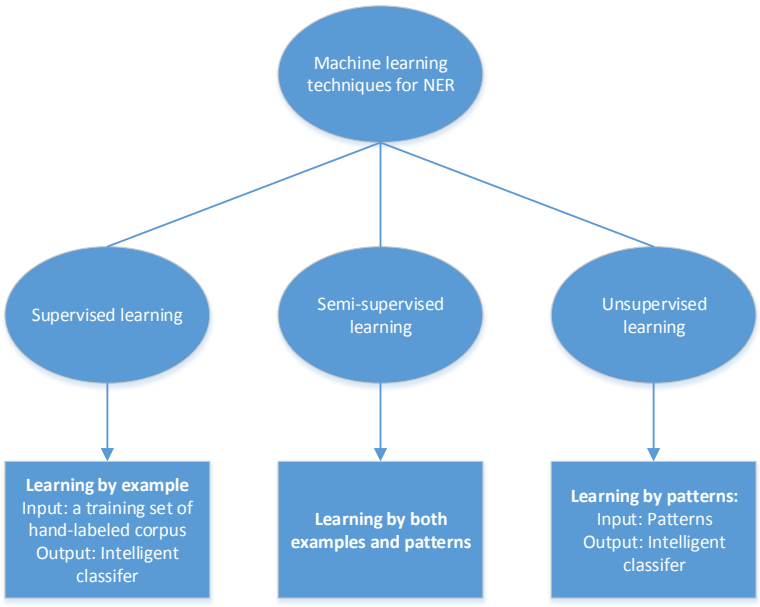


Figure 1. Machine Learning Techniques

2.1.3.1 Supervised Learning

In supervised learning, the classifier is trained using a set of hand-labelled data. The whole process can be summarized to the diagram below (Figure 2):

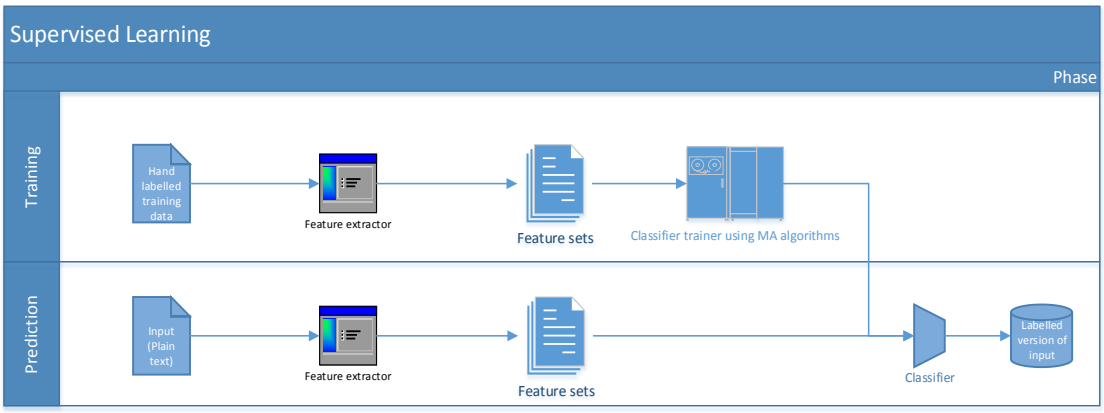


Figure 2. Supervised Classification Process

As can be seen in figure 2, in supervised learning the features should be extracted beforehand, in order to train a classifier based on extracted features using machine learning algorithms. Once the classifier is trained, the same feature extraction phase is required for the input data since the classifier accomplishes its tasks out of feature sets. The whole process might look rudimentary, however, choosing the right features to extract is one of the trickiest step in NER process which have direct impact on the performances and results. Furthermore, choosing the right implementation of feature extractor plays a vital role as well. In order to come over of these crucial issues, proper analysis of the requirements and goals of the system should be taken into the account from the first steps. Once the results of analysis are listed as requirements, the feature extractor should be built base on trial and error method to check which strategy suits the problem the most.

Training a classifier requires selection and implementation of machine learning algorithms. There is wide range of algorithms available for named entity recognition. Out of which we selected those who were implemented the most in the freely available software and packages. Although in several distributions the combination of different methods are used, having insight over each probabilistic models would contribute to the proper selection of the software with regards to the requirements of the final system. In next sub-chapters, first two categorization of learning models namely generative and discriminative is described, followed by the elaboration of Decision Tree, Naïve Bayes, Maximum Entropy and their pros and cons.

- **Generative and Discriminative Models**

  Learning model are categorized into two groups of generative and discriminative (conditional). Generative models distribute probabilities over both observed data *x* and hidden classes *y*. In contrast, Discriminative models are

using natural distribution for classifying x, assigning the probability of belonging to a group y to a give parameter x: $P(a \mid x)$. A simple example makes it clearer, consider the sentence below is selected from a corpus:

Mr. Amelink is visiting Soluso in order to establish a joint collaboration between Amelink B.V. and Soluso.

Selected entities are listed and categorized below in the format (*x,y*):

(Amelink, Person), (Soluso, Organization), (Amelink, Organization), (Soluso, Organization)

Then for $P(x,y)$ two probability distributions are listed below, the former for generative and the latter for discriminative algorithm:

	y = Person	y = Organization
x = Soluso	0	0.5
x = Amelink	0.25	0.25

Table 1. Joint Probability Distribution

	y = Person	y = Organization
x = Soluso	0	1
x = Amelink	0.5	0.5

Table 2. Conditional Probability Distribution

- **Getting Features from Corpus to Use in Models**

As it is already mentioned, feature is an elementary piece of evidence which leads to predicting class C for the observed data D. Mathematically speaking, feature f can be considered as a function which map a real value to a space of classes and a piece of data:

$$f: C \times D \longrightarrow R$$

For example for this sentence:

Mr. Amelink is visiting Soluso in order to establish a joint collaboration between Amelink B.V. and Soluso.

One feature for Amelink B.V. which is an organization would be:

$f(c, d) = [c = \text{Organization} \wedge w[i+1] = \text{“B.V.”} \wedge \text{isCapitalized}(w[i])]$

This feature map a number to the fact of belonging to the category ‘organization’, being capitalized as well as being followed by the word “B.V”.

Models are trying to give each feature a weight which is a real number. It can be either positive or negative. The positive numbers stands for the correct configuration while the negative is representing that the configuration is probably incorrect.

Conditional or discriminative probabilistic models are much more useful nowadays, reason:

- High accuracy and performance
- Making it easier to include a variety of linguistically features
- Incorporating to automatic building of NLP systems in a language independent manner.

Downsides of discriminative to generative models is that Conditional models can easily memorize much of the training set, contributing to over fitting of too much information by observing and memorizing everything which may not appear in test data.

Generative models are also transferable to Discriminative out of applying Bayes rules in or-der to use for classification. In terms of classification, Discriminative models outperform Generative models. [5]

Models such as n-gram models, Hidden Markov Models (HMM), Probabilistic context-free grammars, Naïve Bayes classifiers and so on are generative, while some others like Conditional random fields (CRM), Logistic regression, Maximum entropy models (Maxent) and so on are known as discriminative or conditional models.

➢ Decision Tree

One of well-known learning methods which contribute to building a classifier automatically is Decision Tree, reacting as a flowchart to assign a class to the new input. Each decision tree consists of decision nodes for checking the features and leaf nodes for assigning the classes based on the result of decision nodes. The first decision node is known as root node, assigning a class to a new input by checking the value of one the input’s feature. After this phase, the input which is classified already for one step is now considered as an input at a new decision node. The process is repeated for each new decision node until we achieve a decision leaf which can assign a label for input value. The example is provided below:

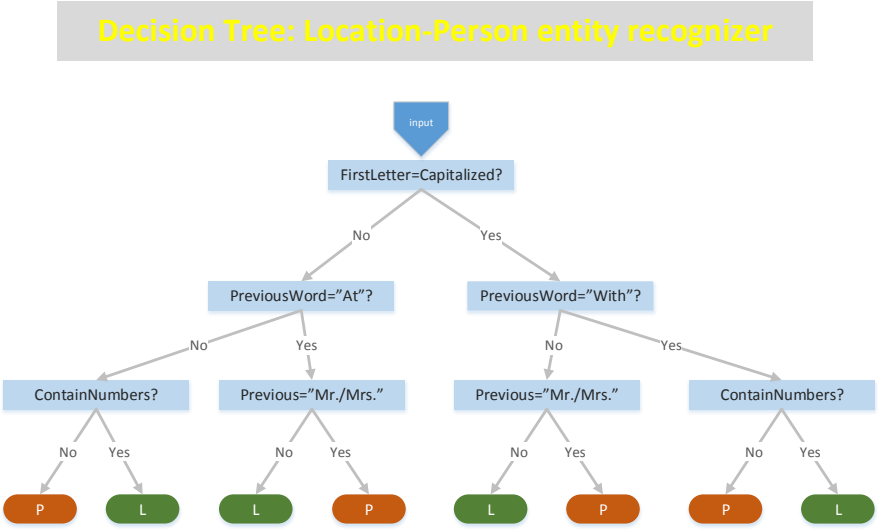


Figure 3. Decision Tree Method

Once the decision tree is trained, it would be straightforward to classify a new input. Hence the complicated part of this solution is to train a proper tree using an efficient and procedural strategy. In order to build up an appropriate tree, there are some steps to take into account. The first and required step is to find and choose the most effective decision stump for a set. Decision stump is a single decision node which classify the input regarding a single feature. To simply the task, it is possible to consider all features separately and assign each feature a decision stump, selecting a class for input which is the most frequent regarding that feature in the training set. Each class is considered as a leaf of decision stump. The leaves are changing continuously to find a class which maximize the performance and accuracy.

There is a wide range of methods for finding the most appropriate and informative feature for the decision stump. An information gain is another solution which calculate how disorganized the data is, using entropy of their classes.

- *Entropy:* An attribute stands for indicating how data is distributed. The more disorganized a data is, the higher an entropy would be. In other words, if the classes of input data is varied immensely, the entropy would be high, in contrast when the input data share the same classes the entropy is low. The entropy is calculated out of multiplying the sum of the probabilities for each label to log probability of that specific label.

$$H = -\sum_{i} P_i(log_2 P_i)$$

Using entropy helps to find a proper classes which occurred as medium scale. If a class is occurred rarely then Pi would be so small so H is low, on the

other hand if a class extremely occurred within a corpus then $log_2 P_i$ returns a small value, contributing to a low entropy.

As it is previously mentioned, finding classes with the highest entropy or in other words finding decision stumps with the highest information gain would help to have an efficient decision tree. However, decision trees has their own pros and cons as other methods. Decision trees are simple and easy to depict as well as interpret the results. Furthermore, in order to find the most useful features, decision trees can play an effective roles. On the other hands, the advantages can be "over fitting" the data set, which means by division of data set into two data set at each nodes, the amount of training data for lower branches might be so low that lead to unreliable results. There are some solution to this issue, such as producing the full decision tree and applying prune to decision nodes using dev-test for those who are not affecting the performance. Furthermore, a major deficiency of this method is checking features in a specific order which might lead to undesirable results since not all the features are relatively dependant on each other. To overcome this drawback, Naïve Bayes Classification offers possibilities to all features to act equidistantly which will be discussed in next part.

