Ontology Guided Information Extraction Using Markov Logic Network

Project report submitted to
Visvesvaraya National Institute of Technology, Nagpur
in partial fulfillment of the requirements for the award of
the degree

Bachelor of Technology
In
Computer Science and Engineering

by

Ritwik Deshpande **BT16CSE073**
Shalaka Patankar **BT16CSE081**
Kapeel Suryavanshi **BT16CSE084**
Anushka Kulkarni **BT16CSE100**

Under the Guidance of
Dr. Umesh A. Deshpande
Mr. Sapan Shah
Mr. Avadhut Sardeshmukh

Department of Computer Science and Engineering
Visvesvaraya National Institute of Technology
Nagpur 440 010 (India)

2020

Ontology Guided Information Extraction Using Markov Logic Network

Project report submitted to
Visvesvaraya National Institute of Technology, Nagpur
in partial fulfillment of the requirements for the award of
the degree

Bachelor of Technology
In
Computer Science and Engineering

by

Ritwik Deshpande **BT16CSE073**
Shalaka Patankar **BT16CSE081**
Kapeel Suryavanshi **BT16CSE084**
Anushka Kulkarni **BT16CSE100**

Under the Guidance of
Dr. Umesh A. Deshpande
Mr. Sapan Shah
Mr. Avadhut Sardeshmukh

Department of Computer Science and Engineering
Visvesvaraya National Institute of Technology
Nagpur 440 010 (India)

2020

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Visvesvaraya National Institute of Technology, Nagpur

DECLARATION

We, hereby declare that this project work titled **"Ontology Guided Information Extraction Using Markov Logic Network"** is carried out by us in the **Department of Computer Science and Engineering** of Visvesvaraya National Institute of Technology, Nagpur. The work is original and has not been submitted earlier whole or in part for the award of any degree/diploma at this or any other Institution / University.

Ritwik Deshpande
(BT16CSE073)

Shalaka Patankar
(BT16CSE081)

Kapeel Suryavanshi
(BT16CSE084)

Anushka Kulkarni
(BT16CSE100)

Date : 3rd June, 2020.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Visvesvaraya National Institute of Technology, Nagpur

CERTIFICATE

This is to certify that the project titled **"Ontology Guided Information Extraction Using Markov Logic Network"**, submitted by **Ritwik Deshpande, Shalaka Patankar, Kapeel Suryavanshi,** and **Anushka Kulkarni** under the Guidance of **Dr. Umesh A. Deshpande, Mr. Sapan Shah,** and **Mr. Avadhut Sardeshmukh** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering**, VNIT Nagpur. The work is comprehensive, complete, and fit for final evaluation.

(Dr. Umesh A. Deshpande)

Project Guide & Head of Department,
Department of Computer Science
and Engineering, VNIT Nagpur.

Date : 3rd June, 2020.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our guide, Dr. Umesh A. Deshpande, for giving us the opportunity to work on this research project of **Tata Research Development and Design Center**, guiding us thoughtfully throughout this project and giving us directions whenever required. We would also like to thank Mr. Sapan Shah and Mr. Avadhut Sardeshmukh for their guidance and help in the implementation of the project and for providing the necessary resources whenever required by us. We would like to thank all the teaching and non-teaching faculty of the Computer Science and Engineering Department for supporting us and providing necessary facilities at all times. We would like to acknowledge the contribution of those who have constantly supported and encouraged us which helped in the successful completion of our project.

ABSTRACT

Understanding text and extracting useful information from the text using either Artificial Intelligence (AI) or Machine Learning (ML) has its own challenges and complexities, and it poses a significant uncertainty in the output obtained. This dissertation is aimed to address these challenges by proposing a unifying approach where the Markov logic network (MLN) is used to improve the results obtained by ML alone.

Using the entities and relations obtained from the ML-based classifier as input (seed instances), the task was to extract additional information in a given corpus of documents that was missed by the ML-based classifiers. The weak evidences i.e. extracted entities and relations obtained from ML-based entity and relation classifiers along with the strong evidences obtained from the dataset, were further used to train the MLN based classifier.

It was observed that the ML-based classifier missed identifying several relations in the given corpus of documents. The aim was to identify these missed relations by combining First Order Logic rules of AI with Probabilistic Graphical Models to obtain a Markov Logic Network. The relations misclassified by the ML model were successfully identified by MLN based classifiers, thus justifying the correctness of this approach. Improvement was seen in the overall accuracy of Relation Extraction. ML-based classifiers gave an accuracy of 63% whereas MLN was able to improve it to 66%.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

MLN	Markov Logic Network
NER	Named Entity Recognition
PRM	Probabilistic Relational Models
RDN	Relational Dependency Network
BLP	Bayesian Logic Programs
MRF	Markov Random Field
SRL	Statistical Relational Learning
RML	Relational Machine Learning
CoNLL	Computational Natural Language Learning
BIO Encoding	Beginning, Inside, Outside Encoding Scheme
NERC	Named Entity Recognition and Classification
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
BiLSTM	Bidirectional Sequential Long Short Term Memory
KB	Knowledge Base
MCMC	Markov Chain Monte Carlo
MAP	Maximum-A-Posteriori
UNK & PAD	Special Tokens - Unknown & Padding
Document	A sentence in the CoNLL dataset
Word	String of characters, each Word is mapped to a unique integer
Tag	Labels assigned to Words (Entity Label) or Sentence (Relation Label)
Essential Relations	Kill, Work_For, Live_In, Located_In, OrgBased_In

FOL	First-Order Logic
I/p & O/p	Input & Output
CNF	Conjunctive Normal Form

Chapter 1. INTRODUCTION

1.1 Introduction to entity and relation classification

Entity classification, also known as Named Entity Recognition (NER), is an information extraction technique that refers to the process of identifying and classifying key elements from text into predefined categories. This helps to convert unstructured data into a machine-readable structured form accessible for standard processing that can be used to retrieve information, extract facts, and answer questions.

Relationship classification is the task of extracting semantic relationships from a text. Extracted relationships usually occur between two or more entities of a certain type (e.g. Person, Organisation, Location) and fall into a number of semantic categories (e.g. located in, employed by, lives in). The base of relation classification is usually entity classification. Finding out relations between classified entities helps to extract further information from the structured data and work towards respective goals.

Text appears as unstructured data in some formats, such as document files, spreadsheets, web pages, and social media. Being able to identify entities — people, locations, organizations — and relationships between them (if they exist) offers an opportunity for any individual who must utilize the data to comprehend what it contains. As a result, the analyst would be able to see an organized description of all names of individuals, businesses, products, cities, or countries in a database that might act as a reference point for further study and research.

1.2 Introduction to SRL and MLN

In certain real-world contexts and challenges, primary data is sometimes believed to be independent and fairly dispersed. However, the data in the real world is much more complex, consisting of many relationships within the data itself. For the most part, existing AI systems can deal with one of these issues (data uncertainty or complex relations), but not both. Such data in the relational paradigm can be represented and modeled using Statistical Relational Learning (SRL).

1.2.1 Statistical Relational Learning

Statistical Relational Learning [1,2,3] (SRL), also referred to as Relational Machine Learning (RML), is an AI and ML subdomain that deals with models having both uncertainty (which can be addressed using statistical methods) and complex relationships between various entities in the data. It uses both first-order logic in order to describe a domain's relational properties in a general way (universal quantification) and model uncertainty by means of probabilistic graphical models (such as Bayesian networks or Markov networks). SRL thus helps resolve two essential ML assumptions. First, propositional data algorithms render and believe that the data instances are documented in a way that each entity has a certain number of attributes which is constant. Second, algorithms presume that data instances are autonomous, but relational data contradicts this assumption in real-world situations.

There are four main models in SRL:

1) Probabilistic Relational Models (PRM)
2) Markov Logic Networks (MLN)
3) Relational Dependency Network (RDN)
4) Bayesian Logic Programs (BLP)

1.2.2 Markov Logic Networks

MLN is a powerful framework that combines statistics (Markov Random Fields) and logical reasoning (First-order logic). It extends the ideas of Markov Network to first-order logic, enabling uncertain inference. It can be viewed as a combination of Markov Random fields and First-order logic. MLN is discussed in detail in chapter 6.

