Markov Logic Network for Information Extraction

Jiayun Chen

Department of Computer Sciences

University of Texas at Austin

jxc043100@mail.utexas.edu

Undergraduate Honors Thesis

Advisor: Dr. Raymond Mooney

1 Introduction

Due to the increasing amount of available electronic text, there is a growing need for an automated system to extract certain information from natural language documents so they may be organized for data mining purposes. For example, we may be interested in extracting certain data such as names of people, organizations, and locations from the numerous news articles available. This task can be accomplished using named entity tagging, in which tokens in natural language text are classified into categories such as names of people, date, company names, etc. For example, given the following sentence

> *President Bush on Monday launched a rare round of intensive personal diplomacy with Saudi King Abdullah aimed at winning support for a variety of American objectives such as rebuilding Iraq, pressuring Iran, fighting al-Qaeda and backing the U.S.-brokered peace talks between Israel and the Palestinians.*

We may want to tag "President Bush" and "Saudi King Abdullah" as names of people, "Iraq", "al-Qaeda", "Iran" as political entities, etc. Once extracted, these data has can then be analyzed or used to build more structured databases. Named entity tagging is an important task in information extraction and can be applied to a variety of fields such as natural language processing and bioinformatics.

There are a variety of algorithms that could be applied to the problem of named entity tagging (Grishman, 2003). In this paper, we will first introduce a few effective algorithms and models that have been used for named entity tagging as well as their strengths and weaknesses. Then we will propose to use a recently developed model called Markov Logic Network to improve upon existing models. MLN is a very powerful statistical relational learning model that provides a very rich representation (Richardson and Domingos, 2006). We will show that using our approach, we are able to model complex dependencies in our named entity extraction task to achieve a better result. We now begin by introducing some background on related named entity tagging approaches.

2 History and Related Work

2.1 Early named entity tagging.

Early approaches to the problem of named entity tagging involve a lot of human effort (Grishman, 2003). For example, we can write a complex series of regular expressions to match each category of entities. We may also need to develop a large dictionary of common names or locations so each token in the text can be compared to ones in the dictionary. These systems have been quite effective in resolving entities in a particular domain. However, the development of these systems requires a lot of human effort and expertise on the subject, which takes a substantial amount of time to develop. In addition, these forms of named entity taggers are engineered to suit a specific domain, and can not be easily extended to new categories of texts.

These limitations motivated the development of machine learning systems in natural language processing. There are a number of different machine learning approaches for named entity tagging such as decision tree or maximum entropy models. Probabilistic graphical models have been very effective because they are capable of handling uncertainty. A number of different models have been used, including Hidden Markov Models, Maximum Entropy, and others (Grishman 2003). We will now introduce the Hidden Markov Model, which is a rather successful probabilistic graphical model.

2.2 Hidden Markov Models (HMM)

The problem of named entity tagging differs somewhat from that of classification because classifiers such as Naïve Bayes and Logistic Regression are built to predict a variable only using its local features. However, in named entity tagging, the label of a token is dependent upon its local features as well as context. For example, while the word *Washington* itself may in indicate a location, the two words *Mr. Washington* combined gives strong evidence that the phrase denotes a person. Probabilistic graphical models have an advantage in this type of tasks because they have the ability to represent such dependencies. If we make an assumption that most of the context dependencies are short range and between neighboring nodes, then we can arrange the tokens in a sequence. A HMM is such a model that models named entity tagging as a sequence labeling task (Rabiner, 1989). In a sequence labeling task, we are interested in assigning a label to each elements of a sequence. In this case, our sequence is the words that appear in sequential order in a sentence.

A Markov model is a state machine in which state transitions are nondeterministic and transitions are made according to a certain probability distribution. The model obeys the Markov property, in which given the present state, the future states are independent of the past states. A Hidden Markov Model is like a regular Markov model, except that the state is hidden and not visible. However, each state will produce and influence certain output variables. Therefore, by observing a sequence of output variables, we are able to make inferences on the sequence of states that produced the output.

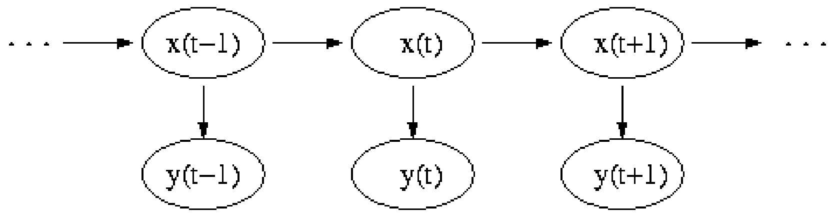


FIG 2.2.1 HMM

The above diagram shows the basic structure of a Hidden Markov Model. X is a random variable ranging over the sequence of states. In the domain of named entity tagging, this might be the named entity labels of the word at position t, which is hidden. Y is a random variable ranging over the output of each state. These outputs are the observed features associated with each word, such as the identity of the word. If we learn the initial probability of the states $p(x_1)$ as well as the transition distribution $p(x_t|x_{t-1})$ and the observation distribution $p(y_t|x_t)$, then we can model the joint probability distribution.

