EHIME Textual Entailment System Using Markov Logic in NTCIR-10 RITE-2

Yuji Takesue
Ehime University
3, Bunkyo-cho, Matsuyama
Ehime, 790-8577, Japan
takesue@ai.cs.ehime-u.ac.jp

Takashi Ninomiya
Ehime University
3, Bunkyo-cho, Matsuyama
Ehime, 790-8577, Japan
ninomiya@cs.ehime-u.ac.jp

ABSTRACT

This paper reports the methods used by the EHIME team for textual entailment recognition in NTCIR-10, RITE-2. We participated in the Japanese BC subtask and Japanese MC subtask. We used Markov logic to infer textual entailment relations. In our Markov logic network, words and hyponyms are used as features.

Team Name

EHIME

Subtasks

Japanese BC, MC

Keywords

Markov logic, Markov logic network, Hyponymy

1. INTRODUCTION

In traditional first-order logic, logic formulas are provided as a set of hard constraints on the possible worlds, and entailment relations of two logic formulas can be inferred by using inference rules. However, the traditional first-order logic has problems to be applied to the real-world problems such as textual entailment recognition due to its hard constraints. Firstly, hard constraints are too strict; if a single formula violates a possible world, the entailment relations cannot be inferred. Secondly, it is difficult for the traditional logic to provide uncertain reasoning, e.g., diagnosing a patient's toothache and identifying the cause of the engine troubles. To handle uncertainty in first-order logic, Markov logic [3, 1] was developed to combine first-order logic and probabilistic graphical models. In Markov logic, each first-order logic formula is given a weight. A set of weighted first-order logic formulas is called a Markov logic network (MLN). Each weight assigned to a logic formula represents a degree of how strong the logic formula should be satisfied.

This paper describes methods used by the EHIME team for textual entailment recognition in RITE-2, NTCIR10 [4]. We used Markov logic for textual entailment recognition. Markov logic enables us to write probabilistic logical relations easily with first-order logic formulas. In the textual entailment recognition for the two text pairs $t1$ and $t2$, we consider textual entailment is related to the common words in both $t1$ and $t2$, and the words that appear only in $t1$ or $t2$. With Markov logic, probabilistic models using these word features can be easily developed.

Section 2 explains Markov logic. In Section 3, textual entailment recognition with Markov logic is presented. The experimental results are described in Section 4.

2. MARKOV LOGIC

2.1 Markov Networks

A Markov network, Markov random field, or undirected graphical model is a probabilistic model for the joint probabilities of a set of variables $\mathbf{X} = (\mathbf{X_1}, \mathbf{X_2}, ..., \mathbf{X_n})$. It consists of an undirected graph G and a set of potential functions ϕ_k. The graph has a node for each variable and undirected edges represent dependencies between variables. The model has a potential function for each clique in the graph, and potential functions are non-negative real-valued functions that represent the state of the clique. The joint probabilities of Markov network is given as follows.

$$P(\mathbf{X} = x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}}) \tag{1}$$

$$Z = \sum_x \prod_k \phi_k(x_{\{k\}}) \tag{2}$$

where $x_{\{k\}}$ is the state of variables in the k-th clique, and Z is a normalization term known as a *partition function*. By replacing the potential functions in the clique with an exponentiated weighted sum of feature functions, we have a log-linear model as follows.

$$P(\mathbf{X} = x) = \frac{1}{Z} \exp\left(\sum_j w_j f_j(x)\right) \tag{3}$$

where $f_j(x)$ is a feature function and w_j is a real-valued weight for feature function $f_j(x)$. In this study, we deal with only binary features $f_j(x) \in \{0, 1\}$ because feature functions in Markov logic correspond to first-order logic formulas, which return only true or false.

2.2 Markov Logic

Markov logic is a unified framework that combines first-order logic and probabilistic graphical models. In Markov logic, a Markov logic network (MLN) is defined as a set of formula-weight pairs (F_i, w_i), where F_i is a first-order logic formula and w_i is a weight for F_i. The weights of the MLN are obtained by learning from a training data. Table 1 and Table 2 show an example of predicates and an MLN.

Table 1: An example of predicates

Predicates	Description
$Family(x, y)$	x and y are family.
$Fat(x)$	x is fat.
$Gout(x)$	x is gout.

Table 2: An example of Markov logic network (MLN)

Weights	Logic Formulas	Description
w_1	$\forall x \forall y Family(x,y)$ $\Rightarrow (Fat(x) \Leftrightarrow Fat(y))$	If some of my family is fat I'm also fat.
w_2	$\forall x Fat(x) \Rightarrow Gout(x)$	Fat men get gout.