1.3 Significance of topic - why MLN?

There is a major problem in applying most of the traditional machine learning algorithms to databases. Some assumptions made by these algorithms are not applicable. One of these assumptions is that all the data used in the training or testing is derived from a single table. In statistical learning, one row is considered to be an object and every variable is one column of the table. When two tables are joined, one section of that

resulting table might be a copy of another section of the same table. If table X is joined with table Y and table Z, then two copies of table X are obtained. Considering the statistics used by machine learning, this redundancy might create problems because it changes the frequency of different entries of the table.

Markov Logic is a high-level language that can be applied to different types of linked data. Markov Logic makes use of these relational properties that are found in data sets. Pattern detection and inference is better than the unstructured tables that are used by most machine learning approaches [4].

Essentially, in Markov Logic, features are defined or learned from data as formulae in first-order logic. The difference from traditional machine learning is that it has weights assigned to these features. A perceptron, for example, has some Boolean features with weights. Here, the features are FOL rules and predicates. Since the predicates can share variables, the features are defined over multiple tables in Markov Logic. The features are learned from data and the learning of weights is done using these features. Thus, the features and the weights together define a probability distribution over possible datasets. The brittleness that is associated with the features of a traditional machine learning model is relaxed in an MLN and each can have a probability in between 0 and 1 as opposed to either 0 or 1.

Exploring new avenues and methods for a classical problem such as information extraction is the way to more accurate results or faster implementations or possibly both.

1.4 Applications of MLN

Owing to the advantages it provides with its expressive power and the efficient learning and inference algorithms described above, Markov Logic has been used for a broad array of AI applications that include information extraction [5], link prediction [6], semantifying Wikipedia [7], robot mapping [8] and many others.

The problem of entity resolution is to identify which of the references correspond to the same underlying object, given a set of annotated objects. For many applications, this is a very critical topic and has been widely discussed in literature [9]. It is possible to present a unified approach [10] to entity resolution using Markov logic, whose

formulation will allow several previous approaches to be expressed using minimal FOL rules in Markov logic. Also, this framework allows easy integration of new approaches.

Another application of Markov logic can be to identify social relationships in collections of images [11]. Given a user's collection of personal photos, the aim is to identify the different types of social relations present in that collection (e.g. child, parent, friend, etc.). The approach to this consists of writing a domain theory in Markov logic to describe heuristic rules, such as, parents are mostly photographed with their children and relatives, friends are clustered across images of a single day, etc. Experiments have demonstrated that this approach can give noticeable improvements over the benchmark [12].

Codifying Domain Information to improve the results:

In MLN, we can use the domain information available to form first-order rules for it using weak and strong evidence. These first-order rules are then assigned weights during learning, which may provide a better result during inference than ML alone.

1.5 Organization of Thesis

The problem statement, along with the implementations of the classifiers and MLN are mentioned in the next chapters of the thesis. Chapter 3 focuses on the problem statement in detail and the approach used. Chapter 4 and Chapter 5 states the implementation of Entity and Relation Classifier. Chapter 6 gives a theoretical overview of the Markov Logic Networks, with Chapter 7 providing a detailed implementation of MLN using Alchemy. Chapter 8 reports a detailed analysis of the results obtained using the two techniques. Chapter 9 provides the conclusion obtained from the results, along with the limitations of using this technique. It is concluded by stating the future prospect and enhancements that can be made using this approach.

Chapter 2. LITERATURE SURVEY

Understanding and extracting important information from a given corpus of documents has been a wide area of interest and goal of AI and Natural Language Processing (NLP). This area has seen significant development using ML algorithms. But it has been observed that extracting information i.e. extracting entities and relations between them using ML algorithms is complex with limited accuracy and uncertain output. In this chapter, a brief review of the literature in entity and relation classification using ML has been presented. Comparatively, the idea of using Markov Logic Network to boost the performance of ML algorithms is recent, and the same has been explored in this work.

2.1 The Classical Approach

Giannis Bekoulis et.al have proposed a joint entity recognition and relation extraction as a multi-head selection problem in [13]. They created a joint neural model that recognizes entities as well as extracts relations concurrently without the need for manual features or any other tool. The model for recognizing entities was created by them using conditional random fields layer, whereas for extracting relations they have proposed the task as a multi-head selection problem. They claimed that their model is superior to other neural models and it automates the process of extracting features.

Goliang Ji et.al [14] have worked on a distant supervision model for relation extraction with the help of sentence-level attention and entity descriptions. They created an attention model that selects valid instances and uses the information from knowledge bases. The entity descriptions have been extracted from the free base and Wikipedia which forms the background knowledge for their task.

End to end relation extraction using LSTMs on sequences and tree structures has been proposed by Makoto Miwa and Mohit Bansal in [15]. They have created an LSTM-RNN model which is based on stacking bidirectional tree structure to represent both entities and their relations by sharing parameters with the help of a single model. They also worked on entity detection while training and used the entity information for

extracting relations using scheduled sampling. They have reported an error reduction of 12.1% and 5.7% in the F1 score on ACE 2005 and ACE 2004 dataset respectively.

2.2 MLN for entity and relation extraction

Mathew Richardson and Pedro Domingos [16] coined the word Markov Logic Network (MLN) for the first time as a first-order knowledge base with weights attached to each clause. This first-order network combined with constants representing objects in the domain specifies a Markov network which has one feature for every possible ground truth of a first-order formula. They have performed inferencing by using MCMC on the subset of the ground network.

The concept of combining logical and statistical AI was proposed by Pedro Domingos et.al in [17]. Weights attached to First-order formulae from AI were considered as templates for Markov networks. They tested this combined approach of Markov logic for entity resolution, link prediction, information extraction, etc. and used the open-source Alchemy tool to validate the experimental results.

Sachin Pawar et.al [18] have worked on end to end relation extraction using MLN. They proposed a simultaneous approach for extracting entities and their relations in a sentence by using inference in MLN. They used three different classifiers, local entity classifier and pipeline relation classifier which uses predictions of the local entity classifier and presented experimental results on the ACE 2004 dataset.

2.3 Scope of the work

After going through these different approaches for entity and relation classification in the literature, it was observed that machine learning can be considered as the classical approach for this task since it has been used extensively for this purpose. However, its performance has scope for improvement. Though the use of MLN combined with ML has shown some promising results, it has not been explored much for entity and relation extraction. The results of entity and relation classification can be improved upon using this approach. This dissertation aims to use such a model consisting of ML combined with MLN.

Chapter 3. PROBLEM STATEMENT

3.1 Detailed Problem Statement.

Title:

Ontology Guided Information Extraction using Markov Logic Networks.

Description:

In this project initially, Markov Logic Networks was explored for information extraction. The domain ontology i.e. entities and relations between them were given as input. Additionally, seed instances of these ontology elements were provided. The task was to extract additional instances as well as relations between them given a corpus of documents. Markov Logic Network (MLN) is used to define probabilistic first-order logic rules. These rules codify domain knowledge about the entities and their relations. Once the MLN for a given ontology was created, the extraction (at a sentence level) step involved performing inference over specific predicates of the MLN to obtain their final values.

3.2 Approach

As mentioned above, the task of information extraction from the given corpus of documents was performed using a two-fold technique:

1. **Classification of Entities and Relations using ML:** Two ML models were built for Entity and Relation Classification respectively. Each document in the corpus was first fed to the Entity Classifier which associated an Entity Tag for each word in the document. The documents were annotated using these Entity Tags and fed to the relation classifier to extract the exact relations present in the document. Both these classifiers were built using traditional machine learning algorithms. The output of the classifier was the foundation upon which the Markov Logic Network was built.
2. **Classification of Entities and Relations using MLN:** A Markov Logic Network was built by codifying the domain knowledge as well as the information obtained from the results of the Entity and Relation Classifier using First-Order Logic Rules. These weighted rules were used to extract any additional entities or

relations in the documents, which were previously misclassified or neglected by the classifier, to improve upon the overall task of information extraction.