Hidden Markov Models have often been used for sequence labeling tasks, but it does have weaknesses because it is a generative model. Consider the above Hidden Markov Model. There is only one feature variable associated with each state. However, we may want to incorporate more output features in our model to improve the accuracy of the tagger. For example, for a given token, we do not want to only consider the identity of the word. We would also want to take into account features such as the previous token, the next token, capitalization, whether the token includes any digits or symbols, and other features. The goal of generative graphical models is to model the joint probability distributions $p(X, Y)$ where X ranges over the observed features about the tokens we wish to tag, and Y ranges over the label of the tokens. If we add additional features such as the ones above, then all possible combinations of the observed features must be enumerated in order to compute the joint distribution. This is usually very difficult to achieve if we want to maintain tractability of computation. One solution to this problem is to assume that each observation feature is independent from each other. This would reduce the amount of computation necessary, and the entire joint distribution can still be derived. However, in real life, most observations do have complex dependencies, and assuming independence between the features can severely impair the performance of the model.

2.3 Conditional Random Fields (CRF)

Because of the above limitation of a HMM, discriminative models are better suited to the problem of named entity tagging. We will now introduce the CRF, which is a discriminative version of the HMM. It solves the problem of making computations tractable when including complex features (Sutton and McCallum, 2007). Comparison of recent experimental results shows that conditional random fields have leading performances in information extraction (Sutton and McCallum, 2007).

A Conditional random field solves the problem of generative models by modeling the conditional distribution $p(X|Y)$ instead of the joint probability distribution. In the case of named entity tagging, we are interested only in determining the labels of the tokens. A model of the joint distribution is not necessary. A CRF is an undirected graphical model that defines a conditional probability distribution. Because dependencies are captured in its representation, dependencies between observations no longer need to be modeled explicitly. The advantage of using conditional random fields is that they relax the independence assumptions needed by generative models.

In a CRF, each vertex represents a random variable. Each dependent pair of vertices is linked together by an edge. A CRF is globally conditioned on the observation sequence *X*.

Even though the structure of a CRF may be random, in named entity recognition systems, one specific type of CRF is most commonly used, the linear-chain CRF. In a linear-chain CRF, the set of output label variables are arranged in a sequence just like a HMM. The only dependency relationships are between an output variable and the previous variable in the sequence, as well as the observations corresponding to that variable. This closely corresponds to the sequenced Hidden Markov Model discussed earlier.

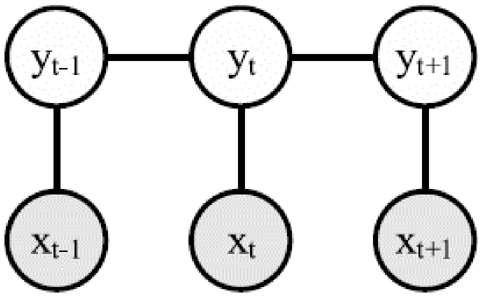


FIG 2.3.1 CRF

In this model, the only dependency relationship would be between each label and its previous label, as well as between each label and the set of features associated with that token. The linear chain CRF has many advantages over CRFs in general. It allows for efficient exact inference while exact inference may be intractable in CRFs in general.

Even though linear chain CRF has many advantages over some of the more traditional models, it also has weaknesses. In a linear chain CRF, we assumed that the only dependencies are between the labels of adjacent words. Thus, CRFs are not able to use information from longer range dependencies to assist in the labeling. Returning to our previous example, even if *Mr. Washington* is tagged correctly as a name because it is preceded by a title, another occurrence *Washington* by itself later in the text may be incorrectly tagged as a location. We would like to add additional dependencies such that the labeling of one token would influence the labeling of another identical token.

As a result, researchers have turned to using a variety of statistical relational learning methods to increase the accuracy. Statistical relational learning (SRL) is a combination of probabilistic learning and relational learning (Richardson and Domingos, 2007). The strength of probabilistic models is that they can handle uncertainty in learning and reasoning. Meanwhile, first order logic or relational databases can effectively represent a wide range of knowledge. Statistical relational learning techniques attempt to combine the strength of the two approaches. This combined strength of probabilistic learning and relational learning gives SRLs more power in learning and inferences (Richardson and Domingos, 2007). Recently, there has been some interest in the application of SRL techniques to information extraction. Bunescu and Mooney have used Relational Markov Networks to identify protein names in biomedical text (Bunescu and Mooney, 2007). Domingos and Poon have applied Markov Logic Networks for the segmentation and entity resolution of bibliographic citations (Poon and Domingos, 2007).

Here, we will propose to use the power of Markov Logic Networks to model these longer range dependencies. We will now introduce Markov Logic Networks, and then present the specific MLN representations we used for our named entity tagging problem.

2.4 Markov Logic Networks (MLN)

Markov Logic Networks is SRL technique that combines first order logic and Markov Networks. Combining the strengths of these two techniques give us the flexibility of knowledge representation as well as the ability to handle uncertainty. We will now provide an introduction to Markov Networks as well as first order logic and then discuss how these two techniques are combined in Markov Logic Networks

2.4.1 Markov Networks

A Markov Network is an undirected probabilistic graphical model that specifies the joint probability distribution. In a Markov network, each vertex represents a random variable and each edge indicates a dependency relationship between the two vertices it links. A node in the Markov network is conditionally independent of another node in the Markov network given its set of neighbors. Each clique in the graph has a corresponding potential function associated with it that maps the possible states of the elements in the clique to non-negative real numbers.