- **Naïve Bayes Classifiers**

As it is discussed previously, the main goal is to assign a class (C) to a word (W). In order to classify base on Naïve Bayes algorithm, two types of probabilities are involved. Firstly probabilities of all classes should be calculated P (C_i). This task can be done out of processing labelled training courses, practically speaking: number of the specific of a group occurred in the corpus divided by the total amount of words in the corpus. The second type of probabilities is P (F_i | C) stands for the probabilities of each feature (F) give class (C). In order to simplify the task, assume that features are binary and independent. Then the equation would dividing the number of words in training set with the feature and class by the number of words with the given class:

$$P\ (F|C) = \frac{Num(F, C)}{Num\ (C)}$$

The multiplication of both factors is the key to the Naïve Bayes classification. For each input, all the probabilities are calculated and the winner is a class which maximize the result of the equation below:

$$S\ (C, W) = P(C)\ \times \prod_{i=1}^{n} P\ (F_i \mid C)$$

The formula might looks rudimentary, however using of this equation merely might lead to unreliable results. There are numerous reasons which cause variation in results, decreasing the performance. Few tips might help to boost the result:

i. **Smoothing:** In the formula above if in the training set there is no word with the feature (F) with the given class (C) then the result of $P\ (F|C)$ would be zero which means there is no chance of a word happening in the test set with the given feature (F) and class (C). In order to prevent this kind of variation in the results, sophisticated solution such as implementing smoothing techniques are required.

ii. **Non-binary to Binary**: As it is already stated, to simplify the task all non-binary features should be translated to binary features. One of the solution is to translate multi-class feature to several binary features. For example, for multi -class feature "WordType = {noun, verb, adjective, adverb}" it would be rational to have four binary feature "WordIsNoun = {0, 1}" and so on. Furthermore, regression methods can be used for the translation of numeric features to binary ones.

Although the method is very simple and easy to implement, it has some major deficiency which makes it not efficient to implement in real sophisticated word. For instance, Double-Counting. This drawbacks comes from the processing of the feature in the independence manner. In other words, a lot of features has some correlations. However, since in one hand, the contribution of all features in training set are processed independently and on the other hand for classifying new input all the features are combined, the contribution of those correlated features might be overestimated. For example, two feature F_1 and F_2 stands for ends_with(1) and ends_with(odd numbers), if the new input ends with 1 then both features are applicable and their contribution weights would overrate the final judgments. To come over this deficiency, Maxent algorithm is a proper alternative which will be discussed in next part.

- **Maxent Classifier**

Maxent stands for Maximum Entropy classifier, sharing a lot of similarities in terms of model they use. The major difference is that Maxent uses search methods to find specific parameters which maximizes the total likelihood of training data, in contrast to Naïve Bayes which finding parameters out of using likelihoods. Direct calculations of *P (label | feature)*, does not merely contribute to finding a reliable set of parameters due to complicated interrelation between some features. The problem is solved by a technique called iterative optimization.

- Iterative optimization: This techniques consist of following steps, starting by initializing the model by random parameters, redefine the

parameters continuously to find the most optimized set. Every iteration guarantee that new set of parameters is more optimized, however there is no solution to determine whether the most optimized set is achieved or still there is a room for improvement. This fact makes this technique time consuming especially when there is a wide range of features and classes available in the training data.

Mathematically speaking, in Naïve Bayes model, each class and pair of (class, feature) has a parameter to calculates the likelihood. Nevertheless, in maximum entropy, combination of features and classes has their own parameter (joint-feature).

2.1.3.2 Unsupervised Learning

In unsupervised learning, all techniques are based on lexical resources and patterns. In order to find named entities, for every new input, models refers to the most similar classified word in a lexical resource such as WordNet which is created out of processing a large amounts of corpus. One of the most outstanding advantages of this method is that there is no human annotated corpus is required in order to train a model. In other words, an input of the system is syntactic patterns with specific properties and the system tries to create a rich list of named-entities. Referring to wide list of entities is not merely enough since it can cause a lot of ambiguities. Nevertheless, the techniques are covering ambiguity resolution techniques to achieve reliable results.
An example for unsupervised learning is the method developed by R. Evans at 2003 in order to identify hypernyms and hyponyms which is capitalized in the text. For instance, if "Microsoft" is capitalized in the text and we are looking for its hypernyms, the query of "such as" might contribute to a solution out of searching on the web and retrieving corpus. The word that preceding the query "such as" is the most probable candidate as a hypernym for "Microsoft", in this example "Organization such as Microsoft" was the most occurrence of passage, giving the clue that the hypernym of "Microsoft" is "Organization".

2.1.3.3 Semi-supervised Learning

Semi supervised learning stands between supervised and unsupervised ones. SSL techniques require small amounts of seeds (e.g. hand annotated data) in order to start the process. The main method in SSL is bootstrapping which is elaborated by an example. For instance, consider a named entity recognizer system which is implemented using SSL specifically bootstrapping approach. To initialize the system, names of few organizations are given to the system, out of which system tries to find sentences that include seed examples, followed by identification of contextual clues which is shared between seeds. The process is followed by finding new entities and context using contextual clues extracted previously. Repeating this process continuously contributes to extraction of numerous new entities and contexts.

The methods might look simple and autonomous which might rise questions over its performance. However an experiment by Nadeu *et al.*in 2006 shows that performance

of Semi-supervised learning methods is comparable to the baseline supervised techniques.

There are some other methods rather than bootstrapping for semi-supervised learning. Mutual bootstrapping was introduced by E. Riloff and Jones in 1999. It suggest how to initialize with a handful of seeds of a given type instead of initializing with pre-defined named entities. Furthermore some techniques were introduced in order to boost the performance of existing methods, such as using syntactic relation in order explore more reliable contextual evidence near entities which is for the first time used by A . Cucchiarelli and Velardi in 2001.

2.2 Mind map of NERC

In previous chapter, theoretical aspects of named entity recognition embedded into existing systems has been elaborated. Nevertheless, it is mostly focused on specific and related topics regarding requirements of the system which will be discuss later. In order to find which most related techniques, the overview of survey of named entity recognition is visualized using mind map representation technique.

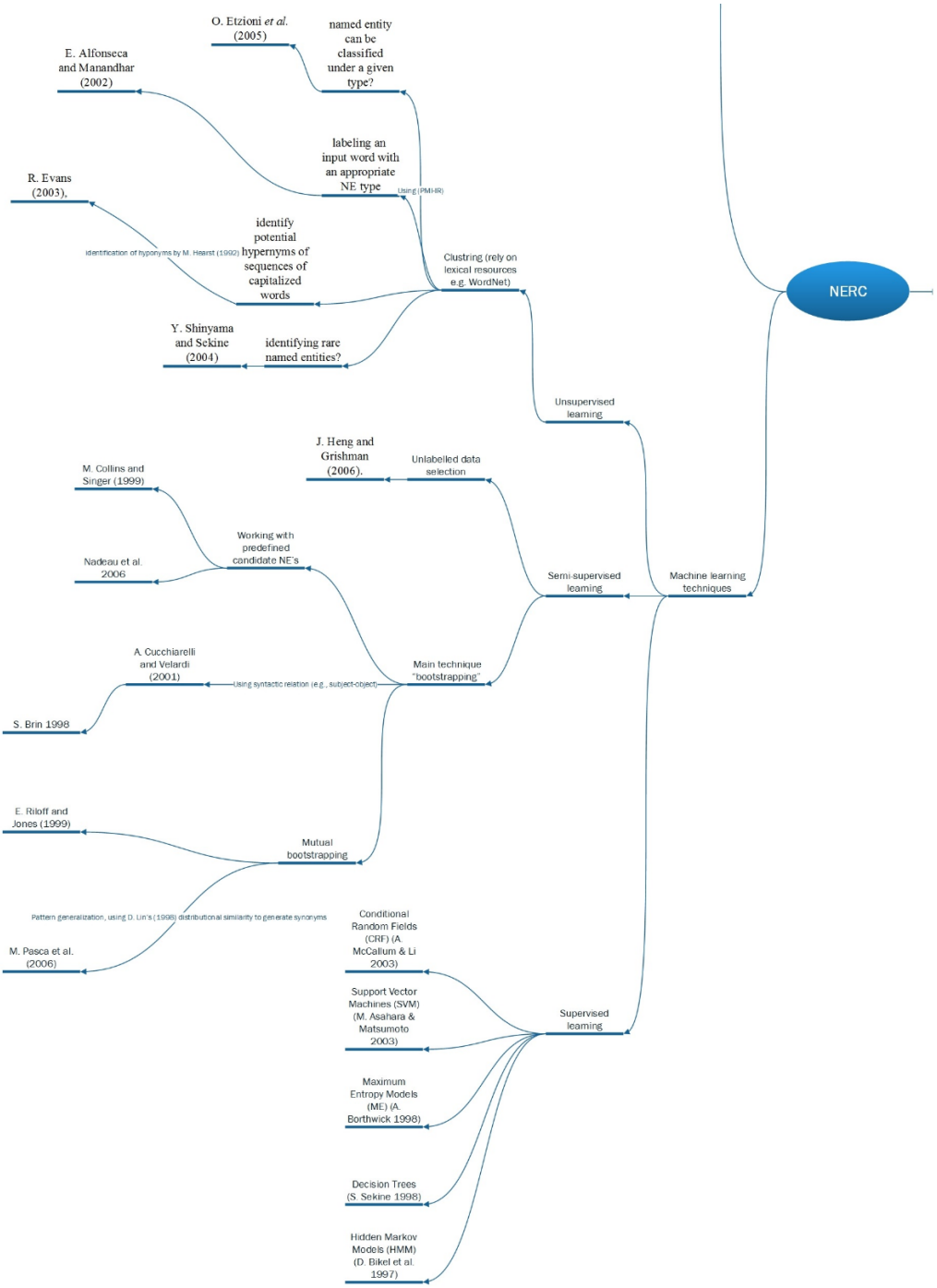


Figure 4. Mind Map of NER: Machine Learning Methods

As can be seen, since the diagram is enormous, the mind map is divided into 3 sub-roots, namely machine learning techniques, handcrafted rule-based algorithms and feature space. In addition to the detailed description for each part, some branch contains information regarding the related papers such as the name of the authors and their publish date.

