A Hybrid Approach to NER by Integrating Manual Rules into MEMM

Moshe Fresko
Computer Science Department,
Data-Mining Lab.
Bar-Ilan University
Ramat Gan, Israel
+972-3-5317874
Freskom1@cs.biu.ac.il

Binyamin Rozenfeld
Computer Science Department,
Data-Mining Lab.
Bar-Ilan University
Ramat Gan, Israel
+972-3-5317874
grur@clearforest.com

Ronen Feldman
Computer Science Department,
Data-Mining Lab.
Bar-Ilan University
Ramat Gan, Israel
+972-3-5317874
Feldman@cs.biu.ac.il

ABSTRACT
This paper describes a framework for defining domain specific Feature Functions in a user friendly form to be used in a Maximum Entropy Markov Model (MEMM) for the Named Entity Recognition (NER) task. Our system called MERGE allows defining general Feature Function Templates, as well as Linguistic Rules incorporated into the classifier. The simple way of translating these rules into specific feature functions are shown. We show that MERGE can perform better from both purely machine learning based systems and purely-knowledge based approaches by some small expert interaction of rule-tuning.

Categories and Subject Descriptors
Machine Learning, Named Entity Recognition, Information Extraction.

General Terms
Machine Learning, Named Entity Recognition, Information Extraction.

Keywords
Keywords are your own designated keywords.

1. INTRODUCTION
Named-Entity recognition (NER) is one of the core components in most Information Extraction and Text Mining systems. The NER task is to find all proper noun phrases (and other easily recognizable phrases) in a text and to classify them into a small predefined set of semantic categories, such as names, locations, dates, organizations, drugs, diseases, books etc. NER is essential as a preprocessing step before applying other text mining techniques – for extracting relations, building ontologies and semantic hierarchies, etc.

There are two traditional approaches to NER: knowledge-based approach and machine learning approach. Knowledge-based systems usually achieve better accuracy, but require huge amounts of skilled labor by linguists and domain experts in order to prepare and maintain the extraction knowledge. Because of this, the recent research in NER is concentrated on machine learning techniques, which only require a manually labeled training set of documents. The best published ML-based systems perform on the level of knowledge-based systems for many categories.

In this paper we present MERGE (Maximum Entropy Rule Guided Extraction) – a hybrid NER system which combines machine learning techniques, namely Maximum Entropy and manually written simple rules. MERGE benefits from both approaches and can outperform both manually written rules and standard machine learning systems. The rule language of MERGE is quite simple and the amount of necessary rule-writing is relatively small, as most of the work is done by the ML part of the system.

1.1 Related Work
Knowledge-based systems employ complicated sets of rules written by teams of linguists, computer scientists and domain experts. For comparison with our system we use the ACE-2 winner DIAL, a system developed by ClearForest Inc. DIAL is based upon a general-purpose pattern language, which allows arbitrary procedural data-processing. The system was top performing at ACE-2 after two months of manual tuning by a team of experts.

There have been several attempts to automate the rule writing process using machine learning methods ([1], [2], [3], [4], [5]). Recently, probabilistic machine learning systems became state of the art for NER ([6],[7]) and for field extraction ([8]). Most prominently, Hidden Markov Models (HMM) have been used for the information extraction task ([6], [9], [10], [11], [12]). Beside HMM, there are other systems based on SVM ([13]), Naïve Bayes ([14]), or combinations of the above ([4], [15]). As a Maximum Entropy Model, MENE ([21]) makes use of diverse knowledge sources. Recently Maximum Entropy conditional models, like Maximum Entropy Markov Models ([8]) and Conditional Random Fields ([16]) were reported to outperform the generative HMM models on several IE tasks.

Hybrid approaches, such as TEG ([17]) combine the benefits of precise manual rule writing and the generality of the machine learning approaches. TEG is based on a short manually-written set of rules that constitute a semantically oriented probabilistic context free grammar, with the probabilities learned from the training set.