The joint distribution of a Markov Network is

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

Where X is a vector of random variables represented in the Markov Network, x is an assignment to these random variables, φ_k is a potential function that maps the state of the kth clique to non-negative real numbers, and Z is a normalization constant that makes sure that the probability of all possible values of x sums to 1.

A Markov Network is usually more conveniently represented as a log-linear model

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

In this representation, $f_j(x)$ is a feature corresponding to each possible state of the clique, and w_j is the weight associated with the feature, which is usually the value of the potential function.

In a Markov Network, exact inference is #P complete (Roth, 1996). Therefore, approximation algorithms such as Markov Chain Monte Carlo and Gibbs Sampling have been used for efficient approximation inference (Gilks et al., 1996). Belief propagation is another commonly used approximation technique (Yedidia et al., 2001).

2.4.2 First Order Logic

First Order Logic is a system of representing knowledge in deductive systems. This system is much more expressive than propositional logic because it allows for predicates as well as quantification.

The syntax of first order logic is composed of user defined variables, constants, functions, and predicates.

A variable ranges over objects in the domain, and are usually represented as lower case letters.
A constant represents a specific object such as a particular person (John), or a specific organization (University of Texas).
Functions are used to map certain objects or tuples of objects to other objects (John = FatherOf(Steve)).
Predicates define certain attributes or relationships between objects (Parent(John, Steve)).

First order logic also includes a set of logical connectives and quantifiers. The connectives include $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (implication) and $\leftrightarrow$ (equivalence). Quantifiers include the universal quantifier ($\forall$) and existential quantifier ($\exists$). ($\forall$x)P(x) is true only if P(x) is true for all possible values of x in the domain. ($\exists$x)P(x) is true if P holds for at least one value of x in the domain.

A term is a constant symbol, a variable symbol, or function applied to other terms. For example, x, John, and FatherOf (Steve) are terms.

An atom or atomic formula is a predicate applied to terms. For example, Parent(John, Steve) is an atom. Atoms can be used to construct formulas by connecting one or more atoms using connectives. For example, if p and q are atomic formulas, then ~q, p ^ q, p → q, etc are also formulas. Formulas can be universally quantified or existentially quantified.

A first order knowledge base is a set of formulas in first order logic. It is usually represented in conjunctive normal form for convenience. Conjunctive normal form is a conjunction of clauses. A clause is a disjunction of literals, which are atoms or negated atoms. Inference can then be performed on such a knowledge base. First order logic provides an expressive way to describe knowledge, but it is impractical in many AI systems. The inference algorithms commonly used include forward chaining, backward chaining, and resolution. Unfortunately, these systems are usually intractable computationally. For first order logic, Godel's Completeness Theorem states that it is possible to prove a sentence if it is entailed by the knowledge base (Russel 2004). However, resolution cannot be used to prove that a sentence is not entailed by the knowledge base. Therefore, resolution theorem proving is only semidecidable. Another weakness in using pure first order logic is that a formula in the knowledge base must be always true, and the knowledge base cannot have any inconsistencies within it. In most cases, each formula may be true most of the time, but one cannot overlook the possibility that it may be false (Richardson and Domingos, 2006). This problem can be solved using Markov Logic Networks, which combine first order logic with Markov networks.

2.4.3 Markov Logic Networks

A Markov logic network consists of a first-order knowledge base with a weight attached to each formula. In a first-order knowledge base, each formula is a hard constraint on the state of the possible worlds. Therefore, if even one formula is violated, the probability of such a world becomes 0. A Markov Logic Network tries to soften this constraint to handle uncertainty. Each weight attached to the formula specifies how strong the constraint is. The stronger the weight, then a world in which the formula is satisfied would have a higher probability than a world in which it is not. If a world violates a few formulas, its probability does not become 0. It is just less likely to occur than one that does not violate any formulas.

A Markov logic network consists of a set of pairs of (F_i, w_i) where F_i is a formula in first order logic, and w_i is a real number weight associated with the formula. The Markov logic network also contains a finite set of constants. The pairs of formulas and weights together with the set of constants can be used as a template for construction ground Markov networks.

A predicate is ground if all variables in the predicate have been substituted with constants. Each possible grounding of each predicate becomes a node in the corresponding Markov Network. The value of the node is 1 if the ground predicate is true, and 0 if the ground predicate is false. Each possible grounding of each formula becomes a feature in the Markov Network. The value of the feature is 1 if the ground formula is true, and 0 if the formula is false. Each feature will have a weight attached to it. If the ground predicates appear together in at least one grounding of one formula, this indicates that there is a dependency relationship between the predicates, and hence the Markov Network would have an edge to link these nodes. Therefore, each formula forms a clique in the ground Markov Network. Once constructed, inference in the ground Markov Networks can be performed in the same way as in regular Markov Networks.