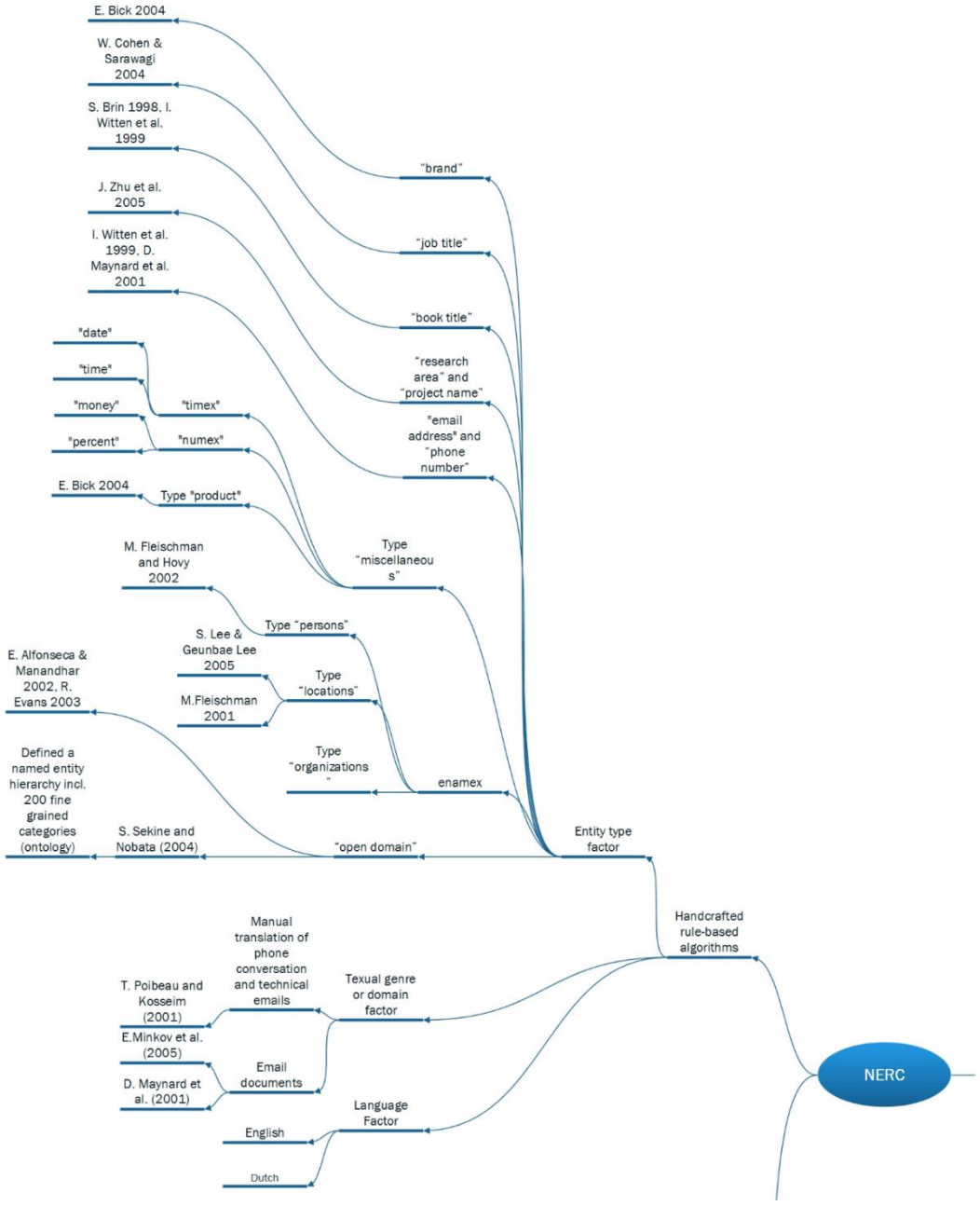


Figure 5. Mind Map of NER: Handcrafted Rule-based Algorithms

The last part of the mind mapping which is the biggest part is allocated to feature space. As it is already mentioned, features merely are not able to solve complex problems, therefore it should be coupled with other techniques such a machine learning to accomplish the entity recognition task.

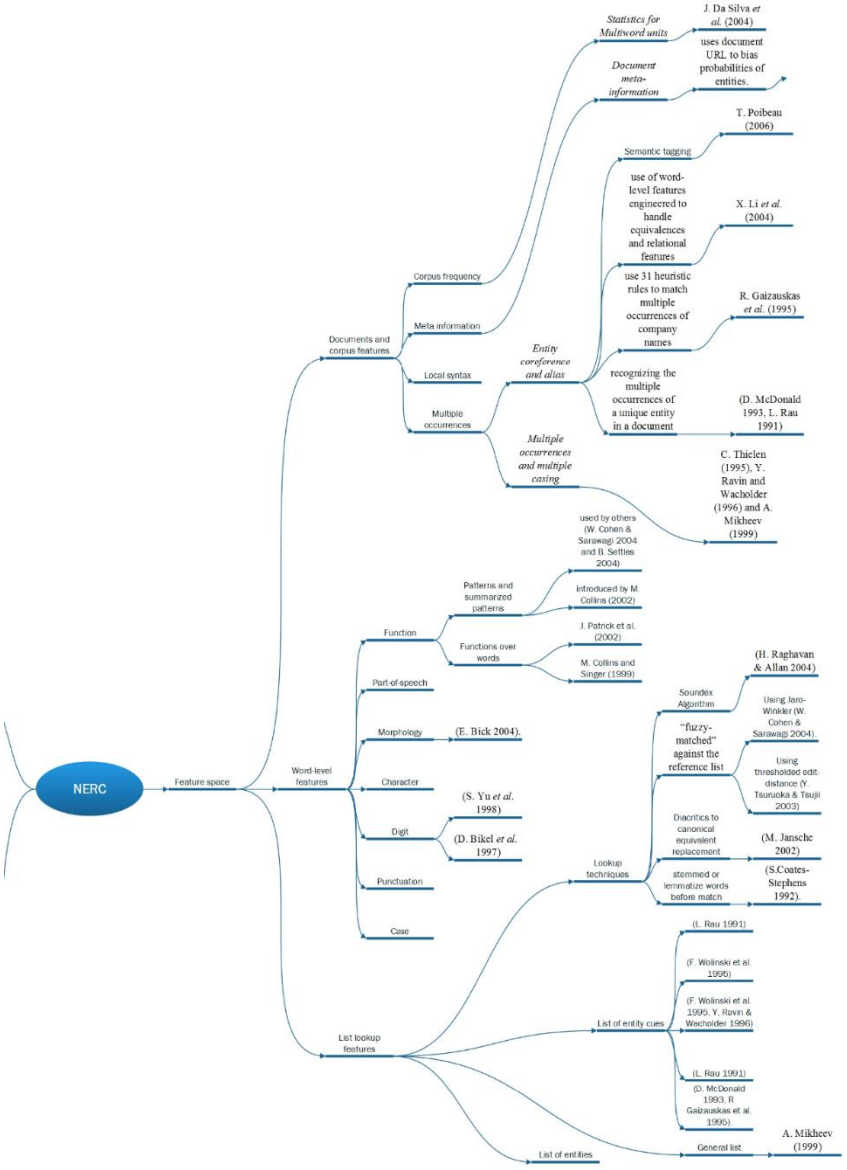


Figure 6. Mind Map of NER: Feature Space

2.3 Evaluation of Named Entity Recognition Systems

The evaluation of named entity recognition simply refers to the comparison between the results returned by system and human expectations. There is wide range of strategies and methods available for NER systems which can be implemented regarding techniques embedded into the system. In most of evaluation methods, three attributes play

vital roles, namely Precision, Recall and F-Measure. Before introducing these attributes, understanding of 2-by-2 contingency table is required.

2.3.1 Contingency Tables

To evaluate each part of data, 4 states are available, namely true positive, true negative, false positive and false negative. Describing four states using 2-by-2 contingency table makes it more clear and understandable. For instance, consider that there is an entity "Radboud" in the text and the system detect it as an entity "University". In this case the answer is correct. Another example, a word "Soluso" is appeared in corpus and the system has not recognize as a name of company which is the incorrect perdition. In contingency tables this attribute can have both value of true and false, based on the situation which will be described later. One dimension of table stands for mentioned characteristic, representing true and false prediction using two columns. On the other side, there is another dimension of table, indicating whether a prediction is accomplished or not. In other words, if system predict an data piece as an entity then attribute assigned to this prediction is positive, while negative is stands for a piece of data which is not detected as an entity, no matter whether it is true or false. This aspect of the result evaluation is depicted by two attributes of negative and positive representing rows of the table.

The combination of both previously mentioned attributes is leaded to a table called 2-by-2 contingency table.

	Correct	Not correct
Detected	True Positive(t_p)	False Positive(f_p)
Not detected	False Negative(f_n)	True Negative(t_n)

Table 3. The 2-by-2 Contingency Table

As can be seen, the main parameter of the table stands for true predictions, out of which the accuracy of the system is calculable.

$$Accuracy = \frac{t_p + t_n}{t_p + f_p + f_n + t_n}$$

However, accuracy does not a good measurement to evaluate the performance of a system. For example, consider a system designed to detect name of universities and we have a corpus of 500,000 words, out of which only 50 words are university names. Assume that system could not detect any of them. Then the contingency table would be filled with values below:

	Correct	Not correct
Detected	0	0
Not detected	50	499,950

Table 4. The 2-by-2 Contingency Table

Then if the accuracy is calculated based on the formula above the result would be:

$$Accuracy = \frac{499{,}950}{500{,}000} = 99.99\%$$

As it is proven, the system which was not able to detect any of entities are considered as a high accurate system with the accuracy of 99.99 percent. Therefore, this kind of accuracy is not a reliable indicator of evaluation and another. Effective evaluation requires a solution which take more factors into the account. This issue is solved by using precision and recall.

2.3.2 Precision and Recall

Precision and recall are two effective attributes which can be calculated out of contingency table. Precision is a factor to measure to how extent the guess of the system was right, mathematically speaking:

$$Precision = \frac{t_p}{t_p + f_p}$$

On the other hand recall stands for how effective the system detects the entities, in other words:

$$Recall = \frac{t_p}{t_p + f_n}$$

As it can be noticed, in previous example it was the attribute True Negative (t_n) which affects the result of the accuracy formula. In both precision and recall true negative is not involved, contributing to more reliable evaluations. Nevertheless, for having a well-designed system there should be a proper tradeoff between both precision and recall, hence based on the requirements some systems are made to have higher precision while the others are more effective in recall. These differences in terms of evaluation makes it complicated to have an overall comparison between the performances of different systems. Establishing a new measurement method called F-measure is a solution to the case, involving both precision and recall factors.

2.3.3 F-Measure

F-measure is a sort of weighted harmonic mean for evaluating the trade-off between precision and recall. There are two equal formulas F_α and F_β for the calculation of F measure using two custom variables α and β :

$$F_\alpha = \frac{1}{\alpha\frac{1}{\mathrm{P}} + (1-\alpha)\frac{1}{\mathrm{R}}}$$

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

Arithmetically speaking, both α and β should be customized to the amounts which fulfill the requirements of system via strengthening the impact of whether precision or recall while weakening the other one.

Out of both formulas introduced for F-Measure, the second latter is more common for the NER evaluation systems. Consequently β is the standard control parameter for using balanced F measure. When there is no reason for maximizing the impact of precision or recall rather than the other one, balanced F1 measure is used as evaluation mean which means initializing $\beta = 1$ $\left(\text{in other words } \alpha = \frac{1}{2}\right)$, creating equal balance between precision and recall.

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

2.4 Technical aspects: Existing Technologies for NER

The theoretical aspects of Named Entity Recognition are fully covered in previous chapter. In order to make a proper decision in choosing the most correspondent package to the system requirements, having insight over all existing technologies is required. There is wide variety of software, packages and libraries available which implemented mentioned algorithms using different techniques and technologies. In this chapter all available toolkits are introduced. Furthermore, in each part their features, languages and licences are summarized. The list is sorted by name (A_Z).

- **AlchemyAPI** [4]

  Type: Web-service

  Programming Language: Multiple

  Services: Named entity recognition, language identification, concept tagging, content scraping, web page cleaning, text classification, keyword extraction.

  License: Commercial, offered in free, basic, professional and metered versions.

[4] http://www.alchemyapi.com/

- **Apache Mahout** [5]

  Type: Library

  Programming Language: Java

  Features: User and item based recommendation, mean shift and fuzzy k-means clustering, singular value decomposition, collaborative filtering, random forest decision tree based and complementary naive bayes classifiers, latent Dirichlet process allocation and parallel frequent pattern mining.

  License: Apache software license

- **Balie** [6]

  Type: Library

  Programming Language: Java

  Features: Tokenization, language identification, named entity recognition, sentence boundary detection, supporting English, French, Spanish, German and Romanian.