The following figure, Fig. 3.1 gives the WorkFlow of the tasks involved in information Extraction.

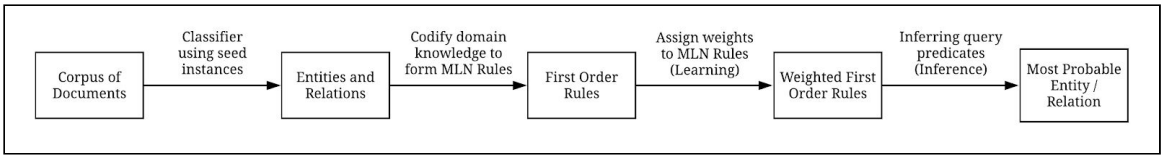

Fig. 3.1 Workflow of Tasks Involved in Information Extraction

3.3 Aim

The primary aim of this project was to combine First Order Logic rules of AI with Probabilistic Graphical Models to obtain a Markov Logic Network that would be used to extract additional information from the documents that were previously missed by the classifier.

It was observed that, although the classifiers were trained to their maximum accuracies, the Relation Classifier had missed several relations present in the document and classified them as “None”. These results of the classifiers would be fed to the MLN for further improvements. MLN using the weighted First Order Rules would correct the results involving documents where an essential relation existed which was previously misclassified as “None” by the relation Classifier.

The underlying goal of this project was to develop a platform to obtain all the essential relations from a document, first using the traditional machine learning algorithms, then using the weighted rules of MLN to obtain more accurate results.

3.4 The CoNLL Dataset

The dataset used throughout the project was the CoNLL04 Dataset. There are four entity types in the dataset (Location, Organization, Person and Other) and five relation types (Kill, Live_in, Located_in, OrgBased_in, and Work_for). The dataset consists of 1153 training instances and 288 instances for testing. Each sentence (i.e. a document) in the dataset is tokenized and a tag is allotted to each token. These tags could be:

1. Peop for Person
2. Loc for Location
3. Org for Organization
4. O or Other for Other.

They are prefixed using BIO Encoding Scheme. As each entity consists of multiple-sequential tokens, the B-type (beginning) is assigned to the first token in the entity, the I- type (inside) to every other token within the entity and the O tag (outside) if a token is not part of an entity. In Fig. 3.2, the third column shows the BIO encoding tags assigned to the tokens of the sentence in the CoNLL Data.

```
#doc 4102
0       Kakizawa        B-Peop  ['N']   [0]
1       suggested       O       ['N']   [1]
2       that    O       ['N']   [2]
3       Hata    B-Peop  ['N']   [3]
4       's      O       ['N']   [4]
5       remarks O       ['N']   [5]
6       expressed       O       ['N']   [6]
7       anticipation    O       ['N']   [7]
8       of      O       ['N']   [8]
9       the     O       ['N']   [9]
10      truth   O       ['N']   [10]
11      of      O       ['N']   [11]
12      claims  O       ['N']   [12]
13      by      O       ['N']   [13]
14      North   O       ['N']   [14]
15      Korean  O       ['N']   [15]
16      President       O       ['N']   [16]
17      Kim     B-Peop  ['N']   [17]
18      Il-song I-Peop  ['Live_In']     [21]
19      that    O       ['N']   [19]
20      North   B-Loc   ['N']   [20]
21      Korea   I-Loc   ['N']   [21]
22      has     O       ['N']   [22]
23      neither O       ['N']   [23]
24      the     O       ['N']   [24]
25      will    O       ['N']   [25]
26      nor     O       ['N']   [26]
27      the     O       ['N']   [27]
28      ability O       ['N']   [28]
29      to      O       ['N']   [29]
30      develop O       ['N']   [30]
31      nuclear O       ['N']   [31]
32      weapons O       ['N']   [32]
33      .       O       ['N']   [33]
```

Fig. 3.2 Format of a document in CoNLL dataset

Consider Fig. 3.2. The last two columns represent the relation that exists between the entities. These can be:

1. Work_For
2. Live_In
3. Located_In
4. OrgBased_In
5. Kill
6. None

The second last column specifies the relation that exists between the current entity and the one mentioned in the last column. An entity may be related to more than one entity in which case the last two columns are tuples specifying the relations and the token number of the corresponding related entity. For example, the entity with token number 18 and type I-Peop (in Fig. 3.2) is having a Live_In relation with the entity at token number 21 which is of type I-Loc. Also, only the relations between the sequentially last token in the entities are considered. As in the above example, it can be seen that the I-type of the entity Peop (at token 17 and 18) is related to the I-type of the entity Loc (at token 20 and 21). If there is only a B-type (entity has a single token) in an entity then the B-type was considered in the relational mapping. Consequently, if there are more than one I-types in an entity, only the last I-type was considered.

Fig. 3.3 summarizes all the entity tags in the CoNLL dataset and how they are related to other entities.

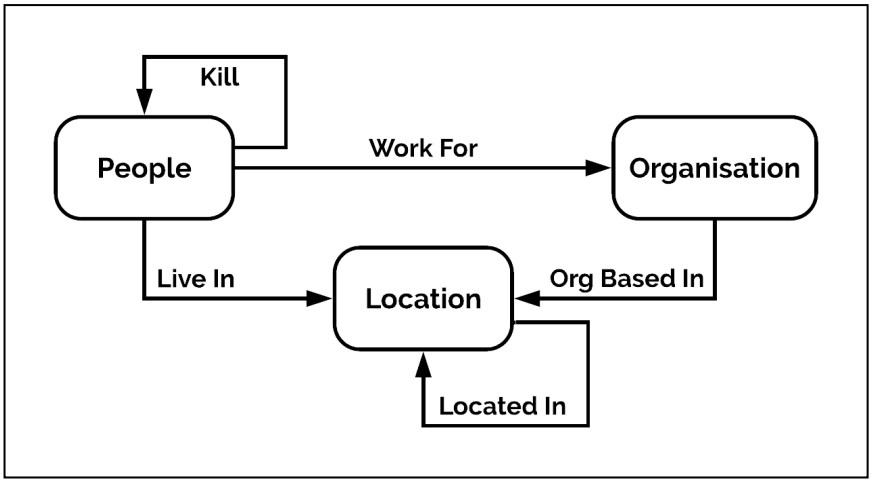

Fig. 3.3 Entity and Relation Tags in CoNLL Dataset

Chapter 4. ENTITY CLASSIFIER

4.1 Introduction

Named Entity Recognition and Classification (NERC) is a process of recognizing information units like names, including person, organization, location, and "Other" names. For such Entity Classification, an Entity Classifier (Base Classifier) was built using traditional machine learning algorithms that were trained on the CoNLL Dataset. The words(tokens) in each document were classified into the following classes:

People, Organisation, Location, Other.

The Entity Classifier would form the Benchmark using which the MLN model was trained and tested as discussed in Section 7.2.

4.2 Data preprocessing

For the Entity Classifier, the CoNLL Dataset was used for training and testing. It was observed that the first three columns denoted the position of Word in the sentence, the actual word and the corresponding Tag which was labeled as per the BIO encoding scheme explained in Section 3.4. Thus, for each document (sentence) the platform had extracted the following Word-Tag pairs and formed two arrays of Words and corresponding Tags for that document. This process was repeated for every document which constituted the input to the ML model.

4.2.1 Mapping of Words and Tags to integers

As an additional utility step to reduce memory usage, each Word was mapped to a unique integer. This mapping (dictionary) along with the reverse mapping (to get back the word from integer) was stored, which were later used when the results were displayed. The same process was done for the tags to get a Tag-integer mapping. Two special tokens were also added to this mapping. These included the "PAD" token (padding token) which mapped to the integer 0 and the "UNK" token (unknown token) which mapped to integer 1.

4.2.2 Uniform Length of Documents

Each document had different lengths. The length of the document was fixed to 100 words. Some documents were padded with a “PAD” token (which maps to integer 0) if their length was less than 100 words. The corresponding Tag arrays for the documents were also padded using the “PAD” token to maintain consistency (same length for Word and Tag arrays representing the document).