Like TEG, MERGE utilizes the hybrid approach, using manual rules to improve the accuracy of a probabilistic model. However, the MERGE rules do not form a grammar like TEG rules, but act as special-purpose features, which are combined with the automatically-generated features using the Maximum Entropy principle. MERGE rules are supposed to be written after the system passed several training-and-test development cycles, in order to catch the problematic cases on which the purely probabilistic model failed. This methodology greatly reduces the amount of manual work, as only necessary rules are ever written.

The remainder of this paper is structured as follows: in section 2 we present our implementation of the Maximum Entropy Markov Model. In section 3 we describe the details of our system. Our results are presented in section 4. In section 5 we discuss the results and conclude.

2. MAXIMUM ENTROPY MODELING

A Maximum Entropy approach models a random process by making the distribution satisfy a given set of constraints, and making as few other assumptions as possible. The constraints are specified as real-valued *feature* functions over the data points. The expected value of each feature function under the ME distribution must equal the empirical expected value of function as found in the training dataset. In all other respects, the target distribution should be as uniform as possible, which means it must have the highest entropy. Those conditions completely specify the unique distribution and show a way to calculate it. For our purposes, we use ME to model the *conditional* probability distributions, which slightly differ in the way expected values are calculated ([18]).

Let X be the set of conditions, usually very big, and Y the set of possible outcomes. We assume that there is a true joint distribution $P(x, y)$, but we are interested only in modeling the conditional $P(y \mid x)$. For this purpose we can use a training set $\{(x_k, y_k)\}_{k=1..N}$ generated by the true distribution, and a set of features $f_i : X \times Y \to \boldsymbol{R}$. Typically, the features are binary and test for specific conditions. For instance, in NER the set X may be the (huge) set of all possible mentions of all possible words, the set Y may be the set of all output categories, and a useful feature may test the correlation between the capitalization of a word and its being labeled as a personal name:

$$f(x,y) = \begin{cases} 1, & \text{if the word in } x \text{ is capitalized and } y = \text{Person} \\ 0, & \text{otherwise.} \end{cases}$$

It can be shown that the unique most uniform distribution that satisfies all feature constraints has the form:

$$(*)\quad p(y \mid x) = \frac{1}{Z(x)} \exp\left(\sum_i \lambda_i f_i(x,y) \right)$$

where λ_i–s are the parameters chosen to maximize the likelihood of the training data, and $Z(x)$ is a normalization constant, which ensures that for every x the sum of probabilities of all possible outcomes is 1. The most common procedure for parameter estimation is the Generalized Iterative Scaling algorithm ([19]).

2.1 Generalized Iterative Scaling

The comprehensive description of the algorithm can be found in [20] and [18].

We need a way to calculate the expected values of features according to the training set and according to a given conditional distribution. For this purpose we define *empirical distribution* $\tilde{p}(x,y)$ of the training dataset $\{(x_k, y_k)\}_{k=1..N}$ as:

$$\tilde{p}(x,y) = \begin{cases} \frac{1}{N}, & \text{if } (x,y) = (x_k, y_k) \text{ for some } k, \\ 0, & \text{otherwise.} \end{cases}$$

(We assume that all x_k are different, which is natural in modeling conditional distributions). The expected value of a feature f_i according to the empirical distribution is:

$$\tilde{E}(f_i) = \sum_{x,y} f_i(x,y)\tilde{p}(x,y) = \frac{1}{N} \sum_k f_i(x_k, y_k).$$

In order to calculate the expected value of a feature according to a given conditional distribution $p(y|x)$ we need the marginal $p(x)$. However, we do not model it, and moreover, we do not wish to sum over the whole X, which is huge and can be ill-defined. Therefore, we use $\tilde{p}(x)$ instead of $p(x)$, which is a reasonable approximation:

$$E_p(f_i) \approx \sum_{x,y} f_i(x,y) p(y \mid x) \tilde{p}(x) = \\ = \frac{1}{N} \sum_k \sum_{y \in Y} f_i(x_k, y) p(y \mid x_k).$$

Now we are sufficiently equipped to state the Generalized Iterative Scaling algorithm:

Input: Feature functions $\{f_i\}$, training set $\{(x_k, y_k)\}$.