  License: GNU GPL

- **Classifier4J** [7]

  Type: Library

  Language: Java

  Features: Text summary, vector and Bayesian text classification

  License: Apache software license

- **Content Analyst** [8]

  Type: Platform

  Language: N/A

  Features: Automatic summarization, vector and Bayesian text classification

  License: Apache software license

- **The Dragon Toolkit** [9]

  Type: Development package

  Language: Java

  Features: Text summarization, text classification, topic modelling and text clustering

  License: Open source (Few condition should be met which is stated in their website)

---

[5] Apache Mahout-scalable machine learning algorithm.
[6] http://balie.sourceforge.net/
[7] http://classifier4j.sourceforge.net/
[8] http://contentanalyst.com/html/tech/technologies.html
[9] http://dragon.ischool.drexel.edu/license.asp

- **FreeLing** [10]

  Type: Library

  Language: C++

  Features: Sentence splitting, sSuffix treatment, text tokenization, morphological analysis, retokenization of clitic pronouns, Rule-based dependency parsing, nominal correference resolution, contraction splitting, named entity recognition, PoS tagging, probabilistic prediction of unknown word categories, chart-based shallow parsing and WordNet based sense annotation and disambiguation, supporting English, Austrian, Portuguese, Spanish, Galician, Italian, Catalan and Welsh.

  License: GPL GNU

- **Gate** [11]

  Type: Framework

  Language: Java

  Features: Text processing features, using of external plugins.

  License: Apache software license

- **Illinois Natural Language Processing Group** [12]

  Type: Tools/Packages

  Language: Java

  Features: NLP curator, quantifier, chunker, named entity tagger, part of speech tagger, semantic role labeller (SLR), lemmatizer and lots of more packages.

  License: BSD license

- **Java Text Mining Toolkit (JTMT)** [13]

  Type: Toolkit

  Language: Java

  Features: Citation based ranking, POS Tagger and recognizer, token recognition, binary naïve and vector space classifier, corrector using word collocation probabilities and summarization with Lucene.

  License: LGPL (Lesser GNU General Public License)

- **Julie NLP** [14]

[10] http://garraf.epsevg.upc.es/freeling/demo.php
[11] https://gate.ac.uk/
[12] http://nlp.cs.illinois.edu/
[13] http://jtmt.sourceforge.net/
[14] http://www.julielab.de/Resources/NLP+Tools.html

Type: Toolkit

Language: Java

Features: Named entity recognition, semantic search, text mining and information extraction.

License: Common Public License

- **Language Computer** [15]

  Type: Products

  Language: N/A

  Features: Named entity recognition, PoS tagging, attribute/event/relationship extraction, co-reference resolution, text summarization, sentence splitting and tokenization.

  License: Fully commercial

- **Lemur Project** [16]

  Type: Toolkit

  Language: C++, C# and Java

  Features: Set of toolkits for information retrieval and text mining such as search engine (Indri), browser toolbar (Lemur) and data resource (ClueWeb09)

  License: BSD license

- **Lingo3G** [17]

  Type: Software

  Language: Java, PHP, Ruby and C#

  Features: Clustering text collections in hierarchy manner

  License: Commercial (but also offering some open source alternatives)

- **LingPipe** [18]

  Type: Toolkit

  Language: Java

  Features: Named entity recognition, language identification, sentiment analysis, clustering, hyphenation and syllabication, Sentence detection, word sense disambiguation, spelling correction, string comparisons, interesting phrase detection, database text mining, Chinese word segmentation, character language modelling,

[15] http://www.languagecomputer.com/
[16] http://www.lemurproject.org/
[17] http://carrotsearch.com/lingo3g
[18] http://alias-i.com/lingpipe/

singular value decomposition, logistic regression, expectation maximization, topic classification, and PoS tagging.

License: Free for academic use and for other purposes commercial licenses are available

- **Mallet** [19]

  Type: Toolkit

  Language: Java

  Features: Numerical optimization, sequence tagging, text classification and topic modelling.

  License: Common Public Licence

- **Minor Third** [20]

  Type: Toolkit

  Language: Java

  Features: Named entity recognition, text annotation and classification.

  License: BSD license

- **MontyLingua** [21]

  Type: Toolkit

  Language: Java and Python

  Features: PoS tagging, natural language summarization, lemmatization and tokenization.

  License: MontyLingua version 2.0 License (Free for non-commercial applications)

- **MorphAdorner** [22]

  Type: Command line program

  Language: Java

  Features: Word tokenization, language recognition, PoS tagging, spelling standardization, text segmentation, name recognition, lemmatization, noun pluralization, verb conjugation and sentence splitting.

  License: NCSA style license

- **NaCTeM** [23]

  Type: Software tools

[19] http://mallet.cs.umass.edu/
[20] http://sourceforge.net/projects/minorthird/
[21] http://web.media.mit.edu/~hugo/montylingua/
[22] http://morphadorner.northwestern.edu/
[23] http://www.nactem.ac.uk/

Language: N/A

Features: Named entity recognition, PoS tagging, text classification, deep syntactic parsing, sentiment analysis, annotation of named entities, shallow parsing for bio-medical text and a bilingual dictionary extraction using a Random Forest method.

License: Free and closed source

- **NLTK** [24]

  Type: Toolkit

  Language: Python

  Features: Text classification, PoS tagging, syntactic parsing, stemming and text to-kenization.

  License: Apache 2.0 license

- **Open Calais** [25]

  Type: Web service

  Language: N/A

  Features: Named entity recognition, fact and event extraction.

  License: Free up to 50,000 transactions a day (more transaction requires service li-cense agreement).

- **OpenNLP** [26]

  Type: Library

  Language: Java

  Features: Named entity recognition, part-of-speech tagging, tokenization, chunking, sentence segmentation, parsing and coreference resolution.

  License: Apache License Version 2.0

- **Palladian** [27]

  Type: Toolkit

  Language: Java

  Features: Named entity recognition, Sentence splitting, tokenization, classification and information retrieval such as crawler and API access.

  License: Apache License Version 2.0

- **RASP** [28]

---

[24] http://www.nltk.org/
[25] http://www.opencalais.com/
[26] http://opennlp.apache.org/
[27] http://palladian.ws/
[28] http://www.sussex.ac.uk/Users/johnca/rasp/

Type: Toolkit

Language: C and Lisp

Features: Lemmatization, morphological analysis, Tokenization, grammar-based parsing and PoS tagging.

License: GNU Lesser General Public License (LGPL)

- **Stanford NLP** [29]

  Type: Library

  Language: Java

  Features: Named entity recognition, word segmentation, PoS tagging and classification.

  License: GNU GPL

- **SRILM** [30]

  Type: Toolkit

  Language: C++

  Features: Statistical tagging and segmentation, Machine translation and speech recognition.

  License: SRILM Research Community License Version 1.1 (Only free for non-commercial use)

- **TextAnalyst** [31]

  Type: Software

  Language: N/A

  Features: Semantic information retrieval, text clustering, text summarization and meaning extraction.

  License: Commercial

- **VisualText** [32]

  Type: Software

  Language: C and Lisp

  Features: Named entity recognition, text classification, text indexing, text summarization, text grading and text filtering.

  License: Commercial

[29] http://nlp.stanford.edu/
[30] http://www.speech.sri.com/projects/srilm/
[31] http://megaputer.com/site/textanalyst.php
[32] http://www.textanalysis.com/

- **WEKA** [33]

  Type: Library

  Language: Java

  Features: text clustering, classification and regression using support vector machines, naïve bayes, k-nearest neighbour and neural networks.

  License: GNU Lesser General Public License (LGPL)

  After introducing theoretical aspects of the Natural language processing specifically Named Entity Recognition and providing latest implementations available, now it is time to elaborate the requirements of the goal system.

[33] http://www.cs.waikato.ac.nz/ml/weka/

3 Named Entity Recognition in Customer Relationship Management (CRM) Systems

The final goal of this thesis to develop and optimize a named entity recognition system inside an existing CRM system called Falcon. In first part of this chapter, details about CRM applications are discussed, followed by the application of named entity recognition in CRM systems. Falcon CRM will be introduced as well. In the last part of these chapter the requirements of Falcon for having a proper NER system is elaborated.

3.1 Introduction to CRM

CRM is abbreviation for Customer Relationship Management which refers to a set of tools for managing all the affairs related to customers, ranging from customer services and support to marketing, sales, inventory and so on. There is wide variety of CRM applications available in the market with different prices, features, specifications, services, accessibility and a lot of factors which will be offer as software packages, online web services or any other platforms. Customer relationship management gives the businesses opportunities improve their relation with existing customers as well as potential customers out of providing more satisfaction. In general, CRM applications can be categorized in three major groups below:

3.1.1 Collaborative Systems

Collaborative CRM focuses on communication and direct interaction with customers through different platforms such as emails, telephone calls, web and etc. Customers can play a vital role in the improvement of services. Collaborative CRM make it possible for the business to offer better products out of collaboration with customers.

3.1.2 Operational Systems

Operational CRM systems are used to take care of the customers. In other words, it offers supports for a variety of business processes such as marketing, managing sales and customer data such as their contacts, history of purchases, last communications and so on. All these data can be collected in one place in order to use later on for marketing purposes or other types of CRM such as Analytical.

3.1.3 Analytical Systems

Analytical systems focus on the analytical aspect of the data. It uses a variety of techniques to process the available data related to sales, customers and services in order to evaluate the business performance, gaining more insight over the past which helps businesses to modify and improve the strategies and marketing plans.

There is a wide range of CRM systems available in the market, offering services via web application, software or other types of online or even offline services. At this moment, market leaders are Microsoft Dynamic CRM, Salesforce.com, Oracle and SAP AG.

3.2 Named Entity Recognition Applications in CRM Systems

Since the number of features in CRM systems which offer services to the users are countless, it would not be pragmatic to cover all of them here in this research. However based on the types of CRM which are discussed in the previous chapter, it is possible to mention and categorize some of the features which can gain benefit out of the results of named entity recognition engines:

3.2.1 NER in Collaborative CRM

The most interesting type of CRM for taking advantage of named entity recognition features is Collaborative. Since the focus is on the communication and direct interaction with customers, a wide range of applications can be implemented through different strategies to offer better services to the customers. Nowadays most of direct interactions with the customers are being done through different online communication platforms such as emails, social media and web. All the data passing through these channels can be passed through the named entity recognition engine in order to be processed. Once the named entities are extracted, a variety of services can be offered. For instance, in the calendar of the user in a CRM application, an event can be automatically created and added to the calendar of the user using extracted entities (such as date and location) from the plain text of an email containing a request for a meeting.

3.2.2 NER in Operational CRM

In Operational CRM systems, a named entity recognition engine can play a role of fascinator in the data collection and categorization. As it is already mentioned Operational CRM systems help the organization to keep track of its business processes and collected all the data related to the fields. NER engine can help to extract key entities to have a more organized and structured data. For instance, to collect and store sales and customer data such as history of purchases, named entity recognition can extract all related entities such as numbers and date of purchases from plain texts to have an automatic data collection service.

3.2.3 NER in Analytical CRM

Due to the catachrestic of Analytical CRM systems which is the analysis of data to gain more insight about the business, named entity recognition can only be helpful in the data collection process which is exactly the same as applications which are discussed in the previous chapter. However, other Natural Language Processing fields can play important role in processing the data for having more insight over business. For instance sentiment analysis of Twits and Facebook posts by customers about an specific products can give the product owner an insight over opinion of users.