4.2.3 Forming the Character Arrays

Along with the document arrays, character arrays were formed corresponding to each word in the document. The significance of these character arrays is stated in Section 4.3. The length of the array was fixed to 10 characters. These characters just like the Words were mapped to unique integers and these mappings were also stored.

Similarly, character arrays were padded using the “PAD” token to make uniform arrays of constant lengths of 10 characters.

4.2.4 Significance of the Special tokens:

The “PAD” token was used for padding the words array as well as the character arrays to create uniform sized input for the machine learning Model. The mappings between Word-integer and Character-integer were stored, which were used as mappings for the test data. During testing the words or characters that were not present in the above mapping, were labelled as unknown or “UNK” and were mapped to the corresponding integer of the “UNK” token.

4.3 Model

As stated above the data was sequence-based which implied certain patterns or positioning of words in the sentence were important features to get the results.

The crux of the machine learning model [19] was based on a special type of RNN (Recurrent Neural Network) known as LSTM (Long Short Term Memory).

There are certain disadvantages of using ANN:

1. Works for fixed-size I/p, O/p
2. Does not capture the pattern in the sequences, Not suitable for time series or sequence-based data

RNN overcomes these disadvantages by using the feedback mechanism in which the output is dependent upon the current as well as the previous inputs. Hence, RNN was chosen for the purpose. It was observed that RNN suffer from the problem of vanishing /exploding gradients. LSTM overcomes this problem using the Forget Gate inside the LSTM cell.

4.3.1 Architecture

The model was structured using Keras library of TensorFlow. The different Keras layers used in the model are as follows: (**Note**: The workflow of the Keras layers stated below is for a single input document.)

1. **Word Embeddings**
    1) ***Input(100,)***: Initial number of nodes in the neural network. The input would be a document that contained 100 words.
       Returned Tensor Shape (100,1)
    2) ***Embedding Layer(Total Words, embedding_dim = 50, embedding_matrix, trainable = false):*** Pre-trained embeddings Word2Vec of Google which contained vectors of float values for each Word of dimension 50 as the embedding matrix, were provided.
       Returned Tensor Shape (100,50)
2. **Character embeddings:**
    1) ***Input(100,10):*** A 2D array where each document had 100 words, each Word had 10 characters.
       Returned Tensor Shape (100,10)
    2) ***Time Distributed Over Embedding Layer***: In the time distributed layer, a time step was equivalent to each word. This implied each of the 100 words were parallelly distributed over the EmbeddingLayer(Total characters,

embedding_dim = 10, trainable = true). Thus, for each character output was (10,10).

Returned Tensor Shape (100,10,10)

3. **Time Distributed One Directional LSTM - *LSTM(hidden_units = 20, recurrent_dropout, return sequence =false)*:** LSTM layer took a 2D input while learning sequences. The LSTM learned 10-character sequences of a word using 10 LSTM cells, where each cell was given an input X (character embedding of dim 10). With return sequence = false only the last LSTM cell returned the Output Y value (hidden units) = 20 (o/p dim.). Since the LSTM layer was wrapped around the time distributed layer this process occurred parallelly distributed over the 100 words in the document.

   Returned Tensor Shape (100,20)

Fig. 4.1 shows the overview of forming the Word and Character Embeddings

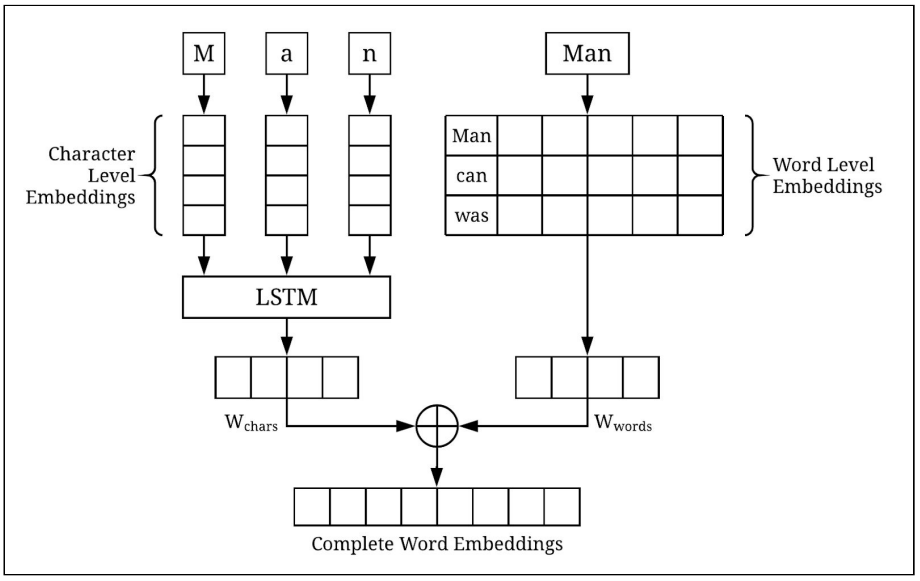

Fig. 4.1 Embedding Layer in detail

The two arrays obtained i.e. Word Embeddings and Character Embeddings are concatenated to form an array of (100,70) which is fed to the Next BiLSTM layer.

Main Layers of the Model

1. **Bidirectional LSTM(hidden_units = 50, recurrent_dropout, return sequence =true)**: For 2D input the number of LSTM cells were 100 with input dimension

for each cell as 70 (combined word and character embeddings). With the return sequence set true each LSTM cell returned Output Y (o/p) with dim 50.

Returned Tensor Shape (100,50)

2. **Time Distributed Wrapped around a Dense Layer:** 50 nodes returned from LSTM were mapped to the 9 output nodes (In this case including the BIO encoding there were 9 tags as follows: B-Loc, I-Loc, B-Peop, I-Peop, B-Org, I-Org, B-Other, I-Other, O) using the Dense(Fully Connected) layer. As done previously Time Distributed layer would distribute this process parallelly across all 100 words.

   Returned Tensor Shape (100,9)

3. **Activation Function Softmax (average of exponential e):**

   Returned Tensor Shape (100,9)

The model returned an output array of dim (100,9), which was used to predict the correct tag for each of the 100 words by taking argmax from the 9 float values.

Fig. 4.2 shows the overview of the layers used in the Entity Classifier.

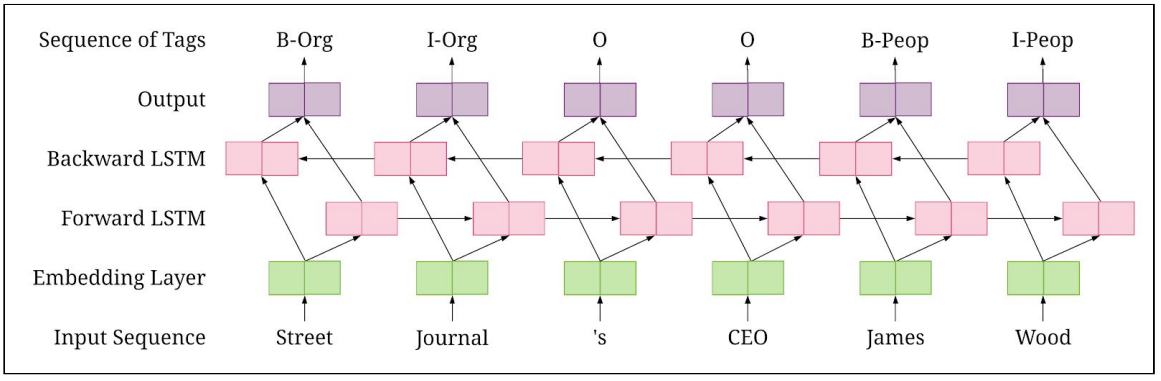

Fig. 4.2 Overview of the Layers in the Entity Classifier Model

As shown a document is given as the input sequence to the model which predicts the output Tag for each Word.