For example, in our entity tagging model, we can define some first order logic formulas to capture our model:

Identical tokens should be tagged the same: $SameToken(x, y) \wedge Label(x, l) \Rightarrow Label(y, l)$

In this formula, *x* and *y* are variables of type token, and *l* is a variable for type label We may also define a weight for this formula, or simply learn the weights from training data.

Applying this formula to the tokens *A* and *B,* and labels Person (P), and Organization (O) will result in the following ground Markov network:

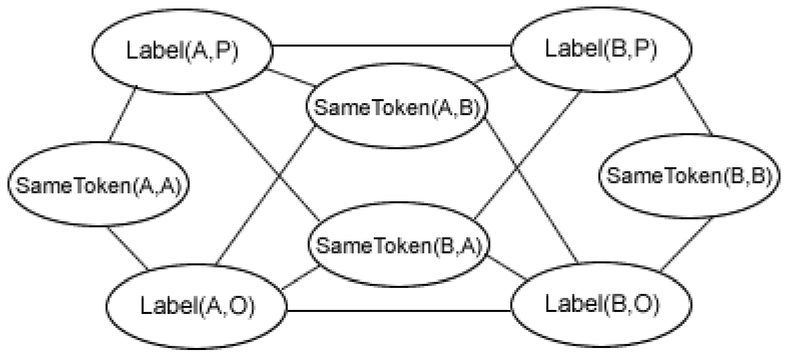


FIG 2.4.1 GROUNDED MARKOV NETWORK

Markov Logic Networks subsume all other propositional probabilistic models. Probabilistic graphical models can be represented as MLNs simply by defining predicates and formulas for the features. A first order knowledge base can also be represented as a MLN by setting all the weights to infinity. A MLN can be used to represent these other models, but it is more powerful than them. A MLN can handle uncertainty and contradictions within the knowledge base. It can also model more dependency relationships through its first order logic knowledge representation. This gives MLNs more power over CRFs in its application in named entity tagging.

3. Algorithms and Methodology

3.1 Alchemy Software

Our MLN model will be developed using the Alchemy system, which is a software package developed at the University of Washington that provides interfaces and algorithms for modeling Markov Logic Networks (alchemy.cs.washington.edu).

This software package allows us to represent a MLN simply by defining our first-order logic predicates and formulas in an input document. Therefore, it provides a very versatile system for defining arbitrary dependencies in the MLN. We will then provide the ground clauses to Alchemy for training.

3.2 MLN representation

We will now discuss our detailed approach to use MLN's for information extraction. To keep the MLN simple, we used only three features:
The identity of the word (W), its part of speech (POS), and its capitalization pattern (Cb). In our MLN, we will declare predicates to represent each of our features: *W(word, w)*, *Pos(word, pos)*, *Cb(word, cb)*. Our task is to determine the label of a token given its features and its context. So our query predicate is Label(word, lab).

3.2.1 Basic MLN to Model CRF

Because the label of a token depends on its features, we will define rules to capture this dependency:
$W(x, +a) \Rightarrow Label(x, +l)$
$Cb(x, +a) \Rightarrow Label(x, +l)$
$Pos(x, +a) \Rightarrow Label(x, +l)$

When a "+" operator is applied, a separate weight is learned for each formula obtained by grounding that variable to each of its values. For example, the formula Cb(x, +a) $\Rightarrow$ Label(x,+l) will be expanded into the following formulas:

Cb(x, Capitalized) $\Rightarrow$ Label(x, Person)
Cb(x, Capitalized) $\Rightarrow$ Label(x, Organization)
Cb(x, Lowercase) $\Rightarrow$ Label(x, Person)
Cb(x, Lowercase) $\Rightarrow$ Label(x, Organization)

A distinct weight will be learned for each of these formulas.

To model the dependency between the label of the previous word and the current label as in a linear chain CRF, we now must define a predicate to represent contextual information: *Neighbor(word, word)*, representing two consecutive tokens.
We will also add the following rule into our MLN:

Neighbor(x, y) $\wedge$ *Label(x, +l)* $\Rightarrow$ *Label(y, +l)*

3.2.2 Including More Short Range Contextual Features

In named entity tagging, knowing more information about a token's context will significantly improve the accuracy of the system. For example, knowing that the previous word is *Dr.*, or that the next word is *said* will greatly increase the probability that the current word is a person. Therefore, we will now modify our MLN to include information about the previous and next word as well as their part of speech and capitalization pattern. To incorporate more contextual information, we will define rules to associate a token with features of neighboring tokens.
Neighbor(x, y) $\wedge$ *W(x, +a)* $\Rightarrow$ *Label(y, +l)*
Neighbor(x, y) $\wedge$ *W(y, +a)* $\Rightarrow$ *Label(x, +l)*
We will also add similar rules for Pos and Cb.

3.2.3 Including Long Range Dependencies

In addition to modeling neighboring context, we would also like to incorporate longer range dependencies. For example, if two tokens are the same word, then they are likely to be tagged the same in the same document. One variation of CRF's, the skip chain CRF, is able to model such dependencies between identical tokens (Sutton and McCallum, 2007). However, adding this dependency requires a complete modification of the CRF's structure. The advantage of using a MLN is that arbitrary dependencies may be specified in just a few formulas.