3.3 Falcon CRM

Falcon is a web application offering a variety of services. The core of the falcon is focusing on collaborative and operative CRM services rather analytical. However, system is not limited to CRM features but also a lot of extra services such as task management and project management which are under development.

Major features are Falcon CRM which is under development for the first version of the product are listed below:

1- Automating a variety of tasks for the employees such as mailings, calendar and schedule, summary of meetings and upcoming tasks.

2- Keep track of different stage of the project out of automatizing writing reports, to do lists and an online communication platform

3- Collect data from different resources to evaluate the business performance.

In the next chapter, more detailed descriptions regarding features which use Named Entity Recognition to offer services will be discussed.

3.4 Named Entity Recognition in Falcon CRM

As it is already stated, there is a wide variety of features within Falcon. Out of which, some of them take advantage of Named Entity Recognition. Namely creation of tasks, events, report and summary out of online communication such as emails, instant messages and so on which is elaborated below.

3.4.1 Task management system and calendar

There is a task management system coupled with a personalized calendar within Falcon which help users to create tasks, assign it to another or a group of users, keep track of its progress, add it to a calendar using due date and lots of more features. However, features are already implemented in some CRM and task management systems, using Named Entity Recognition to automatize these tasks is a new method which makes the features easier than ever before to use. For example, if a user receive an email, they can open it in a Falcon mail management system (similar to outlook). The falcon Named Entity Recognition engine can detect if a name of a task (e.g. asking for a meeting, call or etc.), company (Soluso B.V., Amelink Ltd. And etc.), person (David Notte, Farbod, De rooij and etc.), location (Silicon Valley, Nijmegen, Kerkenbos St. and etc.), date (30-10-2014, 16:30, 14 Jan 2004, Sat 21 Feb 2016 and etc.) is stated in the plain text of an email, report or etc. Based on the detection, it can suggest users to create tasks using template which is initialized with the specification of the task that should be done such as a title, specific assignee, due date or other elements which is stated in the text. Another example can be an email request which a user received by an email from its customer. The named entity recognition engine suggest the users to create an event in their calendar using a prefilled template with its suggestion regarding the parties involved in the meeting, date, place and other entities which might be stated in the email.

3.4.2 Template selection

There is a variety of templates available within Falcon CRM. Users access and use these forms for different purposes. For instance, a form can be created with the project inquiry template in order to provide a customer prices regarding a project, so based on the email conversation, reports or simply the title of the task which users type in the title of the form, the named entity recognizer engines will detect all entities in the plain text and based on the extracted keywords it suggests the most related templates.

For instance, a user received an email in his Falcon inbox. It is stated that "Regarding our meeting, I would like to inform the place is same as previous time (Looimolenweg 13, 6542JN). I am looking forward to see you at 16:30, 30-10-2014". The system can recognize that a place and a date is stated in the email. So it will suggest a template for adding an event to the user's calendar. The system will initialize:

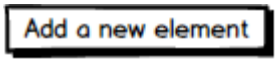
- The time of the event with "16:30, 30-10-2014"
- The title of the event with "Meeting with x" where "x" can be extracted from the keywords in the email such as the name of the person in the email signature:

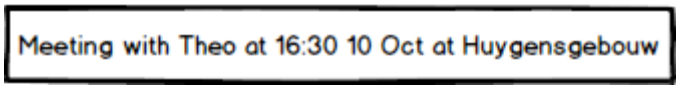
  Regards,
  Farbod Saraf,

  Information Science (M.Sc), Radboud Universiteit Nijmegen
  m:+31 6 27350606 | a: Looimolenweg 13 6542 JN Nijmegen Nederland
  LinkedIn profile

- The location of the email with Looimolenweg 13, 6542JN either from the email text, the mail signature or even out of referring to the contact manager to find the address which is allocated to the person or company.

A similar example can be the same process via adding an event manually. In other words, a user make an appointment with someone on the phone and now he/she wants to add this event to the calendar. Simply the user can click on adding a new element and start typing the title for the element:





Automatically Falcon detects the Entities namely "Theo" as a person, "16:30 10-10-14" as a date and "Huyegensgebouw" as a location. Therefore the system suggest a user a template for creating an event in the calendar, initialized with mentioned entities in their right places.

Lots of example are available to describe the role of named entity recognition in facilitating the manual tasks in Falcon CRM such as events and tasks creation, choosing and filling proper templates in terms of documentation as well as communication and so on. In the upcoming chapter, technical requirements of the system is analysed.

4 Requirements Analysis

In previous chapter, high level functionalities of the Falcon using named entity recognition were discussed. However, in order to have a fully comprehensive solution, the requirements of Falcon should be examine in the detail, regarding both aspect of functional and technical. The former stands for what the system is supposed to do while the latter specify how the system is built.

4.1 Technical Requirements

At the moment of writing this thesis, Falcon CRM is under development in terms of both architecture and implementation aspects. Therefore, to make technical requirements clear, a solution is to consider the named entity recognition engine as a separate element which will be added to the system in order to conduct tasks and prepare the results for other elements to finalize a specific goal (e.g. form template suggestions, element creation and etc.). As it is already stated in previous chapter, there is a wide range of advantages which can be drawn from named entity recognition in the Falcon CRM system which is not possible to discuss all here in this thesis. However, based on brainstorming sessions and Falcon requirements, the type of named entities which are supposed to be extracted were clarified. In upcoming sub-chapters, type of the platform and its programming language, license requirements, inputs of the NER engine, output (targeted types of entities) and their specification such as desire data format will be elaborated. All these factors have impacts on the selection of the NER library, package or software. At the end, the selection and preparation of the package will introduced.

4.1.1 Platform Specification

Referring to Falcon CRM design documents, all elements should be implemented using MVC language pattern specifically C# in .Net framework platform. Therefore the preference for developing an NER engine is to have C# friendly product. It other words, If the selected product is:

- A closed source software/toolkit: There is no requirements for its language since the product can react as the translator. That is, receiving input data, enter it to the package, processing and producing the output, finalizing it via passing back the results to Falcon.
- An open source package/library: A major concrete requirement for a modifiable package or library is to be adoptable to the Falcon which has been written in C#. In other words, the language of the library should either have been written in C# or in a way that is portable to use in C# programs.

4.1.2 License Requirements

There must be no costs involved in either using a package or buying a subscription. Therefore, the modified named entity recognition engine of the Falcon CRM should be freely available without any restriction in using it commercially.

4.1.3 System Input Specification

The input of the named entity recognition engine in Falcon should be a plain text. The input text stream should be consider as raw data which means either might contain entities or not.

4.1.4 System Output Specification

Base on the goal of the systems, system owners and designers came to a conclusion that the output of the NER engine in Falcon CRM should be return specific entity types of "Person's Name", "Company's Name", "Location", "Time" and "Date". The results must be a plain text as well containing all detected named entities. However, at this moment there is no requirements on the format of the results. It can be offered as CSV files or any other formats available for plain-text.

4.2 Functional Requirements

4.2.1 Algorithm selection

The designers of the Falcon has not imposed any restrictions in terms of algorithms selection. Therefore, the most efficient algorithms which are implemented in the latest products will be chosen for the Falcon NER engine, mainly CRF (Conditional Random Field), Maxent (Maximum Entropy), MEMM (Maximum Entropy), HMM (Hidden Markov Model) and decision tree.

4.2.2 License Filtrations

In previous chapter it is mentioned that a product should be embedded to Falcon which has not been restricted for using in commercial software. Therefore all the commercial products will should be eliminated during the selection process. In second step, it comes to open source application. Although there are different licenses available which can be considered as open source, few of them give permission for using them in commercial applications. For example GNU GPL is one of most common license in the world of free software. However, if developers wants to use a code which is distributed under this license in their own product, it should be distribute under GPL as well. In other words, in commercial closed source application such as Falcon it is not possible to use full GPL licensed code. So all the products under this license will be eliminated during the selection procedure. Here is the example directly from the website of Stanford Natural Language Processing Group:

"All these software distributions are open source, licensed under the GNU General Public License (v2 or later). Note that this is the full GPL, which allows many free uses, but does not allow its incorporation into any type of distributed proprietary software, even in part or in translation. Commercial licensing is also available; please contact us if you are interested."[34]

[34] http://nlp.stanford.edu/software/

Another common license is GNU LGPL which is similar to the GNU GPL with less restriction for developers who are using the product. In other words, GNU LGPL does not require the developers to license their own products under the same type. BSD license is another type which impose less limitation on redistribution. BSD distribution is allowed to be modified and used in the close source commercial applications. The most permissive license is MIT which allows users to freely use, copy, and modify. The only minor restriction is that it should be accompanied by the license agreement. Apache is another famous license which is applicable to both copyrights and patents. Regarding copyright which is related to this project, Apache grants user several rights which makes it prominent. Refer to the official website of Apache:

"**Grant of Copyright License**. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form." [35]

Based on what is elaborated regarding licenses, products under apache licence suit Falcon requirements. There are more functional requirements which will be specified after the product selection such as training classifier, porting other languages to c# and so on. In the next chapter one product will be selected to be modified in the solution chapter.

4.3 Product selection and preparation

4.3.1 OpenNLP vs. Stanford NLP

All packages and tools which were introduced in state of the art chapter were examined carefully. First step was eliminating all the commercial and closed source packages from the long list. Then the filtration were done based on the features of the packages. In other words, packages without named entity recognition feature or weak NER engine were filtered from the list. At third step, packages which are neither in C# nor capable of being ported to C# were removed. At the last step, two packages were nominated as the most suitable options with regards to the requirements of Falcon CRM. Both packages have proper documentation and functionalities in terms of named entity recognition. Stanford NLP group used CRF (Conditional Random Field) coupled with some other methods to implement named entity recognition feature. On the other side, OpenNLP group implemented NER engine using Maximum Entropy Model. There was a dilemma regarding the selection of a package out of this two which was solved by the license restriction. Stanford NLP package is licensed under the GNU GPL and it is clearly stated in their website that it is not allowed to use this package in any type of distributed proprietary software. There are some possibilities for using the package in the commercial extent, however it requires some condition from Stanford NLP group. On the other hand,

[35] http://www.apache.org/licenses/LICENSE-2.0

OpenNLP is licensed under Apache version 2.0 which impose no restrictions on modifying and using the package in the commercial applications. Therefore, at the end OpenNLP is selected out of all packages introduced in the stated of the art chapter.

4.3.2 Porting Java to C#

As it is already stated, OpenNLP libraries has been written in Java. In order to use them in Falcon two solutions are available. First one is to make a separate system for Named Entity Recognition and pass the data through a data channel from Falcon to NER system which returns the result to Falcon. This black box which reacts as translator might cause unacceptable amount of latency which affects the performance of real time suggestions by Falcon CRM. The second solution which is selected to implement in this thesis is to port the Java code to C#.