4.4 Training the Model

The ML model was trained using the following parameters:

Optimizer: Adam Optimiser (adaptive learning rate optimization Algorithm)

Loss function: Cross-Entropy Function to Calculate Loss

Epochs: 100

Batch Size: 32

Input: 1153 training documents of the CoNLL train dataset (*train.txt*) in the form of a text file were provided as input to the model for training.

Fig. 4.3 and 4.4 shows the variation of accuracies and losses of the model with respect to the number of epochs for training

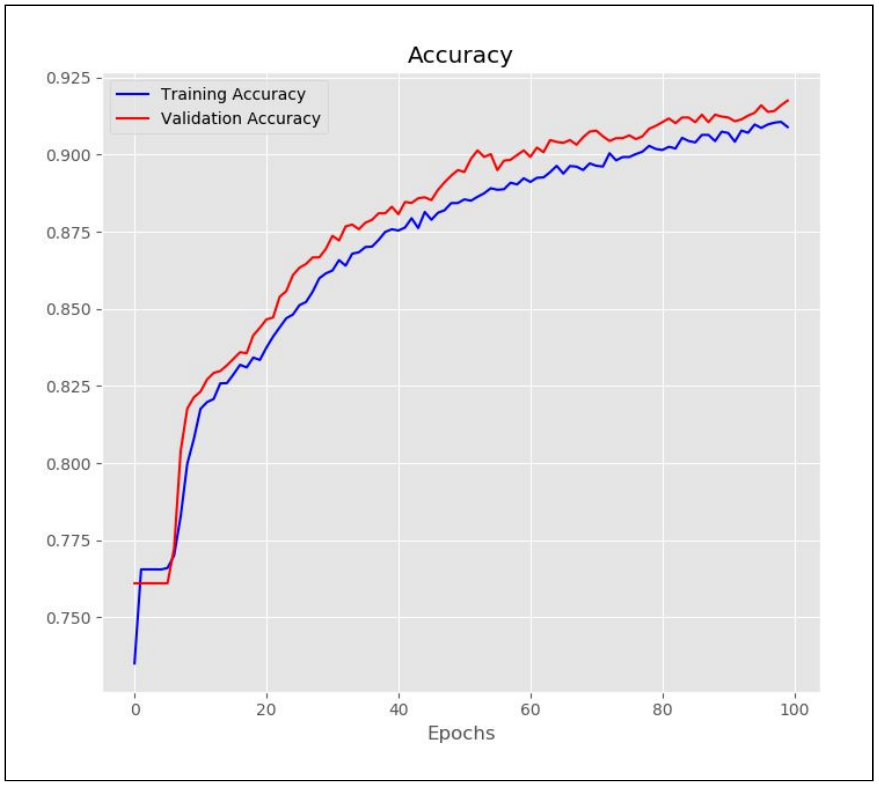

Fig. 4.3 Accuracy of the model (Entity Classifier)

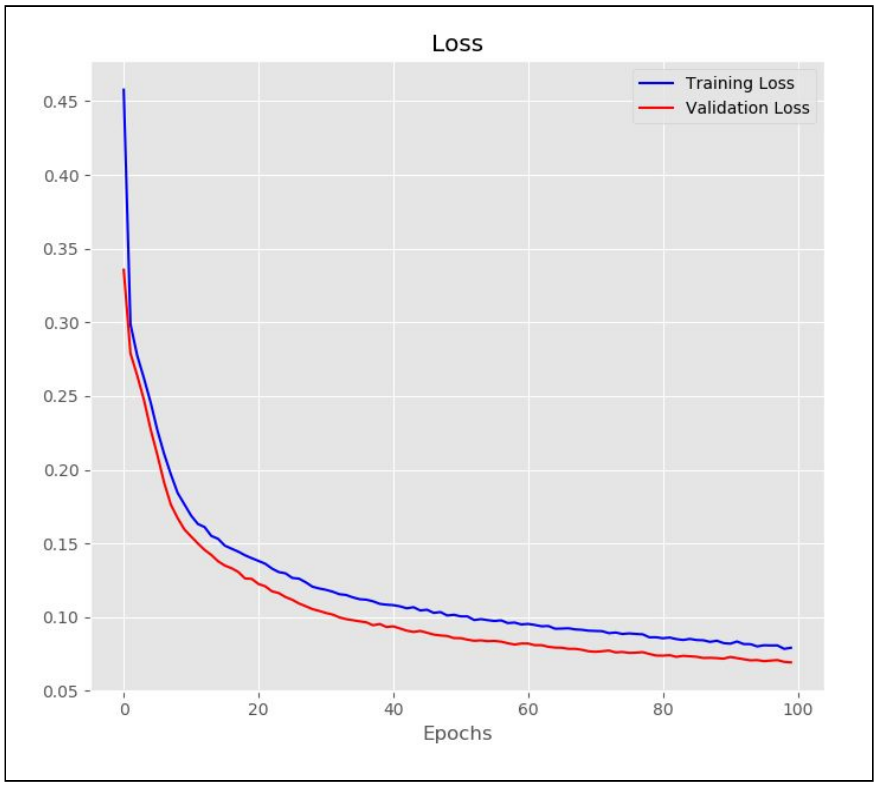

Fig. 4.4 Loss of the model (Entity Classifier)

Output: The predicted tag along with the "Gold Truth" tag of the Word was stored in the *base_classifier_train.csv* as shown in Table 4.1. These results were used in Section 7.2.2 during the training of the MLN model.

Table 4.1 Output of Entity Classifier (train.csv)

SentenceID	tokenID	Word	Predicted	Gold Truth
1	11	Ford	B-Org	B-Org
1	12	Theatre	I-Org	I-Org
1	13	in	O	O
1	14	Washington	B-Loc	B-Loc

4.5 Testing the Model

Input: 288 test documents of the CoNLL test dataset (*test.txt*) were provided as input in the form of a text file.

Output: The platform generated the following classification report (as shown in Table 4.2) to provide a detailed analysis of the results obtained after testing.

Table 4.2 Classification Report (Entity Classifier ML)

Entity Tag	Precision	Recall	F1 Score	Support
B-Loc	0.736	0.817	0.775	427
B-Org	0.691	0.621	0.654	198
B-Other	0.818	0.545	0.655	132
B-Peop	0.752	0.707	0..729	321
I-Loc	0.838	0.654	0.734	205
I-Org	0.613	0.755	0.677	241
I-Other	0.860	0.630	0.727	127
I-Peop	0.786	0.905	0.841	369
O	0.972	0.972	0.972	6369
Accuracy			0.917	8389
Macro Avg	0.785	0.734	0.751	8389
Weighted Avg	0.919	0.917	0.916	8389

The platform generated a *base_classifier_test.csv* in the same format shown in Table 4.1 to store and display the results. These results were used in Section 5.5 and Section 7.2.3 during testing of the Relation Classifier and MLN model on the same test dataset.

Chapter 5. RELATION CLASSIFIER

5.1 Introduction

Relation classification is an important task in natural language processing fields. In this project, a Relation Classifier was built that predicts a relation between two entities marked in a sentence. These entities were marked/annotated in the sentence using their corresponding tags mentioned in the CoNLL Dataset.

Thus, for a given annotated sentence, the Relation Classifier predicted the following relation between the two marked Entities:

Kill, Work_For, Live_In, OrgBased_In, Located_In, None.

Note: A sentence can have several marked Entities, the Platform took the several combinations of these Entities, two at a time, and returned the corresponding relation between them.

The Relation Classifier would form the Benchmark using which the MLN model would be trained and tested as discussed in Section 7.2.

5.2 Data Preprocessing

The Relation Classifier used the CoNLL Dataset for training and testing. As was observed in the last column of the CoNLL Dataset, for each Entity, the relations that it has with other Entities were stated. In case no relation existed for that Entity it was specified as ‘N’. The CoNLL Dataset was parsed to obtain annotated sentences (where each sentence only the two Entities are annotated) and the corresponding relation present between the annotated entities in the sentence.

Consider the example given below depicting *Live_In* relation:

Sentence: Kakizawa suggested that Hata 's remarks expressed anticipation of the truth of claims by North Korean President Kim <I-Peop> Il-song </I-Peop> that North <I-Loc> Korea </I-Loc> has neither the will nor the ability to develop nuclear weapons.