Given an MLN L and a finite set of constants, Markov logic defines a Markov network for a set of boolean variables corresponding to the possible grounding of each predicates in L. The predicates without variable, i.e., the variable with constants or functions, are called grounded predicates. The constructed Markov network provides probabilities for the possible worlds as follows.

$$P(\mathbf{X} = x) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right), \tag{4}$$

where $n_i(x)$ is the number of true groundings of F_i in x, and Z is the partition function. $n_i(x)$ is formally defined as follows.

$$n(x) = \sum_{\mathbf{c} \in \mathbf{C^F}} f^F_{\mathbf{c}}(x), \tag{5}$$

where $f^F_{\mathbf{c}}$ is a binary function that returns 1 if F is true in the possible world x such that all the variables in F are replaced with constants $\mathbf{c}$, and returns 0 otherwise.

C^F is a set of all possible tuples of constants for variables in F. That is, all variables in F can be replaced with a tuple in C^F. For example, we have a set of constants $\{Taro, Jiro\}$ and a predicate $Family(x, y)$. All the possible tuples of constants for variables in $Family$ is $\{(Taro, Taro)$, $(Taro, Jiro)$, $(Jiro, Taro)$, $(Jiro, Jiro)\}$, and consequently all the possible ground predicates are $Family(Taro, Taro)$, $Family(Taro, Jiro)$, $Family(Jiro, Taro)$ and $Family(Jiro, Jiro)$

Figure 1 shows the Markov network for the MLN defined with the predicates in Table 1 and first-order logic formulas

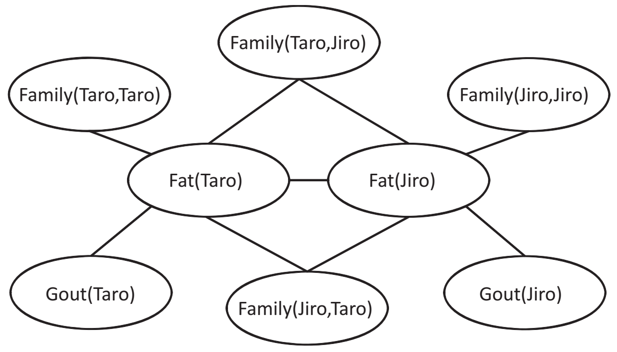

Figure 1: An example of a Markov network constructed from an MLN

in Table 2, given a constants $Taro$ and $Jiro$. Each node is assigned a predicate in which variables are replaced with constants (ground predicate). In the Markov network, edges are connected between the ground predicates in the first-order logic formulas, which form cliques.

3. TEXTUAL ENTAILMENT RECOGNITION WITH MARKOV LOGIC

In this study, we first define the predicates and first-order logic formulas as an MLN for inferring textual entailment relations, and then carry out textual entailment recognition task by using Markov logic with the predicates and the first-order logic formulas. Alchemy [2] and Markov thebeast are the famous tools for Markov logic, that can be used for learning and inference of MLNs. These tools automatically estimate the weights by providing predicates, first-order logic formulas and the database consisting of a set of ground predicates. In this study, we used Alchemy for learning and inferring textual entailment relations.

3.1 Definitions of Predicates

Table 3 and 4 show the definitions of predicates for inferring textual entailment relations.

Table 3: Hidden predicates

Predicate	Definition
Entail($label$, id)	$label \in \{Y, N\}$ indicates the entailment relation for text id.

Table 4: Observed predicates

Predicate	Definition
HasWord($word$, $t1ort2$, id)	$t1ort2 \in \{"t1", "t2"\}$ in text id has $word$
Hypo(hy, $word$)	The hyponym of $word$ is hy

Hidden predicates are the predicates that have unspecified arguments during inference. $label$ in $Entail(label, id)$ is exactly what we want to infer, and is not provided during inference. Observed predicates are the predicates that have fully specified arguments during both learning and inference.

3.2 First-Order Logic Formulas in BC Subtask

We define first-order logic formulas using the predicates defined in the previous section. In Alchemy, the formulas can be defined as hard constraints or soft constraints. Formulas defined as soft constraints are given weights learned from the training data. Formulas defined as hard constraints must be satisfied and are given infinite weights. In what follows, the universal quantifiers at the outermost level of formulas are omitted for simplicity.