4.3.2.1 IKVM

IKVM is a free software for converting java files to a .Net assembly (dll). IKVM is distributed under a permissive free software license, therefore there is no restrictions in using it either for this thesis or the commercial application (Falcon CRM). The software makes it possible to run compiled java code of OpenNLP directly on Microsoft .Net which is the goal since Falcon is being written in C#. A short guide to port OpenNLP java code to C# using installation guide provided in OpenNLP wiki[36]:

- Download and extract files from latest version package of IKVM[37]
- Download and extract the latest distribution of OpenNLP
- Copy three files from OpenNLP package namely opennlp-maxent-incubating.jar, jwnl.jar, opennlp-tools-incubating.jar[38] to bin folder of IKVM (ikvmbin-version/bin)
- Open the command prompt and changed the directory to the bin folder of IKVM
- Run this command: ikvmc -target:library -assembly:opennlp opennlp-maxent-incubating.jar jwnl.jar opennlp-tools-incubating.jar.
- Running the previous command leads to the creation of dll files which can be used in .Net project. Copy the created files which is listed below to the folder of the project:
    - opennlp.dll (the assembly you have just created)
    - IKVM.Runtime.dll
    - IKVM.OpenJDK.Core.dll
    - IKVM.OpenJDK.Jdbc.dll

[36] https://cwiki.apache.org/confluence/display/OPENNLP/
[37] http://sourceforge.net/projects/ikvm/files/
[38] These names are mostly coupled with numbers indicating versions of the distribution.

- IKVM.OpenJDK.Text.dll
- IKVM.OpenJDK.Util.dll
- IKVM.OpenJDK.XML.API.dll

Now OpenNLP is ready to use in a .Net project using the references to these assemblies. In the Appendix, the ported code using IKVM is provided.

5 Recognition of Named Entities

As it is stated in previous chapter, OpenNLP is selected to fulfil the requirements of the case. In the state of the art chapter OpenNLP is described shortly. However, since the package is going to be modified, more insight over the whole product is needed. Therefore, in the next subchapter OpenNLP is elaborated, followed by the introduction to named entity recognition of the package, preparing the training data, training of a classifier and the evaluation is described. In the end, a new method for boosting the results is described, followed by the evaluation and comparison of the results in terms of accuracy.

5.1 Introduction to OpenNLP

OpenNLP is a natural language processing library written in Java and distributed under Apache version 2.0 license. It offers solution for several NLP tasks which are accessible either via command line or API including:

- **Part-of-speech tagging:**

  POS tagger assign each token a label which corresponds with its appropriate word type, using the word itself or its context.

- **Chunking**

  OpenNLP chunker divide a sentence into the groups which share the same syntactic types such as noun group, verb group and so on.

- **Tokenization**

  It segment an input text into tokens such as numbers, words, punctuation and so on. The output of this feature is vital for some other tasks such as named entity recognition.

- **Parsing**

  Sentence segmentation: This feature contribute to sentence detection. In other words, it can determine if a punctuation character in the sentence represent the end of sentence.

- **Named entity recognition:**

  Named entity recognizer is capable of detecting named entities such as persons, date, locations and so on. In next chapter details regarding this procedure are described.

- **Coreference resolution**

  It contributes to finding all tokens refers to the same entity. This feature of OpenNLP is not fully grown as other tasks, however they already made some progress in this area.

In addition, OpenNLP offers several pre-built models for different languages using machine learning techniques such as Maxent and perception. It also provides the sources for annotated texts which were used to train models. The clear and structured documentation is provided in their website[39], coupled with comments in the raw codes. For each feature, a guide through evaluation process is described as well.

5.2 Named Entity Recognizer

As it is already mentioned, OpenNLP offers named entity recognition to find entities in the raw input corpus. There is two way for using name entity recognizer in OpenNLP, a command line interface and API. For name entity recognition, there are few pre-trained models available. However to have better performance, training your own model on proper corpus is required.

5.2.1 Training Data Specification

To train a model, annotated corpus is required which should be in a specific format below:

- There should be one sentence per line.
- The entities in the sentences should be tokenized and labelled using spans.
- The separation of the documents are done using empty lines.
- The minimum amount of sentences in the training data to train a high performance model is 20000.

Here is an example of the training data:

On <START:date>October 17<END>, during an academic ceremony at the Stevenskerk in <START:location>Nijmegen, the Rectorship will be transferred from <START:person>Bas Kortmann<END>, the Rector for the last seven years, to <START:person>Theo Engelen<END>.

[39] http://opennlp.apache.org/documentation/

<START:person>Kortmann<END> is looking forward to taking up his professorship again.

Meanwhile, the newest product of <START:organization>Soluso B.V<END> will be available online on the website next week on <START:date>Friday 30th Aug<END>.

5.2.2 Training a classifier

For training a classifier in OpenNLP the tree major steps are required:

1. Opening a training data stream

```
Charset charset = Charset.forName("UTF-8");
ObjectStream<String> lineStream =
          new PlainTextByLineStream(new FileInputStream("en-ner-
person.train"), charset);
ObjectStream<NameSample> sampleStream = new
NameSampleDataStream(lineStream);
```

2. Call the NameFinderME

```
TokenNameFinderModel model;

try {
  model = NameFinderME.train("en", "person", sampleStream,
TrainingParameters.defaultParams(),
            null, Collections.<String, Object>emptyMap());
}
finally {
  sampleStream.close();
}
```

3. Saving the Model to a file/DB

```
try {
  modelOut = new BufferedOutputStream(new FileOutputStream(mod-
elFile));
  model.serialize(modelOut);
} finally {
  if (modelOut != null)
     modelOut.close();
}
```

Since no feature generation is specified, OpenNLP uses its default feature generation.

5.2.3 Custom Feature Generation Specification

In order to specify a custom feature generation, either the implementation of an interface called AdaptiveFeatureGenerator or the extension of the FeatureGeneratorAdapter is required. Here is the example from OpenNLP which shows the specification of its default feature generator:

```
AdaptiveFeatureGenerator featureGenerator = new CachedFeatureGenera-
tor(
         new AdaptiveFeatureGenerator[]{
           new WindowFeatureGenerator(new TokenFeatureGenerator(), 2,
2),
           new WindowFeatureGenerator(new TokenClassFeatureGenera-
tor(true), 2, 2),
           new OutcomePriorFeatureGenerator(),
           new PreviousMapFeatureGenerator(),
           new BigramNameFeatureGenerator(),
           new SentenceFeatureGenerator(true, false)
           });
```

The method which must be used for training is mentioned below. Feature generator can be an argument for this function:

```
public static TokenNameFinderModel train(String languageCode, String
type, ObjectStream<NameSample> samples,
       TrainingParameters trainParams, AdaptiveFeatureGenerator gen-
erator, final Map<String, Object> resources) throws IOException
```

In order to detect name entities, both model and the feature generator must be entered as arguments to the NameFinderME method:

```
new     NameFinderME(model,     featureGenerator,     NameFinderME.DE-
FAULT_BEAM_SIZE);
```

In this thesis, the default feature generator was used. However in the future, based on the new requirements in Falcon, a customized feature generation might be needed.

5.3 Named Entity Recognizer in Falcon Case

5.3.1 Training Data Preparation

The training data is collected from NLP-Geo project[40]. The data set is derived automatically from Wikipedia and WordNet. The data is also redefined in the format required

[40] https://code.google.com/p/nlp-geo/source/browse/trunk/models/en-ner-location.train?r=11

for the training, each line a sentence including tags for specific goal. In this case, the targeted entities were the name of the locations which suits the Falcon requirements as well. To finalize the preparation phase, all the named entities of locations were labelled. Now the repository of 5452 sentences per line including tags for location named entities is ready for the training level. It should be taken into account that all part of the corpus should be included into the training data and not only parts and sentences containing named entities.

```
1530  What do we call the imaginary line along the top of the <START:location> Rocky Mountains <END> ?
1531  What country 's capital is <START:location> Lagos <END> ?
1532  Where does Ray Bradbury 's Chronicles take place ?
1533  Who is the current prime minister and president of <START:location> Russia <END> ?
1534  Who established a Viking colony in <START:location> Greenland <END> about 985 ?
1535  What was the distinguishing mark on the `` Little Rascals '' dog ?
1536  What is the latitude and longitude of <START:location> El Paso<END> , <LOC>Texas <END> ?
```

Figure 7. Training Data Sample

5.3.2 Model Training

To train a model inside the application, calling API is more appropriate. In one of the upcoming chapter, the description of code ported to C# is described. However, during training, recognition and evaluation the tasks are done via command line. In the training command, the source and the output file should be specified which is en-ner-location.test for the former and en-ner-location.bin for the latter in this case.

```
C:\Users\Saraf\apache-opennlp-1.5.3-bin\apache-opennlp-1.5.3\bin>opennlp TokenNameFinderTrainer -model en-ner-location.bin -lang en -data en-ner-
location.test -encoding UTF-8
```

After calling this command, the system starts doing specific preparation for computation namely computing event counts, indexing, sorting and merging events as well as calculation for the number of event tokens, outcomes and predicates.

```
Indexing events using cutoff of 5

       Computing event counts...  done. 3758 events
       Indexing...  done.
Sorting and merging events... done. Reduced 3758 events to 2058.
Done indexing.
Incorporating indexed data for training...
done.
       Number of Event Tokens: 2058
           Number of Outcomes: 3
         Number of Predicates: 894
...done.
```

After the preparation phase, model parameter are calculated using the likelihood of 100 iteration, followed by saving the model in the specified location.

```
 96:  ... loglikelihood=-149.1767920188561      0.9874933475252794
 97:  ... loglikelihood=-148.70912267308935     0.9874933475252794
 98:  ... loglikelihood=-148.2487323449284      0.9874933475252794
 99:  ... loglikelihood=-147.79543276532002     0.9874933475252794
100:  ... loglikelihood=-147.3490425427844      0.9874933475252794
Writing name finder model ... done (0.073s)

Wrote name finder model to
path: C:\Users\Saraf\apache-opennlp-1.5.3-bin\apache-opennlp-1.5.3\bin\en-ner-location.bin
```

Now the model is ready to recognize the location named entities.

5.3.3 Entity Recognition using Trained Model

As it is already stated, there are two ways of accessing the NER engine, API as well as command line interface. To show the result, the command line interface is selected. In

order to recognize entities, the model should be called. After calling the models, the command line is ready to receive the input. Once a text is entered, a system print out the same text coupled with tags for recognized named entities. Here is an example of one sentence were inserted to the model and two named entities were recognized by it.

```
C:\Users\Saraf\apache-opennlp-1.5.3-bin\apache-opennlp-1.5.3\bin>opennlp TokenNameFinder en-ner-location.bin
Loading Token Name Finder model ... done (0.042s)
I will leave Nijmegen on Wednesday since i have a meeting in Amersfoort on Thursday early morning.
I will leave <START:location> Nijmegen <END> on Wednesday since i have a meeting in <START:location> Amersfoort <END> on Thursday early morning.
```

5.4 Evaluation

OpenNLP offers an evaluation tool for calculating the performance of named entity recognizer using three attributes including precision, recall and F-measure. The evaluation can be done out of either pre-trained model coupled using a test set or cross validation. To evaluate the model trained in previous chapter, a test dataset is provided. Using the command below the evaluation is carried out.

```
C:\Users\Saraf\apache-opennlp-1.5.3-bin\apache-opennlp-1.5.3\bin>opennlp TokenNameFinderEvaluator -model en-ner-location.bin -data en-ner-loca
n.test -encoding UTF-8
Loading Token Name Finder model ... done (0.089s)


Average: 1252.5 sent/s
Total: 501 sent
Runtime: 0.4s

Precision: 0.8461538461538461
Recall: 0.45294117647058824
F-Measure: 0.5900383141762452
```

As can be seen, the achieved numbers are 0.84 for Precision and 0.45 for Recall which were leaded to the overall of 0.59 for F-measure score. The high precision indicates that 84% of all recognized entities were actual locations while average recall shows that 45% of all the locations mentioned in the test corpus were recognized by the NER engine. In Falcon system (and most of commercial applications), the strategy is to not to offer the users a suggestion until there would be a high likelihood that the recognized entity is correct in order to not irritate users by a numerous suggestions which are not proper. In other words, the high precision and acceptable recall achieved in this project properly fulfil the Falcon requirements. However, to make this research applicable and reliable to a wide range of projects, in the next chapter an innovative way to boost the performance specifically recall is introduced.