Relation: Live_In

Also, there are several Entities that do not have any relation between each other. These sentences are also formed where it is specified that a “None” relation exists between the annotated entities.

Consider the example given below-depicting *None* relation:

Sentence: Hatcher also fled to the Onondaga <I-Loc> territory </I-Loc> but has since moved to a Shoshone-Bannock reservation in <B-Loc> Idaho </B-Loc>.

Relation: None

Thus, as a part of pre-processing these sentences along with their relation tags to be fed to the machine learning model are obtained.

The Data Preprocessing Steps mentioned in Section 4.2 are repeated which include Mapping of the word to integer, padding of documents to create sentences of Uniform Length (100 words in this case), and including the Special Token “PAD” and “UNK”.

Note: Character Arrays were not created, hence any Character Embeddings were not used in the Relation Classifier model.

5.3 Model

The crux of the Machine Learning model [20] was based on a special type of RNN (Recurrent Neural Network) known as LSTM (Long Short Term Memory) for the reasons mentioned in Section 4.3.

Along with this, the Model consisted of an Attention Weighted Layer to enhance the ability of the Model to learn long sequences which were a disadvantage, in standard sequence to sequence models using encoder and decoder. This technique gave more weightage to certain words (features) in the sequence to obtain a more accurately trained model.

5.3.1 Architecture

The model was structured using Keras library of TensorFlow. The different Keras layers used in the model are as follows:

1. **Word Embeddings - Embedding(Total_words, input_length=100, trainable=True):** Initial number of nodes in the neural network. The input to this layer would be a document that contained 100 words.

   Returned Tensor Shape (100,50).

2. **BiLSTM Layer - Bidirectional(LSTM (units=64, return_sequences=True, recurrent_dropout=0.68):** Based on the 2D input the number of LSTM cells were 100 where each cell was given input of length 50 and returned an output vector Y of length 64(no of hidden units). Since the return sequence was set to true, each LSTM cell returned an output vector.

   Returned Tensor Shape (100,64).

3. **Attention Weighted Layer:** The output of the BiLSTM layer was an array of shape (100,64). The array was processed in the following manner.

   Let H be a matrix consisting of output vectors $[h_1, h_2, \ldots, h_T]$ that the LSTM layer produced, where T was the document length. The shape of H was (100,64).

   The representation r of the document was formed by a weighted sum of these output vectors as shown below:

   ***Calculating Weights of each LSTM Output*:**

   $M = tanh(H)$

   w is the weight vector of Shape (100,1) which was randomly initialized. It denoted the weight for each word in the document.

   $\alpha = softmax(w^T M)$. The shape of α is (1,64).

   α was a vector where each value denoted the final weightage of that LSTM output w.r.t to all 50 words after normalizing it using Softmax.

   Calculating the Weighted Sum of the LSTM output:

   $r = H\alpha^T$

   (Multiply and calculate the weighted sum using the weights obtained from the Softmax layer).

   Returned Tensor Shape (100,1).

4. **Dense Layer(units=6, activation="softmax"):** Mapping of 100 values returned from the attention weighted layer with the output layer, having 6 nodes (no of relation classes i.e [“ Work_for”, ”Live_In”, “Located_In”, “OrgBased_In”,

”Kill”, ”None”]), was done using the Dense (Fully Connected) Layer. The softmax was the activation function.

Returned Tensor Shape (6,1).

The output from the model was used to predict the correct relation by taking maximum out of the 6 values returned by the dense layer. Fig. 5.1 gives an overview of the Relation Classifier model.

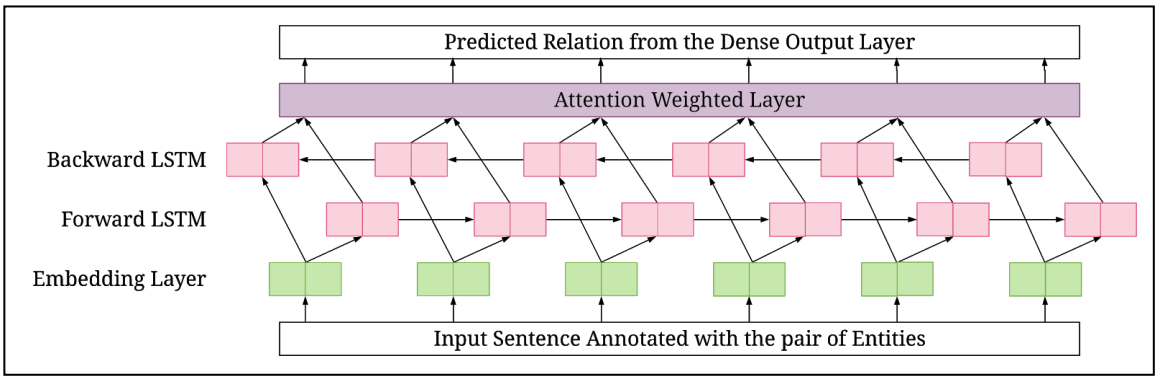

Fig. 5.1 Overview of the Layers in Relation Classifier Model

5.4 Training the Model

The machine learning model was trained using the following parameters:

Optimizer: RMS Optimiser (adaptive learning rate optimization Algorithm)

Loss function: Cross-Entropy Function to Calculate Loss

Epochs: 50

Batch Size: 32

Input: 1153 training documents of the CoNLL train dataset (*train.txt*) in the form of a text file were provided as input to the model for training.

Fig. 5.2 and 5.3 shows the variation of accuracies and losses of the model with respect to the number of epochs for training

Output: The predicted relation along with the “Gold Truth” relation of the sentence was stored by the platform in the *relation_classifier_train.csv* as shown below in Table 5.1. This file would be required during the training of the MLN model as stated in Section 7.2.2.

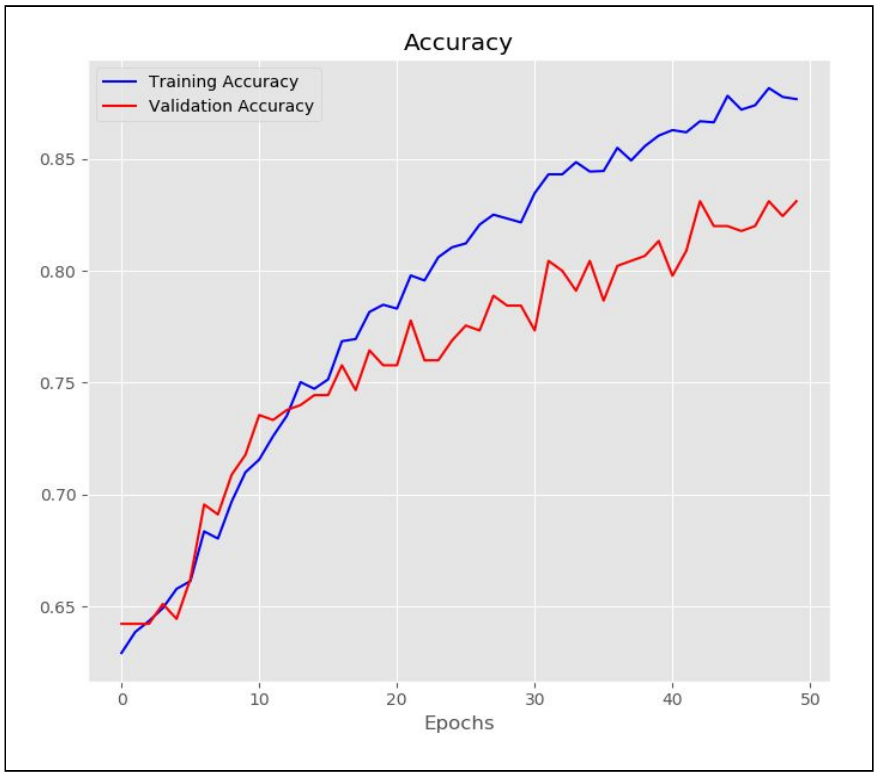

Fig. 5.2 Accuracy of the model (Relation Classifier)