We will define a predicate to represent two tokens that are the same word:
SameToken(word, word).

The label of one token will have some influence over the label of another identical token: *SameToken(x,y)* $\wedge$ *Label(x, z)* $\Rightarrow$ *Label(y,z)*.

3.3 MLN Weight Training

In addition to the formulas, a MLN must also include the relative weights of each of these clauses. For some cases, it is reasonable to hard code these weights, but in our case it is unlikely that we will know the relative strength of all of the above dependencies before hand. Therefore, we must train the model to automatically learn the weights of each formula.

In our system, we already know a specific query predicate we would like to predict, which is the label of a token. Therefore, using a discriminative training approach will be more effective because in discriminative training, we attempt to maximize $p(q|e)$ instead of $p(q,e)$, where q is the query predicates and e is the evidence predicates.

The state-of-the-art discriminative weight learning algorithm for MLN's is the voted perceptron algorithm (Lowd and Domingos, 2007). The voted perceptron is a gradient descent algorithm that will first set all the weights to zero. It will iterate through the training data and update the weights of each of the formulas based on whether the predicted value of the training set matches the true value. Finally, to prevent overfitting, we will use the average weights of each iteration rather than the final weights. In order to train the data using the voted perceptron algorithm, we must know the expected number of true groundings of each clause. This problem is generally intractable, and therefore, the MC-SAT algorithm is used for approximation. Refer to Singla and Domingos (2005) for a more detailed discussion of the weight training algorithm.

3.4 MLN Inference

Inference in MLN requires reasoning with both probabilistic and deterministic dependencies. Traditionally, MCMC algorithms have been used for inference in probabilistic models, and satisfiability algorithms have been used for pure logical systems. Since a MLN contains both probabilistic and deterministic dependencies, neither will give good results. In our experiments, the MC-SAT algorithm will be used to determine the values of query predicates. The MC-SAT is an algorithm that combines MCMC and satisfiability techniques, and therefore performs well in MLN inferences. Refer to Poon and Pedro Domingos (2006) for a more detailed discussion of the inference algorithm.

4. Experimental Methodology and Results

4.1 Dataset

In order to compare the MLN approach with existing approaches, we tested our approach on the ACE dataset (http://projects.ldc.upenn.edu/ace/). This dataset contains 97 news articles from various sources. Each document contains approximately 400 words. Our task is to tag each word in the dataset with one of the following labels:

1. Person
2. Organization
3. Location
4. Facility
5. GPE(Geopolitical Entity)
6. None

4.2 Evaluation metrics

In our experiments, we used cross validation to estimate the accuracy of our named entity tagging system. We divided our data into 10 different subsets. Each time, we selected a fraction of the data for training, and used all the rest of the data for testing.

For each of the training and testing sets, we preprocessed the data and extracted the capitalization and word features for each token. The part of speech feature is extracted using opennlp (opennlp.sourceforge.net).We then used the Alchemy package for training as well as testing.

The MLN tagging system will output the probability of all six labels for a given token. We used the label with the highest probability as the predicted output label.

The accuracy of our system is evaluated using three standard metrics: precision, recall, and F1.

Precision is the fraction of the tokens tagged as a label other than "NONE" that are tagged correctly:

$$\text{Precision} = \frac{NumberCorrect}{TotalTagged}$$

Recall is the fraction of the tokens that should be tagged as a label other than "NONE" that are tagged successfully:

$$\text{Recall} = \frac{NumberCorrect}{ExpectedLabels}$$

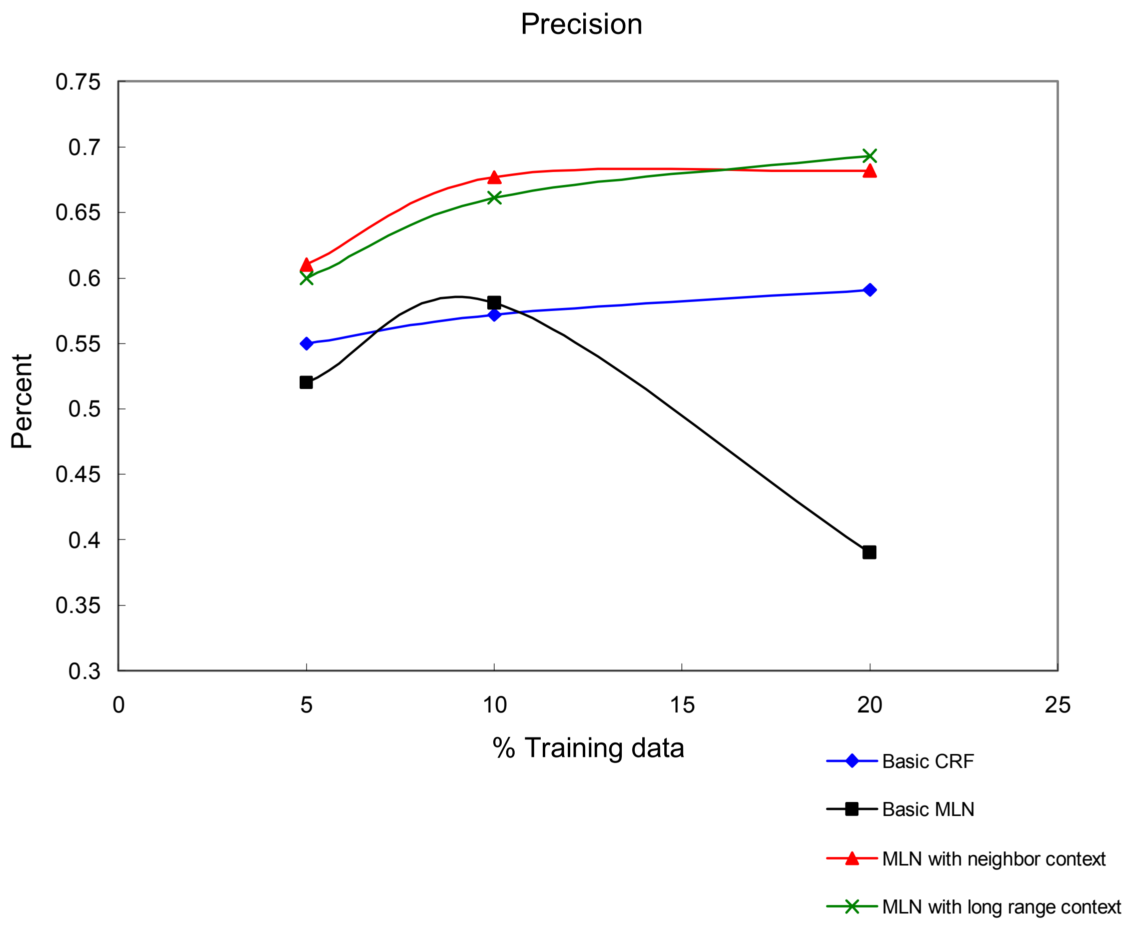
The F-measure is the weighted harmonic mean of precision and recall. We used the F1 measure, which weighed precision and recall equally:

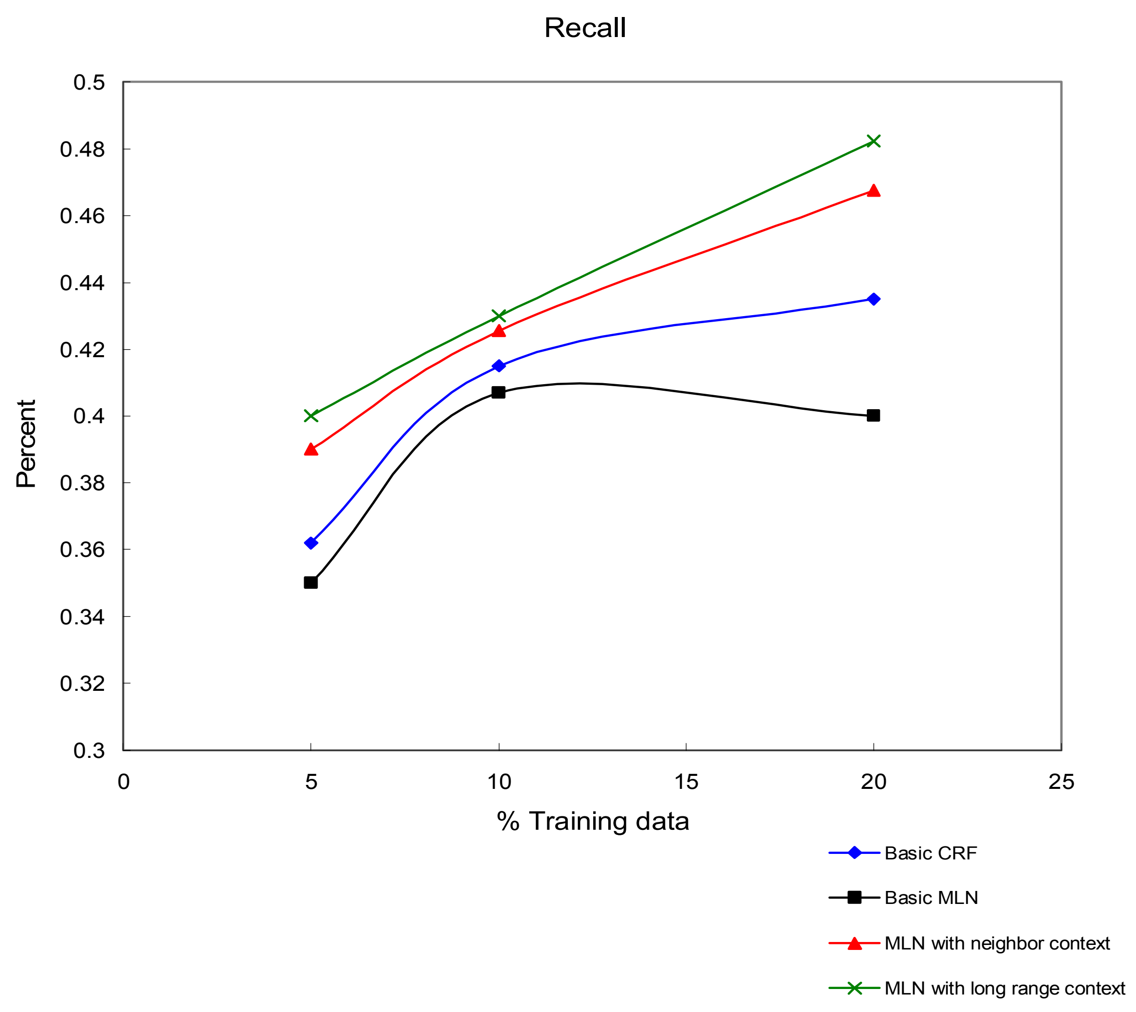
$$\text{F1}=\frac{2*(\Pr ecision * \operatorname{Re} call)}{(\Pr ecision + \operatorname{Re} call)}$$