Output: Optimal parameter values λ^*_i and optimal model p_{λ^*}.

1. Start with $\lambda = (\lambda_1, \lambda_2, \ldots) = 0$
2. Let $\Delta\lambda_i = \frac{1}{M} \log \frac{\tilde{E}(f_i)}{E_{p(\boldsymbol{\lambda})}(f_i)}$

   Update lambdas: $\lambda_i := \lambda_i + \Delta\lambda$
3. Repeat step 2 until convergence.

The expectation $E_{p(\lambda)}(f_i)$ is calculated according to the distribution (*) with the current set of parameters λ. The constant M is the sum of all feature functions for any given pair (x,y). Our simple version of the GIS algorithm works under assumption that this sum is independent of x and y. The common approach for making this assumption true is to fix an arbitrary sufficiently large M and add an auxiliary feature function

$$f_{aux}(x,y) = M - \sum_i f_i(x,y)$$

However, in our system we do not need to use such auxiliary function, because our feature set is generated in such a way that the sum of features is always constant and equals to the number of feature templates, described below.

2.2 Maximum Entropy Markov Models

Generative probabilistic models, such as HMMs and SCFGs perform sentence labeling tasks in the following way: Sentences and their labelings are assumed to be jointly distributed, with some unknown true distribution $P(X,Y)$, where X is the set of all sentences and Y the set of all labelings. Then, a model $p(X,Y)$ of the true distribution is estimated using the training data. After that, the model can be used to label a previously unseen sentence x by maximizing $p(x, y)$:

$$y := \arg\max_{y \in Y} p(y \mid x) = \arg\max_{y \in Y} p(x, y),$$

since the marginal $p(x)$ does not depend on y.

The necessity to model the joint probability places heavy restrictions upon the form of X. Typically, elements of X are sequences of tokens, elements of Y are sequences of category labels, and the probability of a sequence is a product of probabilities of its constituents. The probabilities of constituents are allowed to be conditioned upon the immediate neighbors but not upon anything else. In practice, longer-range dependencies and dependencies upon some arbitrary features are quite common, which makes conditional models attractive.

Conditional models, such as MEMMs ([8]) and CRFs ([16]), model $p(y|x)$ directly, avoiding the necessity of generating x. Instead, the models use Maximum Entropy to combine arbitrary contextual features of x into a single conditional distribution. The MEMM is a straightforward conditional modification of HMM, introduced in [8]. It is used in our MERGE system. The more complex CRFs produce improvement in the presence of *label bias* problems ([16]), but in our case label biases do not pose a critical problem, so a simpler model was chosen.

A MEMM consists of $|Y|$ conditional ME models $p_{y'}(y|x) = p(y|x,y')$, one for each y'. The model $p_{y'}(y|x)$ estimates the probability of appearance of the label y immediately after the label y' in the context x. The probability of a whole label sequence $\mathbf{y} = y_1 y_2 \ldots y_m$, given the sentence $\mathbf{x} = x_1 x_2 \ldots x_m$, is the product

$$p(\mathbf{y} \mid \mathbf{x}) = p_0(y_1 \mid x_1) \cdot \prod_{i=1}^{m-1} p_{y_i}\left(y_{i+1} \mid x_{i+1}\right)$$

Note that the conditioning elements x_i are not the words of a sentence, as in generative models. They are better thought of as *positions* in the sentence, and the feature functions can use any of their properties – current and neighbor tokens, their capitalization and morphological properties, indentation, etc.