Table 5: Hard constraints in BC subtask

Type of Constraints	Formulas
Hard Constraint	$Entail(”Y”, i) \Rightarrow \neg Entail(”N”, i)$
	$Entail(”N”, i) \Rightarrow \neg Entail(”Y”, i)$

Table 6: Formulas for BC Subtask (BC-01 Method)

Type of Constraints	Formulas
Soft Constraint	$HasWord(+w, ”t1”, i) \wedge HasWord(+w, ”t2”, i) \Rightarrow Entail(+lab, i)$
	$HasWord(+w, ”t1”, i) \wedge \neg HasWord(+w, ”t2”, i) \Rightarrow Entail(+lab, i)$
	$\neg HasWord(+w, ”t1”, i) \wedge HasWord(+w, ”t2”, i) \Rightarrow Entail(+lab, i)$
	$\neg HasWord(+w, ”t1”, i) \wedge \neg HasWord(+w, ”t2”, i) \Rightarrow Entail(+lab, i)$
	$HasWord(w1, ”t1”, i) \wedge HasWord(h, ”t2”, i) \wedge Hypo(h, w1) \Rightarrow Entail(+lab, i)$
	$HasWord(h, ”t1”, i) \wedge HasWord(w2, ”t2”, i) \wedge Hypo(h, w2) \Rightarrow Entail(+lab, i)$

Table 7: Formulas for BC Subtask (BC-02 Method)

Type of Constraints	Formulas
Soft Constraint	$HasWord(+w, ”t1”, i) \wedge HasWord(+w, ”t2”, i) \Rightarrow Entail(+lab, i)$
	$HasWord(w1, ”t1”, i) \wedge HasWord(h, ”t2”, i) \wedge Hypo(h, w1) \Rightarrow Entail(+lab, i)$
	$HasWord(h, ”t1”, i) \wedge HasWord(w2, ”t2”, i) \wedge Hypo(h, w2) \Rightarrow Entail(+lab, i)$

In this paper, we propose three methods for BC subtask. Table 5 shows the hard constraint formulas commonly used in these three methods for BC subtask. Table 6, 7 and 8 show the soft constraint formulas used in BC-01 method, BC-02 method and BC-03 method respectively. The phrases in double quotation mean that they are constants. When a variable is prefixed with '+,' the formula including the variable is instantiated with constants, and consequently different formulas are obtained for each possible instantiation of constants. This means that a separate weight is learned for each formula instantiated with different constants. For example, when the first formula in Table 6 is instantiated with

Table 8: Formulas for BC Subtask (BC-03 Method)

Type of Constraints	Formulas
Soft Constraint	$HasWord(+w, ”t1”, i) \wedge HasWord(+w, ”t2”, i) \Rightarrow Entail(+lab, i)$
	$HasWord(+w1, ”t1”, i) \wedge HasWord(h, ”t2”, i) \wedge Hypo(h, +w1) \Rightarrow Entail(+lab, i)$
	$HasWord(h, ”t1”, i) \wedge HasWord(+w2, ”t2”, i) \wedge Hypo(h, +w2) \Rightarrow Entail(+lab, i)$

constants, the following formula is obtained.

$$\begin{aligned} &HasWord(”\text{プロメーテウス}”, ”\text{t1}”, i) \\ &\wedge HasWord(”\text{プロメーテウス}”, ”\text{t2}”, i) \\ &\Rightarrow Entail(”\text{Y}”, i) \end{aligned} \tag{6}$$

This means that if the word ”プロメーテウス” appears in both $t1$ and $t2$ in a text, this implies the entailment relation is true in the text. Such a formula is obtained for each word in the training text, i.e., the number of formulas and weights increases to the vocabulary size. In our experiments, all morphemes that appear in the training data set were instantiated.

Table 6 shows the formulas for BC-01 Method. The first formula in BC-01 Method represents that "if a common word appears in both $t1$ and $t2$, the entailment label is lab." The following three formulas are obtained by negating $HasWord$ predicates, which means that "if a word appears in $t1$ but does not in $t2$, the entailment label is lab," "if a word appears in $t2$ but does not in $t1$, the entailment label is lab," and "if a word does not appear in both $t1$ or $t2$, the entailment label is lab." Though the fourth formula contains only negated $HasWord$ predicates, it is also instantiated for each word in the training text, i.e., all the words in $HasWord$ predicates in the training text constitute the set of vocabularies for the fourth formula. The last two formulas in BC-01 Method represent "if a hyponym appears in the other part of a text, the entailment label is lab."