4.3 Comparison of CRF vs. MLN

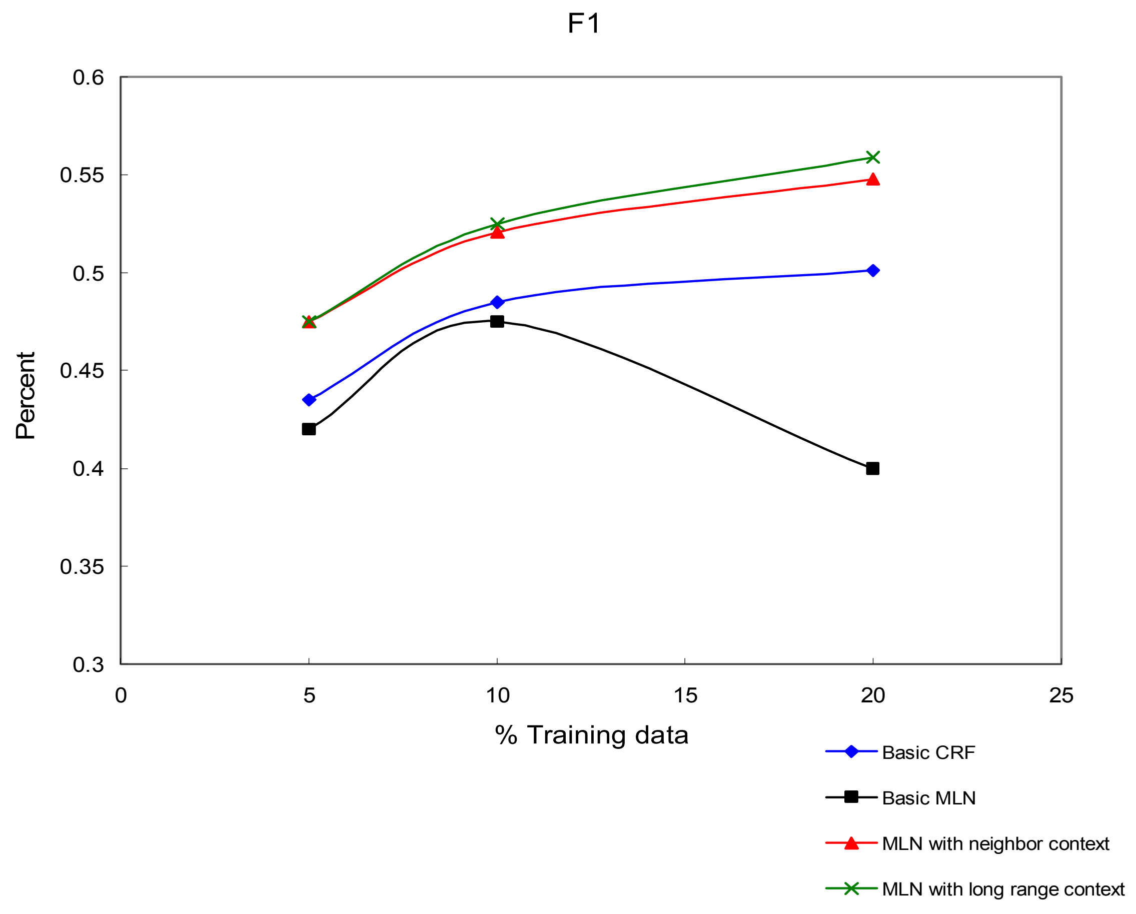
In our experiment, we first constructed a first order linear chain CRF using Mallet (http://mallet.cs.umass.edu/) with the basic features described above. Mallet is a java package that can be applied to a variety of machine learning tasks. We used its CRF class to construct our CRF. We then constructed the MLN representation in section

3.1.1 to represent similar dependencies as the CRF. We trained the CRF and MLN on 5%, 10%, and 20% of the training data. We did not attempt to train our models with more than 20% training data because the training time for the MLN increases dramatically. When we included both short and long range context, training the MLN with 10% of the training data will expand it into about 20000 formulas, and training with 20% of the training data will expand the MLN into about 30000 formulas. The number of ground clauses for each 10% of the data is over 100000. Therefore, training and testing are both very time consuming for bigger datasets.