The model $p_0(y|x)$ used at the beginning of a sentence is separate. There are a number of possible ways to implement it, and the correct one is probably domain and language-dependent: For instance (i) to build the model from only the first tokens in each training text, (ii) to use $p_{\text{None}}(y|x)$ instead, or (iii) to build another model from all the tokens in the training data. The second option is the one we adopted for our experiments.

The model $p_{y'}(y|x)$ estimates can suffer from data sparseness if the overall number of instances of y' in the training data is very small. In order to alleviate the problem we train a separate $p(y|x)$ model that does not depend on the previous tag, and interpolate between the two models. The exact value of the interpolation coefficient was experimentally found to be not significant, so λ=½ was used.

3. SYSTEM DETAILS

3.1 Preprocessing

The preprocessing of any text is done according to the external definitions. First, the text is divided into tokens. A token is defined by a regular expression in which for an English text domain it might be in the form:

Token = "[A-Za-z]+|[0-9]+|\S"

Which can be read as: A token is an adjacent English letter sequence or adjacent Digit sequence or any single non-white-space character.

Then, each token is assigned its features according to the feature templates. A template consists of a context rule – a binary test upon the positions in the text. It can test the exact character value of the current token and its neighbors, capitalization information, membership in an external word list, arbitrary regular expressions, externally supplied features like pos tags, etc. A template can also define a generic context rule by specifying a generic test, such as "token value". Such a generic context rule is instantiated during training, according to the token values found in the training set.

Each template defines a set of features – one feature for every combination of an instantiated context rule and a category. The features are always built in such a way that exactly one feature from a template tests true at any text position. This implies that the sum of all features is constant, satisfying the requirements of the GIS algorithm.

For example, one Feature Template may have the following form:

```
FeatureTemplate : {
        Check: -1, Word
        Check:  0, Capitalization
}
```

This template would generate a feature function for every combination of the value "Word" feature at the previous token, the value of "Capitalization" feature at the current token, and the current tag. One such feature function might be:

```
f = {   1       if previous token value is "in"
                and current token is Capitalized
                and current tag is Location
        0       otherwise }
```

3.2 Tagging

The labels are assigned at the token level – each word in a sentence is labeled by an entity type label or by label *None*. For multi-token entities we label each token by the same category label, making no distinction between beginning, middle, or ending tokens of the entity. This practice produces errors in the cases where two entities of the same category are adjacent. However, in our datasets there are less than 0.1% of such cases, which is negligible.

3.3 Integrating Manual Rules

In order to show the benefits of our manual rules, we compare them with typical knowledge-based system rules. A DIAL rulebook consists of a set of sequentially checked rules, prepared for a specific domain by a linguist-programmer-domain expert. Aside from the sheer amount of the necessary rules, there are other problems, which make the effort of creating and maintaining the rules enormous. Consider the following example:

The texts "Smith said …" and "Turner said …" may imply a rule "Any Capital word followed by said, is a Person" may yield incorrect result for the text "Microsoft said …". Another rule "If the word is Microsoft, then it is a Company" can fix that mistake, but the correcting rule must be located before the erroneous rule. Also, such rule introduces new mistakes: when there is a new text in the form "Microsoft Windows …" it is talking about a product name and not the Company.

Two problems exist in the example above. First, there is no natural order in the rules, so it is hard for a person to order them correctly. Second, it is even harder to foresee the possible undesirable consequences of a new rule in most unlikely places.

The rules of MERGE are free from those problems. Most of the "regular" patterns get caught by the automatically generated features, so only rules for problematic cases are necessary. This keeps the size of the rule-sets small and manageable. The order of rules is not important, and undesirable consequences of rules are impossible, since the bad rules are automatically rejected by the Maximum Entropy learner.