3.3 First-Order Logic Formulas in MC Subtask

Table 9: Hard constraints in MC subtask

Type of Constraints	Formulas
Hard Constraint	$Entail(”F”, i) \Rightarrow \neg Entail(”B”, i) \vee \neg Entail(”C”, i) \vee \neg Entail(”I”, i)$
	$Entail(”B”, i) \Rightarrow \neg Entail(”F”, i) \vee \neg Entail(”C”, i) \vee \neg Entail(”I”, i)$
	$Entail(”C”, i) \Rightarrow \neg Entail(”F”, i) \vee \neg Entail(”B”, i) \vee \neg Entail(”I”, i)$
	$Entail(”I”, i) \Rightarrow \neg Entail(”F”, i) \vee \neg Entail(”B”, i) \vee \neg Entail(”C”, i)$

We also define the first-order logic formulas for MC Subtask. Table 9 shows the hard constraint formulas commonly used in these three methods for MC subtask. Table 10, 11 and 12 show the soft constraint formulas used in MC-01 method, MC-02 method and MC-03 method respectively.

Table 10: Formulas for MC Subtask (MC-01 Method)

Type of Constraints	Formulas
Soft Constraint	$HasWord(+w, "t1", i) \wedge HasWord(+w, "t2", i) \Rightarrow Entail(+lab, i)$
	$HasWord(w1, "t1", i) \wedge HasWord(h, "t2", i) \wedge Hypo(h, w1) \Rightarrow Entail(+lab, i)$
	$HasWord(h, "t1", i) \wedge HasWord(w2, "t2", i) \wedge Hypo(h, w2) \Rightarrow Entail(+lab, i)$

Table 11: Formulas for MC subtask (MC-02 Method)

Type of Constraints	Formulas
Soft Constraint	$HasWord(+w, "t1", i) \wedge HasWord(+w, "t2", i) \Rightarrow Entail(+lab, i)$
	$\neg HasWord(+w, "t1", i) \wedge HasWord(+w, "t2", i) \Rightarrow Entail(+lab, i)$
	$HasWord(+w, "t1", i) \wedge \neg HasWord(+w, "t2", i) \Rightarrow Entail(+lab, i)$
	$\neg HasWord(+w, "t1", i) \wedge \neg HasWord(+w, "t2", i) \Rightarrow Entail(+lab, i)$
	$HasWord(w1, "t1", i) \wedge HasWord(h, "t2", i) \wedge Hypo(h, w1) \Rightarrow Entail(+lab, i)$
	$HasWord(h, "t1", i) \wedge HasWord(w2, "t2", i) \wedge Hypo(h, w2) \Rightarrow Entail(+lab, i)$

Table 12: Formulas for MC subtask (MC-03 Method)

Type of Constraints	Formulas
Soft Constraint	$HasWord(+w, "t1", i) \wedge HasWord(+w, "t2", i) \Rightarrow Entail(+lab, i)$
	$\neg HasWord(+w, "t1", i) \wedge HasWord(+w, "t2", i) \Rightarrow Entail(+lab, i)$
	$HasWord(+w, "t1", i) \wedge \neg HasWord(+w, "t2", i) \Rightarrow Entail(+lab, i)$
	$\neg HasWord(+w, "t1", i) \wedge \neg HasWord(+w, "t2", i) \Rightarrow Entail(+lab, i)$
	$HasWord(+w1, "t1", i) \wedge HasWord(h, "t2", i) \wedge Hypo(h, +w1) \Rightarrow Entail(+lab, i)$
	$HasWord(h, "t1", i) \wedge HasWord(+w2, "t2", i) \wedge Hypo(h, +w2) \Rightarrow Entail(+lab, i)$

4. EXPERIMENTS

We conducted experiments to evaluate our methods for textual entailment recognition. We used Alchemy [2] as a Markov logic tool for learning and inferring textual entailment relations. We used the default settings for Alchemy except (1) we used generative weight learning (-g option for learning program) and (2) we ran inference using MC-SAT (-ms option for inference program). In Alchemy, predicates in the database are assumed by default to be closed-world.

4.1 Data Set

We used the XML data provided in NTCIR10 RITE-2. The database for Alchemy was constructed from the data analyzed by Japanese morphological analyzer Juman (7.0) and Japanese dependency and case analyzer KNP (4.01). The database was used for training and test. The hyponymy pairs are obtained from the Wikipedia dump data (10. Dec. 2012 version) by using Hyponymy extraction tool (1.0).

The size of the training data for BC subtask is 611, the additional training data was obtained from the data for MC subtask and ExamBC subtask. Among the labels in MC subtask, label F and B mean that the pair is in the entailment relation, and label C and I mean that the pair is not in the entailment relation. We obtained the data for BC subtask from the data for MC subtask by changing the label F and B to label Y, and changing the label C and I to label N. Because the label definitions in ExamBC subtask is the same as BC subtask, we added the data for ExamBC subtask to the data for BC subtask. Finally, we had 1,669 training data and 610 test data for BC subtask. We had 548 training data and 548 test data for MC subtask.