5.5 Boosting the Performance

One of the first step in improving the result is finding out the algorithm used inside the NER engine. The OpenNLP named entity recognizer is implemented using Maximum Entropy (Maxent) algorithm. Below the details about improvement cycle is elaborated.

5.5.1 Improvement Process

Based on the characteristic of the Maximum Entropy approach which is previously elaborated in the state-of-the-art chapter, the best results are achieved when the models are trained out of running against samples of data. To implement this method, an approach is to going through steps below recursively:

1) Building a model using annotated data

2) Reading new dataset
3) Extract named entities using models
4) Go through the results and add the entities which was not recognized by the system to the list of recognized entities
5) Repeat the process from step two

The reason that cycle above is improving the performance is that each time the model will be improved base on the fact that in Maxent method the contribution weights of similar features would not overrate the final judgments.

5.5.2 Applying the Improvement Process in Falcon

In previous chapters all the steps required for recognizing named entities using OpenNLP NER Engine is described. However, to have a better insight over the improved process, the default process is depicted in the Figure 8.

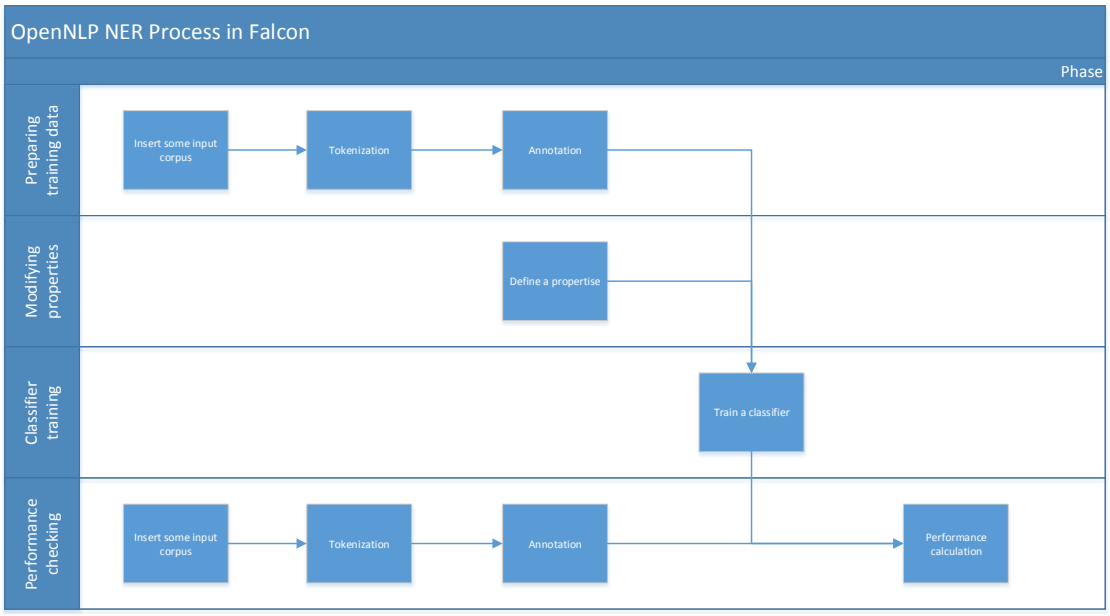


Figure 8. OpenNLP NER Process in Falcon

As can be seen named entity recognition process involves four phases, namely preparing training data, modifying properties, classifier training and performance checking. In the improvement cycle a new phase is added called customer data preparation.

To embed the improved method into the Falcon, different strategies are applicable. Out of which, the most smart one is to use the user interaction with the system to check if recognized entities are actual ones. To implement this strategy, several features of the Falcon can be involved.

For instance, when user receive an email containing a request for a meeting, the system should offer the user “adding an event to the calendar” with some pre-filled data based on the results of NER, such as the time and place of the meeting. If a user has not change the fields, it means that the recognized entities were correct. In the case that the user

changed the pre-filled data, the newly added data should be searched in the text and if it was included in the body of email, it should be tagged and added automatically to the training data for the next cycle of the improvement. This improvement cycle period is varied based on the number of the users and their interactions with the system. At this moment Falcon is in the development stage and the platform is not ready to have the full implementation of the solution. Therefore the method is implemented and tested using real freely available data on the internet. During the whole process the

Default properties is used. In the diagram below, all the phases regarding the Falcon NER improvement cycle is described

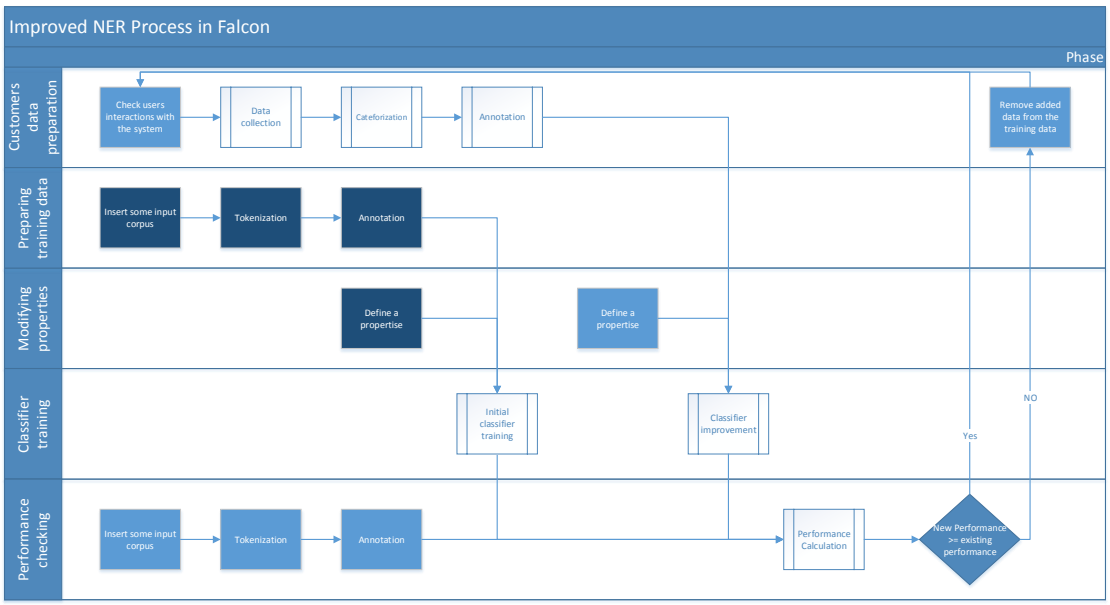


Figure 9. Improved NER Process in Falcon

5.5.3 Validation

To validate the method, an evaluation should be taken into account to compare the new result and the old method ones. To simulate the improvement cycle, firstly a model is trained using the same training data. Then an entity recognition is carried out on the newly added corpus. All entities which were neglected by the system were founded, annotated and added to the existing training data. The model is trained on the new training data and tested using the same evaluation method.

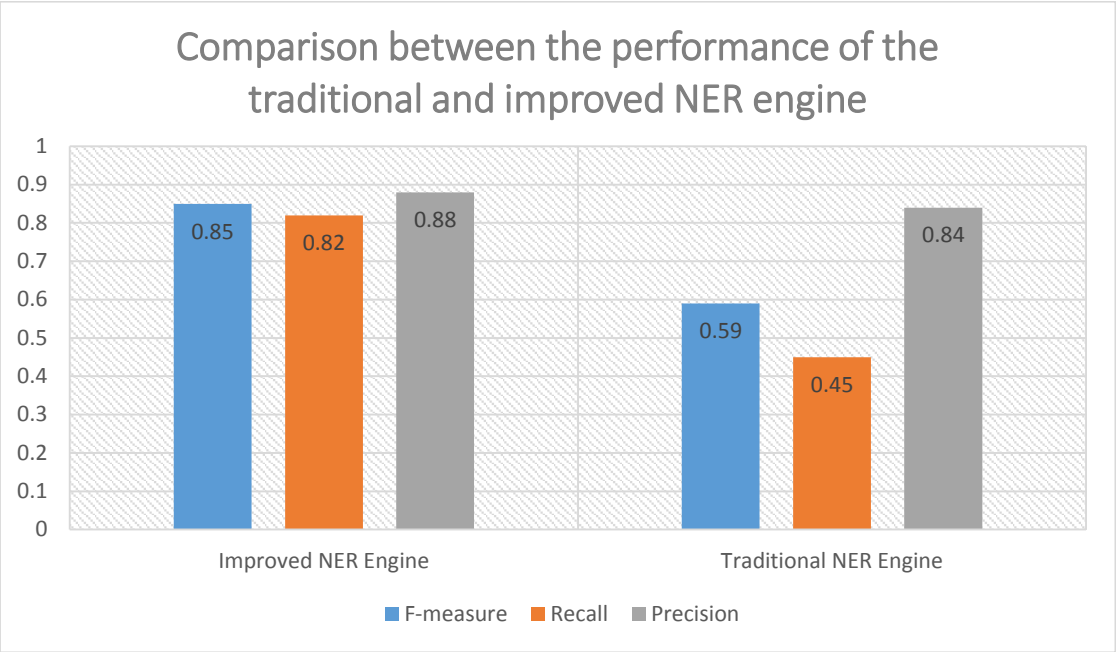
```
C:\Users\Saraf\apache-opennlp-1.5.3-bin\apache-opennlp-1.5.3\bin>opennlp TokenNameFinderEvaluator -model en-ner-location.bin -data en-ner-locatio
n.test -encoding UTF-8
Loading Token Name Finder model ... done (0.031s)


Average: 1626.6 sent/s
Total: 501 sent
Runtime: 0.308s

Precision: 0.8867924528301887
Recall: 0.8294117647058824
F-Measure: 0.8571428571428571
```

As can be seen, the recall was increased dramatically from 0.45 to 0.82 which was lead to the improvement of F-secure by 44% from 0.59 to 0.85. In addition, the precision has

also been incremented from 0.84 to 0.88. The chart below shows the comparison between the results of both methods, indicating the superiority of improved NER engine.



6 Conclusion

6.1 Summary

Named Entity Recognition is one of the natural language processing subfields. It refers to the recognition of specific entity types such as names of the person, locations and dates in the plain text. The outcome of NER systems can be used in wide range of applications to offer better services to the user. In this thesis, the focus was on the implementation of named entity recognition engine in an existing CRM application as well as boosting the results via the modification of the system. In order to have a reliable named entity recognizer, reviewing state-of-the-art of the field was a vital step, finding out what is already achieved in the field. All of the existing techniques, algorithms and technologies were reviewed and elaborated in two separate chapters. Next step was to analyse requirements to be able to choose an appropriate technique and technologies which suit all needs of the system. With regards to all the requirements OpenNLP has been nominated, an open source library which has been written in Java under the Apache license. The named entity recognition offers several natural language services, out of which the most related library of it which is related to this project is named entity recognition engine which has been implemented using Maximum Entropy probability distribution. In the solution chapters, a model is trained and the performance of system has been evaluated using three measurement factor namely Precision, Recall and F-measure. In the last part, a new method has been introduced and tested which was lead to boosting the performance up to 44%. The bootstrapping strategy to implement the improvement method using the customer data has been elaborated as well which helps the system to have an automatic increase in terms of accuracy out of the user's interactions with the system.