3.4 MERGE Rule Definition

Defining specific rules is done via a simple pattern matching language, with patterns working at the token level. A pattern syntax is similar to the regular expressions syntax, but with tokens instead of characters. Quantifiers *, +, ? are allowed, as well as the grouping parentheses "(" ")". The angular brackets "<>" delimit the target entity. Features are generated for each token in the delimited entity. The tokens are either specified directly, or represented by token-classes of the form "[Boolean expression]".The expressions are simple token-attribute=value checks, combined with logical operators & (and) | (or), and ! (not). The currently allowed token-attributes are:

- "cl" – character-contents of the token. Possible values are Capital, Lower, Number, etc.
- "wc" – checks the appearance of the token in a list of words. Value specifies the name of the list.
- "adj" – adjacency information with values True, False and Start-Sentence.
- Token, Word, Stem – the exact character sequence (token value), the lowercase token value, the stem of the token.

Here is an example set of rules for catching some tricky organizations:

```
Rule: Organization {
        // to catch: France's Org
: [wc=Country&cl=Capital] "'s" < [cl=Capital] >
        // to catch: Person of (the) Organization
: [cl=Capital] "of" "the"? < [cl=Capital]{1,3} >
        // to catch: company/firm/group called Xyz
: [wc=CompanyAlias] "called" < [cl=Capital]+ >
}
WordClass: Country { france turkey israel … }
WordClass: CompanyAlias { company firm group …
}
```

In our MUC-7 evaluation the list of published Word Classes are used as feature functions themselves. Besides that, some intuitive Word Classes are defined within the Rule-Development framework described in Section 3.5, in order to use them as "wc" feature for Rule Writing. As an example we had a Word Class wc=WeekDay, including the words Sunday, Monday, etc. and it is used in a Rule:

```
Rule: Date {
        : < ["next"|"last"] > [ wc=WeekDay]
}
WordClass: WeekDay { sunday monday tuesday … }
```

The translation of Rules into a Feature Function is done by checking the rule pattern at each token position. The above rule for Date will be translated into feature function:

f = { 1 if current token is "next" or "last",
and the next token is found in wc WeekDay ,
and current Tag is Date,
0 otherwise }

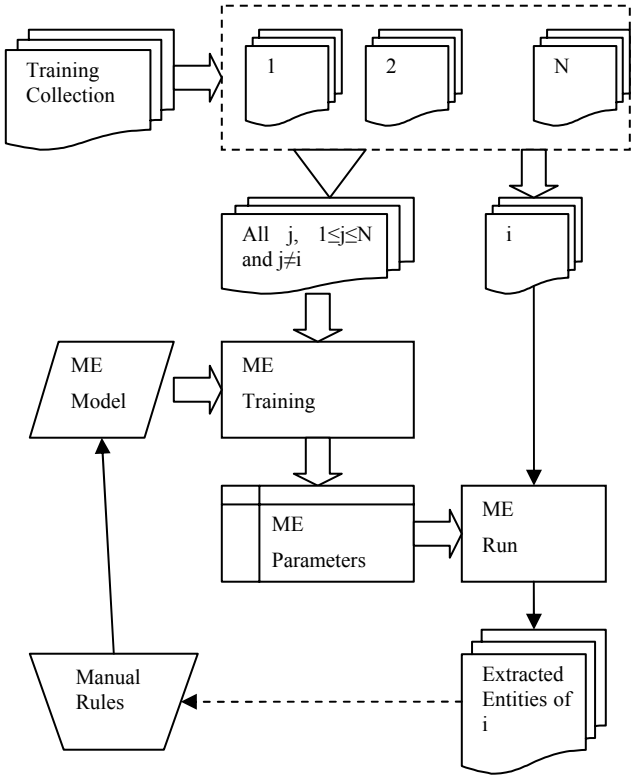

Fig. 1. Methodology for Adding Manual Rules

For keeping the value M constant for each pair (x,y), we defined also this same feature function for all the other possible tags. These definitions will not be necessary for a ME system with the auxiliary feature function as described in section 2.1 or for ME system that doesn't use the Iterative Scaling for estimating the parameters.