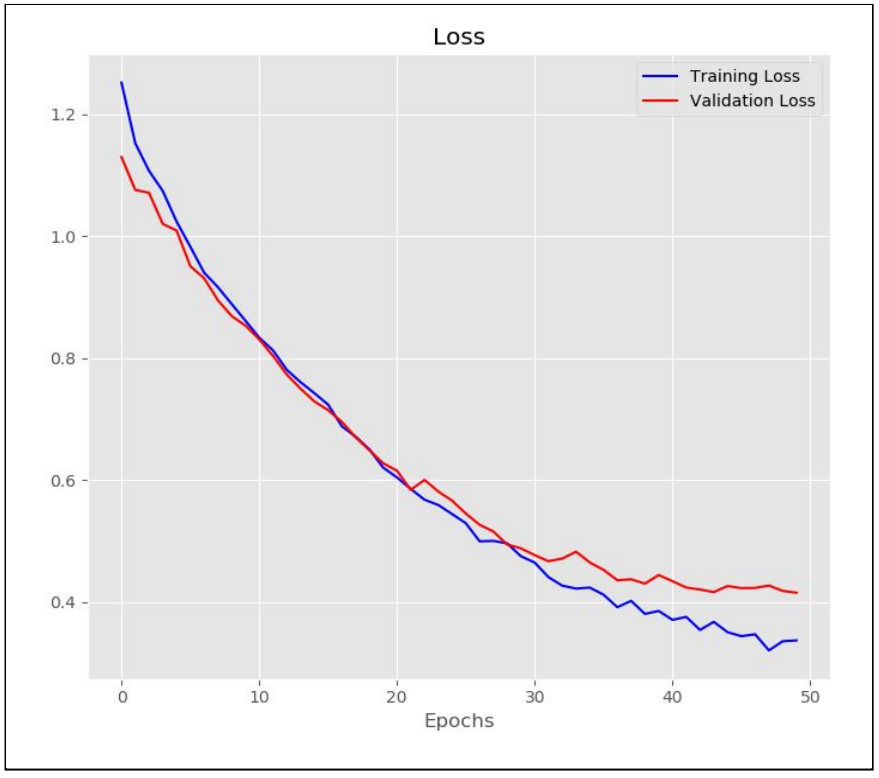

Fig. 5.3 Loss of model (Relation Classifier)

Table 5.1 Output of Relation Classifier (train.csv)

SentenceID	token1	token2	Sentence	Predicted	Gold Truth
1	17	22	Very strong south winds accompanied the storm system, with 50-to 70-mph wind gusts reported near Grande <I-Loc> Isle </I-Loc> and St. Albans, <B-Loc> Vt. </B-Loc>, blowing down a large radio tower and causing several power outages.	Located_In	Located_In
1	20	22	Very strong south winds accompanied the storm system, with 50-to 70-mph wind gusts reported near Grande Isle and St. <I-Loc> Albans </I-Loc>, <B-Loc> Vt. </B-Loc>, blowing down a large radio tower and causing several power outages.	Located_In	Located_In

5.5 Testing the Model

Input: 288 test documents of the CoNLL test dataset (*test.txt*) were provided as input in the form of a text file.

Note: For each sentence, the two entities were annotated using the Predicted Tags of the Base Classifier. These were read from the *base_classifier_test.csv* that was generated as mentioned in Section 4.5. This was done in order to obtain a cumulative result of both since the relation classifier depends on the output from the base classifier.

Output: The platform generated the following classification report (as shown in Table 5.2) to provide a detailed analysis of the results obtained after testing.

The platform generated a *relation_classifier_test_base.csv* in the same format as shown in Table 5.1 to store and display results. This file would be required during testing of the MLN model on the same test dataset as stated in Section 7.2.3.

Table 5.2 Classification Report (Relation Classifier ML)

	Precision	Recall	F1 Score	Support
Kill	0.38	0.74	0.51	47
Live_In	0.48	0.49	0.48	100
Located_In	0.39	0.77	0.51	94
None	0.83	0.65	0.73	704
OrgBased_In	0.62	0.68	0.65	105
Work_For	0.46	0.49	0.47	76
Accuracy			0.64	422
macro avg	0.53	0.63	0.56	422
Weighted avg	0.70	0.64	0.65	422

Chapter 6. MARKOV LOGIC NETWORKS

In this chapter, the Markov Logic Networks [21] shall be discussed at length, briefly describing the algorithms used for inference and learning.

6.1 Introduction

For a long time, one of the goals of AI was to combine probabilistic representation with first-order logic. Uncertainty in the real-world problems is handled efficiently using Probabilistic graphical models. A wide range of knowledge can be represented using first-order logic. To effectively handle the scenarios in the real world, a combination of both is required.

Markov Logic Network (MLN), which is a probabilistic network, enables inference by applying the ideas of Markov Network to first-order logic. Markov Logic networks provide a generalization for first-order logic. Within certain limits, it assigns all unsatisfiable statements a probability of zero, and all tautologies, a probability of one. Thus, Markov logic is a probabilistic extension of finite first-order logic [22,23]. Logic can easily specify relations among many entities. However, the major shortcoming of FOL is its inherent property of rigidity by the virtue of which, in the presence of noise and uncertainty, the model considers only the true and false aspects of a problem. Markov logic's main concept is to soften the first-order formulae by assigning weights to them, thus overcoming the rigidity.

More formally, a Markov Logic Network (MLN) can be viewed as a first-order knowledge base, i.e. a collection of first-order rules. In the knowledge base, each clause is assigned a weight. It defines a set of constants, where each constant represents an object in the domain. The first-order formulae in the knowledge base are grounded using this set of constants and a weight is then assigned to each of them. Using this, a ground Markov network is constructed by MLN, which contains one feature for each possible grounding.

Markov logic stands as a leading framework amongst the numerous methods proposed for Statistical Relational Learning [24]. Markov logic comprises both first-order logic and Probabilistic Graphical Models. The various dependencies that are

expressed by other approaches are restricted since they use only a subset of first-order logic features, in contrast to the Markov logic approach. This enables a simple transition from other approaches to the Markov logic Framework.

Markov logic is the most advanced when compared to other approaches. It uses algorithms like MC-SAT for inference, and lifted belief propagation for assigning weights to the formulae and predicates that are available during learning. Alchemy is the most complete open-source software implementation with all the existing algorithms for Markov logic, out of the several other implementations available.

6.2 Background

First-order logic lets you represent a complex domain with several objects and constants [25]. A term is a variable or a constant representing an object in the domain, such as Peop and Org. An atomic formula or an atom is a predicate symbol applied to a tuple of terms that describes relations, such as Etype(sid,tid,token) and Rtype(sid,tid,tid,rel). FOL Formulae are constructed from atomic formulae using logical connectives ($\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftrightarrow$) and quantifiers ($\forall$, $\exists$).
Below are two example formulae:

1. $\forall tk\ Etype(s,t,tk) \Rightarrow EFtype(s,t,tk)$

This formula states that if token *t* in sentence *s* is of entity type *tk* as per weak evidence, then this implies that the final entity type of that token is *tk.*

2. $\forall t1,\ t2\ EFtype(s,t2,Org) \wedge Rtype(s,t1,t2,Work\_For) \Rightarrow EFtype(s,t1,Peop)$

This formula states that if in a sentence s, the final entity type of token t2 is Org and there is weak evidence that there is a relation between tokens t1 and t2 which is Work For, then the final entity type of token t1 is Peop.

A set of FOL formulae is known as a first-order knowledge base (KB). A ground term contains variables that are initialised using constants. A ground atom or ground predicate is an atom whose arguments are ground terms. A ground formula is a formula that contains only ground atoms.