Recall
Percent
0.5
0.48
0.46
0.44
0.42
0.4
0.38
0.36
0.34
0.32
0.3
0
5
10
15
20
25
% Training data
Basic CRF
Basic MLN
MLN with neighbor context
MLN with long range context



F1
Percent
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0
5
10
15
20
25
% Training data
Basic CRF
Basic MLN
MLN with neighbor context
MLN with long range context

We noticed that the CRF outperformed the MLN slightly, especially with 20% training data, in which the MLN performance actually declined. Next, we tested the MLN with additional neighboring context. This MLN achieved about a 5% better performance in precision, recall, and F1, which is reasonable because we are using neighboring token's features to help determine the label. Finally, we incorporated the long range dependencies. In the preprocessing, we removed SameToken clauses containing irrelevant words such as articles, prepositions to reduce noise. Removal of clauses containing stop words gave us better results, especially with 20% training data. For training data less than 20%, the improvement in precision is offset by a drop in precision. For 20% training data, the overall improvement is much more significant. We used a paired t-test to compare the result for before and after the long range dependency is added to the MLN. We found that the improvement by using long range dependencies is statistically significant when training on 20% of the training data, which gave us a P=0.043.

5 Future Work

Including additional features or conjunctions of features

MLN provides a very rich representation and the ability to model complex dependencies easily. This allows a lot of room for improvement upon this existing model.

One way to improve our tagger is to include a larger selection of relevant features such as bigrams or whether the word appeared in any dictionaries such as a dictionary of names or companies. The inclusion of more contextual information should be able to improve the performance.

Another possibility is to use conjunctions of features to improve labeling results. Sometimes, the combination of a few features will give us indicative information about the label (McCallum 2003). The expressive representation of MLN gives us the ability to represent this dependency very easily compared to other models. However, we must also be careful to select only relevant conjunctions so the performance will not be penalized due to overfitting.

Co-reference resolution

The tagger we constructed simply classifies each token into categories. In the ACE dataset, each entity may have multiple mentions, and it is useful to determine which entity mentions are referring to the same entity. This is the problem of co-reference resolution (Denis and Baldridge, 2007). Building upon our existing model, we may be able to define an additional MLN to determine which of our labeled entities represent the same entity.

6 Conclusion

This paper has shown an approach for utilizing Markov Logic Networks for the problem of named entity tagging. We also improved upon our basic model by including short and long range context to influence the predicted labels. We showed that the inclusion of short range context can increase the F1 of the labeling by 5%, and that including long range context can positively increase the recall of our labeling. Despite this positive increase, the overall improvement is confined to using 20% of the training data.

The expressive representation power of MLN's provides us a flexible framework for modeling information extraction problems. Using this framework, we can easily model other graphical models, but we can also modify them to incorporate more complex dependencies by the inclusion of additional formulas. Unfortunately, using MLN's do pay a heavy price in training and inference time.

References

Bunescu and Mooney. 2007. Statistical Relational Learning for Natural Language Information Extraction. *Introduction to Statistical Relational Learning* (pp. 535-552), Cambridge, MA: MIT Press.

Denis and Baldridge. 2007. Joint determination of anaphoricity and coreference resolution using integer programming. In Proceedings of NAACL-2007. Rochester, NY.

Domingos and Richardson. 2007. Markov Logic: A Unifying Framework for Statistical Relational Learning. *Introduction to Statistical Relational Learning* (pp. 339-371). Cambridge, MA: MIT Press.

Gilks, W. R., Richardson, S., & Spiegelhalter, D. J. (Eds.). 1996. *Markov Chain Monte Carlo in Practice*. London, UK: Chapman and Hall.

Grishman, Ralph, Information Extraction. *The Oxford Handbook of Computational Linguistics*, 2003.

Lowd and Domingos. 2007. Efficient Weight Learning for Markov Logic Networks, Proceedings of the Eleventh European Conference on Principles and Practice of Knowledge Discovery in Databases (pp. 200-211). Warsaw, Poland: Springer.

McCallum, Andrew. 2003. Efficiently Inducing Features of Conditional Random Fields. Conference on Uncertainty in Artificial Intelligence (UAI).

Poon and Domingos. 2006. Sound and Efficient Inference with Probabilistic and Deterministic Dependencies. Proceedings of the Twenty-First National Conference on Artificial Intelligence (pp. 458-463), Boston, MA: AAAI Press.

Poon and Domingos. 2007. Joint Inference in Information Extraction. Proceedings of the Twenty-Second National Conference on Artificial Intelligence (pp. 913-918). Vancouver, Canada: AAAI Press.

Rabiner, L.R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. In Proceedings of the IEEE (pp. 257-286), Vol 77.

Roth, D. 1996. On the Hardness of Approximate Reasoning. *Artificial Intelligence*, *82*, 273.302.

Richardson and Domingos. 2006. Markov Logic Networks. *Machine Learning*, 62, 107-136.

Russel and Norvig. *Artificial Intelligence a modern approach*. Prentice Hall, 2004.

Singla and Domingos. 2005 Discriminative Training of Markov Logic Networks. Proceedings of the Twentieth National Conference on Artificial Intelligence (pp. 868-873). Pittsburgh, PA: AAAI Press.

Sutton and McCallum. 2007. An Introduction to Conditional Random Fields for Relational Learning. *Introduction to Statistical Relational Learning*. Cambridge, MA: MIT Press.