3.5 Methodology

The MERGE system can reach high-quality performance autonomously. We want to further improve this performance by adding rules for the more difficult cases, which are incorrectly handled by the automatic processing. Thus, our process of rules development has the following form:

Input: Manually-tagged document collection.
Step 1. Randomly divide the corpus into *N* parts.
Step 2. Take one of these parts as the Validation set, and the other *N*-1 as the Training set.
Step 3. Train the initial MERGE Model.
Step 4. Run this model on the Validation set.
Step 5. Check the missed and incorrectly tagged entities in the Validation set, and add intuitive rules for those tokens.
Step 6. Repeat Steps 3-5 until no further improvement is possible.
Step 7. Repeat Steps 2-5 with another part as the Validation set.
Step 8. Train the complete system on the whole Training data.

The most critical step is Step 5. In our experiments we extensively used Perl scripts for investigating the mis-tagged examples from the Validation data, in order to find appropriate rules for fixing them. There are separate scripts for the Current, Preceding and Succeeding contextual clues of the mis-tagged entities.

3.6 Real Example of Rule Writing

We will demonstrate the process of the rule writing and an effect of the rules with a real example.

First, we trained the system autonomously upon a subset of the MUC-7 training set. The feature templates were allowed to test a single token in a window of five (two previous tokens, the current, and two following tokens). The allowed tests were the exact token value (generic) and the character traits of the token (capitalized, numerical, punctuation, etc.). After evaluation upon a validation set we discovered many missing organizations of the form:

secretary of **defense**
vice president of **Avitas**
president of the **NFL**

We defined a Word Class of "Position" and we added a rule to deal with such cases:

```
Rule: Organization {
        : [wc=Position]+ "of" "the"? <>
}
WordClass: Position
{ secretary "vice president" president … }
```

The rule solved the problems. For comparison we tested several models with much bigger number of feature templates that were checking bigrams and trigrams in the preceding tokens. The bigrams were insufficient, and only trigrams did succeed in catching some of the errors. And one of the cases appears to require at least a 4-gram. Yet, the trigrams and 4-grams are infeasible because of the huge number of features they create, most of which are just noise.

3.7 Rule to Feature Function Translation

A rule translates into a feature function as follows: Each line in the rule definition is tested at every token position in the training and test data. Tokens that matched *i*-th pattern in the rule definition have their feature value set to *i*. Tokens that matched no expression receive zero for the feature value.

For instance, the rule definition for finding Dates may look as follows:

```
Rule: PossibleDate
{
      // Rule_Check_1
: < > [wc=NumberWord|cl=Number]
      ["days"|"weeks""months"|"years"|"decades"|"centuries"]
      ["later"|"before"|"earlier"]
      // Rule_Check_2
: < > ([wc=MonthAbbr&cl=Capital] ["."& adj=True]|
        [wc=MonthName&cl=Capital])
       wc=DayOfMonth "," cl=YearFour
      // Rule_Check_3
: < > wc=MonthNumber                [adj=True&"-"]
      [adj=True& wc=DayOfMonth] [adj=True&"-"]
      [adj=True&[cl=YearTwo|cl=YearFour]]
}
```

The value of the feature *PossibleDate* will be equal to 1 for tokens matching Rule_Check_1, such as "19 years later", to 2 for tokens matching Rule_Check_2, such as "Sept. 8, 1994", to 3 for tokens matching Rule_Check_3, such as "05-22-96", and to 0 for tokens that match neither of the expressions.

The value of the *PossibleDate* feature can subsequently be incorporated into a Feature Template Definition, for example as follows:

```
FeatureTemplate: {
        Check: -1, Word
        Check: 0 , PossibleDate
}
```

4. EXPERIMENTS

4.1 Evaluation on MUC-7

We made an evaluation of our methodology on MUC-7 data set. The 100 Training documents were used for developing the MERGE Model and the Test (Dry Run) documents were used for evaluation of the system. Only the training documents were used for discovering and writing rules, which was done according to the iterative process described in section 3.5.