6.2 Validation

In the chapter 4 all the requirements of the system has been analysed. With regards to the results of analysis a technique and technology have been selected to be implemented which means the solution fulfil the all the system requirement.

In the solution chapters, a named entity recognizer engine has been implemented. A training corpus of 5400 sentences which has been annotated for location entity was used to train a model. The embedded evaluation tool of OpenNLP was used to check the accuracy of the system and the result of evaluation was 0.84 for Precision, 0.45 for Recall and 0.59 for F-Measure. Although the Recall was low compared to the high Precision of the system, but still 0.45 is acceptable number for Recall of a named entity recognizer since in CRM applications most focus is on the precision in order to not interrupt the users with numerous suggestion.

In order to generalize the solution and make it more applicable to all other problem areas, the Recall should had been improved. Using a technique which is introduced and elaborated in previous chapters, the Recall has been amazingly raised from 0.45 to 0.82, leaded to the improvement by of recall by 82%. Recall has been increased slightly from 0.84 to 0.88. The increments in both Precision and Recall was resulted in the growth of

F-Measure from 0.59 to 0.85. In other words, the overall performance of the NER engine has been enhanced by 44%.

6.3 Future Work

As it is already stated the performance has been increased using a technique with hand-annotated data while the improvement method offer an implementation technique to have an automatic improvement using bootstrapping method which requires the user's interaction data with the system. Therefore the next step is to embed the named entity recognition engine into the CRM environment to be able to implement and test the improvement method with real-time data.

Another step in future work is to solve the privacy issues in using the customer data. A separate solution is needed to be able to use the data which involves customer interaction to improve the accuracy of named entity recognition engine. There is a variety of strategies such as encryption of the data, asking customers for data access privileges or using the data in anonymous paradigm which can be chosen with regards to the requirements.

The broader aspect of the future work is related to the improvement of OpenNLP tool itself. At this moment the package is implemented using Maximum entropy probability distribution model. Embedding some other powerful algorithms such as CRF (Conditional Random Fields, the algorithm which is implemented in the engine of Stanford NLP package) might help to enhance the overall performance of the toolkit.

References

[1] Collins, Michael. 2002. Ranking Algorithms for Named–Entity Extraction: Boosting and the Voted Perceptron. In Proc. Association for Computational Linguistics.

[2] A Survey of Named Entity Recognition and Classification, D Nadeau, S Sekine - Lingvisticae Investigationes, 2007.

[3] Olena Medelyan, Eibe Frank, and Ian H. Witten. Human-competitive tagging using automatic keyphrase extraction. In Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, pages 1318-1327. Association for Computational Linguistics, 2009.

[4] David Nadeau. Supervised Named Entity Recognition: Learning to Recognize 100 Entity Types with little Supervision. PhD thesis, Ottawa-Carleton Institute, 2007.

[5] Andrew Y. Ng and Michael I. Jordan. On Discriminative vs. Generative classifiers: A comparison of logistic regression and naïve Bayes, Advances in neural information processing systems, 2002.

[6] Robert Munro. 2013. Crowdsourcing and the Crisis-affected Population. Information Retrieval 16(2):210-266

[7] Cohen, William W.; Sarawagi, S. 2004. Exploiting Dictionaries in Named Entity Extraction: Combining Semi-Markov Extraction Processes and Data Integration Methods. In Proc. Conference on Knowledge Discovery in Data.

[8] Tzong-Han Tsai, Richard; Wu S.-H.; Chou, W.-C.; Lin, Y.-C.; He, D.; Hsiang, J.; Sung, T.-Y.; Hsu, W.-L. 2006. Various Criteria in the Evaluation of Biomedical Named Entity Recognition.

[9] Marta Recasens, M. Antònia Martí, and Constantin Orasan. 2012. Annotating Near-Identity from Coreference Disagreements. In Proceedings of LREC 2012.

[10] X. Zhou, X. Zhang, and X. Hu. Dragon Toolkit: Incorporating auto-learned semantic knowledge into large-scale text retrieval and mining. 19th IEEE International Conference Citeseer, 2007.

[11] Mengqiu Wang, Wanxiang Che, and Christopher D. Manning. 2013. Joint Word Alignment and Bilingual Named Entity Recognition Using Dual Decomposition. In ACL.

[12] Sonal Gupta and Christopher D. Manning. 2014. Improved Pattern Learning for Bootstrapped Entity Extraction. In Proceedings of the Eighteenth Conference on Computational Natural Language Learning (CoNLL).

[13] Heng, Ji; Grishman, R. 2006. Data Selection in Semi-supervised Learning for Name Tagging.

[14] Wanxiang Che, Mengqiu Wang, Christopher D. Manning, and Ting Liu. 2013. Named Entity Recognition with Bilingual Constraints. In NAACL-HLT.

[15] Cucchiarelli, Alessandro; Velardi, P. 2001. Unsupervised Named Entity Recognition Using Syntactic and Semantic Contextual Evidence. Computational Linguistics 27:1.123-131, Cambridge: MIT Press.

[16] Jansche, Martin. 2002. Named Entity Extraction with Conditional Markov Models and Classifiers. In Proc. Conference on Computational Natural Language Learning.

Appendix

An introduction for porting named entity recognizer to C# is provided on the wiki of OpenNLP[41]. It has been modified and used inside a .Net application in using Microsoft Visual Studio. In this chapter the code inside two major classes namely EntityExtractor.cs and Program.cs has been provided.

```
/// Farbod Saraf Jadidian – s4346319
/// Master Thesis Project – Oct 2014

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;


namespace OpenNLPtest
{
    public class EntityExtractor
    {
        /// <summary>
        /// Extractor for the entity types available in openNLP.
        /// Copyright 2013, Don Krapohl www.augmentedintel.com
        /// This source is free for unlimited distribution and use
        /// TODO:
        ///     try/catch/exception handling
        ///     filestream closure
        ///     model training if desired
        ///     Regex or dictionary entity extraction
        ///     clean up the setting of the Name Finder model path
        /// </summary>
        /// Call syntax:  myList = ExtractEntities(myInText, EntityType.Person);

        private string sentenceModelPath = "c:\\models\\en-sent.bin";   //path
to the model for sentence detection
        private string nameFinderModelPath;                              //Name-
Finder model path for English names
        private string tokenModelPath = "c:\\models\\en-token.bin";     //model
path for English tokens
        public enum EntityType
        {
            Date = 0,
            Location,
            Money,
            Organization,
            Person,
            Time
        }

        public List<string> ExtractEntities(string inputData, EntityType target-
Type)
        {
            /*required steps to detect names are:
             * downloaded sentence, token, and name models from
http://opennlp.sourceforge.net/models-1.5/
```

---

[41] https://cwiki.apache.org/confluence/display/OPENNLP/Introduction+to+using+openNLP+in+.NET+Projects

```
             * 1. Parse the input into sentences
             * 2. Parse the sentences into tokens
             * 3. Find the entity in the tokens

            */

            //------------------Preparation -- Set Name Finder model path based
upon entity type-----------------
            switch (targetType)
            {
                case EntityType.Date:
                    nameFinderModelPath = "c:\\models\\en-ner-date.bin";
                    break;
                case EntityType.Location:
                    nameFinderModelPath = "c:\\models\\en-ner-location.bin";
                    break;
                case EntityType.Money:
                    nameFinderModelPath = "c:\\models\\en-ner-money.bin";
                    break;
                case EntityType.Organization:
                    nameFinderModelPath = "c:\\models\\en-ner-organization.bin";
                    break;
                case EntityType.Person:
                    nameFinderModelPath = "c:\\models\\en-ner-person.bin";
                    break;
                case EntityType.Time:
                    nameFinderModelPath = "c:\\models\\en-ner-time.bin";
                    break;
                default:
                    break;
            }

            //----------------- Preparation -- load models into objects---------
--------
            //initialize the sentence detector
            opennlp.tools.sentdetect.SentenceDetectorME sentenceParser = pre-
pareSentenceDetector();

            //initialize person names model
            opennlp.tools.namefind.NameFinderME nameFinder = pre-
pareNameFinder();

            //initialize the tokenizer--used to break our sentences into words
(tokens)
            opennlp.tools.tokenize.TokenizerME tokenizer = prepareTokenizer();

            //-----------------  Make sentences, then tokens, then get names---
----------------------------

            String[] sentences = sentenceParser.sentDetect(inputData); //detect
the sentences and load into sentence array of strings
            List<string> results = new List<string>();

            foreach (string sentence in sentences)
            {
                //now tokenize the input.
                //"Don Krapohl enjoys warm sunny weather" would tokenize as
                //"Don", "Krapohl", "enjoys", "warm", "sunny", "weather"
                string[] tokens = tokenizer.tokenize(sentence);

                //do the find
                opennlp.tools.util.Span[] foundNames = nameFinder.find(tokens);
```

```
                //important:  clear adaptive data in the feature generators or
the detection rate will decrease over time.
                nameFinder.clearAdaptiveData();

                results.AddRange(opennlp.tools.util.Span.spansToStrings(found-
Names, tokens).AsEnumerable());
            }

            return results;
        }

        #region private methods
        private opennlp.tools.tokenize.TokenizerME prepareTokenizer()
        {
            java.io.FileInputStream tokenInputStream = new java.io.FileIn-
putStream(tokenModelPath);     //load the token model into a stream
            opennlp.tools.tokenize.TokenizerModel tokenModel = new
opennlp.tools.tokenize.TokenizerModel(tokenInputStream); //load the token model
            return new opennlp.tools.tokenize.TokenizerME(tokenModel);  //create
the tokenizer
        }
        private opennlp.tools.sentdetect.SentenceDetectorME prepareSentenceDe-
tector()
        {
            java.io.FileInputStream sentModelStream = new java.io.FileIn-
putStream(sentenceModelPath);       //load the sentence model into a stream
            opennlp.tools.sentdetect.SentenceModel sentModel = new
opennlp.tools.sentdetect.SentenceModel(sentModelStream);// load the model
            return new opennlp.tools.sentdetect.SentenceDetectorME(sentModel);
//create sentence detector
        }
        private opennlp.tools.namefind.NameFinderME prepareNameFinder()
        {
            java.io.FileInputStream modelInputStream = new java.io.FileIn-
putStream(nameFinderModelPath); //load the name model into a stream
            opennlp.tools.namefind.TokenNameFinderModel model = new
opennlp.tools.namefind.TokenNameFinderModel(modelInputStream); //load the model
            return new opennlp.tools.namefind.NameFinderME(model);
//create the namefinder
        }
        #endregion
    }
}
```

```
using java.util;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace OpenNLPtest
{
    class Program
    {
        static void Main(string[] args)
        {
            EntityExtractor myExtractor = new EntityExtractor();
```

```
            string myInText = "Henk and Nima are going to collaborate for the
new project in San Francisco. The owner of the company, Farbod, has been prom-
ised to help Jan during the collaboration.";

            List<string> results = myExtractor.ExtractEntities(myInText, Enti-
tyExtractor.EntityType.Person);

            foreach (var item in results)
            {
                Console.WriteLine(item);
            }
           // List<string> myList1 = NaturalLanguageProcessingCSharp.EntityEx-
tractor.ExtractEntities(myInText, EntityExtractor.EntityType.Person);
        }
    }
}
```