4.2 Experimental Results

Table 13 and 14 show the accuracy and confusion matrix for BC and MC subtasks. Table 15 and 16 show the Macro F1 for BC and MC subtasks.

Table 13: Confusion matrix(BC)

BC-01 Method

Accuracy: 59.34 (362/610)

		system	
		N	Y
gold	N	282	72
	Y	176	80

BC-02 Method

Accuracy: 51.48 (314/610)

		system	
		N	Y
gold	N	207	147
	Y	149	107

BC-03 Method

Accuracy: 48.36 (295/610)

		system	
		N	Y
gold	N	171	183
	Y	132	124

In the experiments, we defined formulas that represent word

Table 14: Confusion matrix(MC)

MC-01 Method

Accuracy: 28.10 (154/548)

		system			
		F	B	C	I
gold	F	71	46	40	48
	B	21	14	16	19
	C	18	17	11	15
	I	59	45	50	58

MC-02 Method

Accuracy: 36.31 (199/548)

		system			
		F	B	C	I
gold	F	117	6	0	82
	B	51	1	0	18
	C	36	1	1	23
	I	123	5	4	80

MC-03 Method

Accuracy: 40.33 (221/548)

		system			
		F	B	C	I
gold	F	137	5	0	63
	B	52	2	0	16
	C	36	2	3	20
	I	124	5	4	79

Table 15: F1-measure(BC)

BC-01 Method

Macro F1: 54.34

Y-F1	Y-Prec	Y-Rec	N-F1	N-Prec	N-Rec
39.22	52.63	31.25	69.46	61.57	79.66

BC-02 Method

Macro F1: 50.14

Y-F1	Y-Prec	Y-Rec	N-F1	N-Prec	N-Rec
41.96	42.13	41.80	58.31	58.15	58.47

BC-03 Method

Macro F1: 48.05

Y-F1	Y-Prec	Y-Rec	N-F1	N-Prec	N-Rec
44.05	40.39	48.44	52.05	56.44	48.31

Table 16: F1-measure(MC)

MC-01 Method

Macro F1: 24.47

B-F1	B-Prec	B-Rec	F-F1	F-Prec	F-Rec
C-F1	C-Prec	C-Rec	I-F1	I-Prec	I-Rec
14.58	11.48	20.00	37.97	42.01	34.63
12.36	9.40	18.03	32.95	41.43	27.36

MC-02 Method

Macro F1: 21.99

B-F1	B-Prec	B-Rec	F-F1	F-Prec	F-Rec
C-F1	C-Prec	C-Rec	I-F1	I-Prec	I-Rec
2.41	7.69	1.43	43.98	35.78	57.07
3.03	20.00	1.64	38.55	39.41	37.74

MC-03 Method

Macro F1: 25.89

B-F1	B-Prec	B-Rec	F-F1	F-Prec	F-Rec
C-F1	C-Prec	C-Rec	I-F1	I-Prec	I-Rec
4.76	14.29	2.86	49.46	39.26	66.83
8.82	42.86	4.92	40.51	44.38	37.26

features and hyponymy features, and we also defined formulas which were obtained by negating predicates. For example, MC-02 Method in Table 11 is obtained by negating predicates in MC-01 Method in Table 10. From Table 14 and 16 show that the accuracy increased by negating predicates, but the Macro F1 decreased. This implies that the expansion of formulas does not always increase the performance of the classifier.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed methods for textual entailment recognition with Markov logic. In Markov logic, a model for inference can be easily developed by giving defitions for predicates and first-order logic formulas.

In this study, we defined only formulas that represent word features and hyponymy features. These formulas are useful, but are not formulas that can pull out the full ability of Markov logic. As a future work, we need to develop elaborate formulas that can infer complex logical relation, such as syllogisms.

6. REFERENCES

[1] P. Domingos and M. Richardson. *Introduction to Statistical Relational Learning*, chapter Markov Logic: A Unifying Framework for Statistical Relational Learning. The MIT Press, 2007.

[2] S. Kok, P. Singla, M. Richardson, P. Domingos, M. Sumner, and H. Poon. *The Alchemy System for Statistical Relational AI: User Manual*, 2007.

[3] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, pages 107–136, 2006.

[4] Y. Watanabe, Y. Miyao, J. Mizuno, T. Shibata, H. Kanayama, C.-W. Lee, C.-J. Lin, S. Shi, T. Mitamura, N. Kando, H. Shima, and K. Takeda. Overview of the Recognizing Inference in Text (RITE-2) at NTCIR-10. In *Proceedings of the 10th NTCIR Conference*, 2013.