The initial simple model, working in an autonomous mode (without manual rules) produced 84.9% f-measure, as shown in Table 1.

	Simple MERGE			MERGE+Rules		
	Rec.	Pre.	F.	Rec.	Pre.	F.
Org.	79.4	87.8	83.4	83.9	90.9	87.2
Per.	86.7	89.9	88.3	90.6	93.4	92.0
Loc.	85.9	85.3	85.6	91.5	91.8	91.7
All	82.7	87.3	**84.9**	87.9	91.7	**89.8**

Table 1. MUC7: After training with 100 documents.

In comparison, the system with rules performed 5% f-measure better. The rule development process took approximately 10 man-hours, during which about 300 rules were added.

Besides the simple experiment of above data set we checked the system performance also on the whole possible training data (350 documents). By adding some approximately 50 more rules the overall performance reached to 93.5% as shown in Table 2, while overlapping match results come up to 95.4%. (In overlapping-match, a entity is considered correct if at least one word of it is tagged correctly) The lower overall performance is due to the entities of Date, Time, and Money which had slightly lower results.

	Exact Match			OverlappingMatch		
	Rec.	Pre.	F.	Rec.	Pre.	F.
Org.	92.7	96.4	94.5	94.3	98.3	96.3
Per.	95.1	95.4	95.2	95.1	95.4	95.2
Loc.	92.3	98.4	95.2	92.9	99.0	95.9
All	91.2	95.9	**93.5**	93.0	97.8	**95.4**

Table 2. MUC7: After training with 350 documents.

We also compared our system to other successful models. We run an implementation of Nymble ([9]) and the TEG system ([17]) on that same corpus, which produced the results shown on Table.3. The results are slightly different from the previous ones, since the Header-Footer parts of the documents are dropped to make a fair comparison. MERGE can use any external information like which section we are in, but not the HMM system.

	HMM	TEG	Merge+Rules
Org.	87.8	90.9	93.6
Per.	80.5	91.8	93.7
Loc.	90.9	91.9	95.4
Avg.	86.4	91.5	**94.2**

Table 3. Comparetive Results for MUC-7 (without document Headers-Footers).

4.2 Evaluation Using a Large Training Dataset

We also evaluated the performance of MERGE on a large industrial corpus. The collection consists of 1170 financial news articles manually tagged with several categories. Altogether it includes 800K tokens, and around 39000 manually tagged Locations, Organizations and Persons.

The initial MERGE model included these feature function checks: the token value checks in a window size of two previous and two next tokens; character traits feature checks for previous, current and next tokens; combinations of the token value check for the current token and character traits checks for the previous and the next tokens; the global information feature checking whether the token was ever assigned a certain tag (as described in [7]); and a set of external word lists, which were also used by the TEG system and the DIAL rules.

The comparative results between HMM (Nymble), TEG, DIAL, and our MERGE system are shown below:

	HMM	TEG	DIAL	Merge	Merge Rules
Loc.	81.8	92.8	93.6	90.4	93.7
Org.	78.9	86.7	84.1	90.1	90.2
Per.	80.7	87.8	94.2	92.6	92.8
Avg.	80.5	89.1	90.6	**91.0**	**92.2**

Table 4. Comparetive Results on a Large Data-Set (F-Measure with β=1.0)

As can be seen when using a large collection for training, the MERGE's performance is fine even without any additional rules. During rule development, another 36 rules were added, which further improved the performance. Among these problematic cases were the manual tagging errors in test data, inconsistent labeling between train and test data, and all the hardly-to-find cases.

5. DISCUSSION AND CONCLUSION

We presented a hybrid NER system, which by combining the probabilistic machine learning with a set of manually written rules developed in relatively very short time period, is able to get the best of the two worlds. Our MERGE system gives better results than both purely knowledge-based and purely-ME systems while requiring only a limited amount of manual rule writing, necessary to catch the patterns that are too complex for automatic learning. The feature functions are generated by template definitions, where the training data fills the necessary feature check slots. The manually written rules are incorporated into those templates, letting human heuristic be used together with the automatic learning system.

6. REFERENCES

[1] Kushmerick, N.: Finite-state approaches to Web information extraction. 3rd Summer Convention on Information Extraction, Rome (2002)

[2] Freitag, D.: Using grammatical inference to improve precision in information extraction. Workshop on Grammatical Inference, Automata Induction, and Language Acquisition (ICML'97), Nashville, TN (1997)

[3] Aitken, J. S.: Learning Information Extraction Rules: An Inductive Logic Programming approach. 15th European Conference on Artificial Intelligence. IOS Press. (2002)

[4] Freitag, D.: Information Extraction from HTML: Application of a General Machine Learning Approach. AAAI/IAAI (1998) 517-523

[5] Grieser G, et al: A Unifying Approach to HTML Wrapper Representation and Learning. Discovery Science. 3rd International Conference, Kyoto, Japan, Proceedings. Springer, Berlin. (2000) 50-64

[6] Bikel, D. M., Schwartz, R., Weischedel, R. M.: An Algorithm that Learns What's in a Name. Machine Learning. (34): (1999) 211–231.

[7] Chieu, H. L., Tou Ng, H.: Named Entity Recognition: A Maximum Entropy Approach Using Global Information. Proceedings of the 17th International Conference on Computational Linguistics (2002)

[8] McCallum, A., Freitag, D., Pereira, F.: Maximum Entropy Markov Models for Information Extraction and Segmentation. Proceedings of the 17th International Conference on Machine Learning. (2000)

[9] Bikel, D. M., Miller, S., Schwartz, R., Weischedel, R.: Nymble: a high-performance learning name-finder. Proceedings of ANLP-97. (1997) 194-201.

[10] Leek, T. R.: Information extraction using hidden markov models. M.Sc.Thesis, UC San Diego. (1997)

[11] Freitag, D., McCallum, A.: Information Extraction with HMM Structures Learned by Stochastic Optimization. AAAI/IAAI. (2000) 584-589.

[12] Freitag, D., McCallum, A.: Information extraction with HMMs and shrinkage. Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction. (1999)

[13] Sun, A., et al.: Using Support Vector Machine for Terrorism Information Extraction. 1st NSF/NIJ Symposium on Intelligence and Security Informatics. (2003)

[14] De Sitter, A., Daelemans, W.: Information Extraction via Double Classification. International Workshop on Adaptive Text Extraction and Mining. Dubrovnik. (2003)

[15] Kushmerick, N., Johnston, E., McGuinness, S.: Information extraction by text classification. IJCAI-01 Workshop on Adaptive Text Extraction and Mining. Seattle, WA. (2001)

[16] Lafferty, J., McCallum, A., Pereira, F.: Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. Proc. 18th International Conf. on Machine Learning. (2001)

[17] Rosenfeld, B., Feldman, R., Fresko, M., Schler, J., Aumann, Y.: TEG - A Hybrid Approach to Information Extraction. Proc. of the 13th ACM. (2004)

[18] Berger, A., della Pietra, S., della Pietra, V.: A maximum entropy approach to natural language processing. Computational Linguistics 22(1), (2004) 39-71.

[19] Darroch, J. N., Ratcliff., D.: Generalized iterative scaling for log-linear models. Annals of Mathematical Statistics. 43(5): (1972) 1470-1480.

[20] della Pietra, S., della Pietra, V., Lafferty, J.: Inducing features of random fields. IEEE Transactions on Pattern Analysis and Machine Intelligence (1997) 19(4):380-393

[21] Borthwick, A., Sterling, S., Agichtein, E., Grishman, R.: Exploiting Diverse Knowledge Sources via Maximum Entropy in Named Entity Recognition. In the proceedings of the 6th Workshop on Very Large Corpora. (1998)