Probability is the tool of choice for managing ambiguity. A common representation of probabilistic modeling is a Probabilistic Graphical Model, which by exploiting conditional independence, can compactly define probability distributions. A Markov network (i.e. Markov random field) is an undirected graphical model that defines a joint distribution over a set of variables $X = (X_1, \cdots, X_n)$ as follows [26]:

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

where Φ_k is a potential function defined on a subset of variables, and $x_{\{k\}}$ is the state of the variables in that subset.

$$Z = \sum_{x \varepsilon X} \prod_k \Phi_k(x_{\{k\}})$$

Z is the normalization constant, also known as the partition function. Markov networks are commonly represented as log-linear models, where the potential function is an exponentiated weighted sum of the state features as follows:

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

where f_j denotes the Features, w_j the Weights, and x is the state.

6.3 Representation

A Markov Logic Network can be represented as follows:

A Markov logic network is a set of first-order formulae having weight associated with each formula. Together with a set of constants, it defines a Markov network with one variable per ground atom and one feature per ground formula. The weight of a feature is the weight of the first-order formula that it originated from. The probability of a state x in such a network is given by

$$P(x) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right)$$

where Z is the normalization constant as in Markov networks, w is the weight of the i^{th} formula, and $n_i(x)$ denotes the features that is the number of its true groundings generated from the first order formula given a state x.

Higher the weight, stronger is the probability that the domain information represented by the formula is true. With infinite weight, the formula becomes a hard constraint. For example, in Alchemy, an assigned infinite weight is twice the maximum of the weights given to the formulae. First-order logic is a special case with all weights being infinite. As the number of constants increases, the size of the grounded Markov network grows exponentially with respect to the number of variables in a formula, but the Markov logic has its compactness maintained.

6.4 Inference

Inference can be understood as computing the answer, when a query is provided, in a given probabilistic model. For a given evidence (i.e., a partial truth assignment of the variables), the most probable state is determined using maximum a posterior (MAP) inference technique. On the other hand, the conditional probability for a formula to be true i.e. the conditional likelihood of a result being valid is calculated using marginal inference. Similar to graphic models, inferences in Markov logic have an exponential time complexity and a high cost of computation, due to the number of possible states [27].

Hence, approximate inference algorithms are used. The marginal inference uses the standard approach of Markov chain Monte Carlo (MCMC). In this method, states are sampled by constructing a Markov chain and the probabilities are computed using the samples [28]. Gibbs Sampling is one of the most common MCMC algorithms. In this algorithm, the next state of a variable is sampled iteratively and the current value of other variables is kept fixed. The drawback of this algorithm is that in the presence of logical constraints the state space is broken into disjoint regions. In such a scenario, it becomes difficult to switch between the regions using a local update scheme. Due to this limitation, Gibbs Sampling returns incorrect probabilities as it gets stuck in the initial region. The MAP inference problem of Markov logic can be simplified to determining the assignments to the predicates such that all the FOL formulae are satisfied. The assignments should be such that the sum of weights of satisfied clauses is maximized. In

Markov logic, the MAP inference problem is essentially a weighted SAT problem. A Weighted SAT solver such as MaxWalkSAT can be used for this purpose.

By combining logical and statistical methods, the MC-SAT algorithm was developed [29]. To create a sample for MCMC, a SAT solver is used. MC-SAT is based on the "slice sampling" algorithm of MCMC. Isolated modes are efficiently determined in the distribution by using a satisfiability solver (WalkSAT[30]). This results in the Markov chain getting mixed up very rapidly. This is the major advantage of using WalkSAT. A detailed balance (approximately) is preserved by the slice sampling scheme. MC-SAT has outperformed previous MCMC algorithms like Gibbs Sampling by being orders of magnitude faster than them. This has helped to achieve an efficient sampling on a scale that was previously out of reach.

Pseudocode for MC-SAT can be

Algorithm 1 MC-SAT(*formulae, weights, num_samples*)

$x^{(0)} \leftarrow$ Satisfy(hard *formulae*)
  for $i \leftarrow 1$ to *num_samples* **do**
    $M \leftarrow \varnothing$
    for all $c_k \in$ *formulae* satisfied by $x^{(i-1)}$ **do**
      With probability $1 - e^{-w_k}$ add c_k to M
    end for
    Sample $x^{(i)} \sim \mu_{SAT(M)}$
  end for

All predicates and formulae are grounded by MC-SAT [31]. Therefore, memory and time exponential in the predicate and formula arities (i.e., the number of variables in the predicate or formula) are required. μ_S can be defined as the uniform distribution over set S. All sampled states satisfy the hard clauses, at every step, as they are all selected with a probability 1. The case of negative weights is ignored by the code for keeping it concise. A clause with weight $w < 0$ can be considered equivalent to its negation with weight $-w$, and a clause's negation is nothing but the conjunction of the negations of all of its literals. Thus, it can directly be checked if the clause's negation is satisfied. If it is satisfied, all of its negated literals are selected with probability $1 - e^{w}$, and none are selected with a probability e^{w}.

6.5 Learning

The weight assigned to a formula in the Markov logic network can be learned from the training data. Based on the learning being performed, there are three major learning tasks in Markov logic.

- Weight or Parameter learning: These formulae define the features which constitute the domain knowledge and are devised by domain experts and application developers. The weights are learnt for these formulae with the objective of maximising the likelihood.
- Structure Learning [32,33,34,35]: The Markov logic network is initialized by a KB (which may be empty, populated later). This network then learns both the formula and weights. The goal is to learn features that will construct new formulae and/or modify the existing ones by learning from the data.
- Predicate invention: Along with learning formulae and weights, construction of new predicates by writing simple ones is also learnt. This would enable the designing of formulae using the new predicates to compactly capture more general regularities.

Depending on the type of examples (labeled, unlabeled, or both) used by the systems, learning scenarios can also be classified into supervised, unsupervised, and semi-supervised learning. Although supervised weight learning is the most widely used approach, progress has been made using other types of learning.

Log-Likelihood function

A standard objective of supervised weight learning involves finding the weights that optimize the log-likelihood function $L(x, y) = log\, P(Y = y \mid X = x)$, where Y represents the non-evidence predicates, X the evidence predicates, and x, y their values in the training data. (A closed-world assumption is made, whereby all ground atoms not in the database are assumed false.) A global optimum for this convex optimization problem can be found using the derivative of log-likelihood with respect to its weights:

$$\frac{\partial}{\partial w_i} \; logP_w(X = x) \;=\; n_i(x) \;-\; \sum_{x'} P_w(X = x')\, n_i(x')$$

where the sum is over all possible databases x' , n_i is the number of true groundings of formula i in the training data, and $P_w(X = x')$ is $P(X = x')$ computed using the current weight vector w = (w_1, . . ., w_i, . . .). Stated differently, the i^{th} component of the gradient is the difference between the number of true groundings of the i^{th} formula in the data and its expectation according to the current model. The computation of these estimates involves performing inference over the model, which can be expensive. While we can perform approximate inference using existing Markov Chain Monte Carlo (MCMC) algorithms such as Gibbs Sampling, Richardson and Domingos [6] found its performance unsatisfactory. Instead, they used an efficient alternative to optimize the pseudo-likelihood metric, involving only the probability of each variable given its Markov blanket (graph neighbors) in the data [36] and applying Gaussian prior on all weights to avoid overfitting. Although it is very effective to measure pseudo-likelihood and its gradient because it does not require inference, its parameters can yield poor results if inference is needed over non-neighboring variables.

Unlike, generative learning methods, such as pseudo-likelihood and likelihood, that aim to maximize the joint distribution of all variables, discriminative approaches maximize the conditional likelihood of a set of outputs given a set of inputs (e.g., Lafferty et al. [37]) yielding improved results since no attempt is being made to model the dependencies between the inputs. The voted perceptron is one such discriminative algorithm for labeling of sequence data, that uses the Viterbi Algorithm [38] to perform inference by computing the MAP state.

Chapter 7. ALCHEMY - TOOL FOR MLN

In this chapter, the tool used for MLN learning and inference is discussed in detail. In the project, Alchemy is used as a tool for implementing MLN algorithms. Various aspects of Alchemy, the files, input and output, and the implementation have been explained in the subsequent sections.

7.1 Introduction

Alchemy [39] is a software package or a software tool which provides various algorithms for Statistical Relational Learning and probabilistic logic inference, based on the Markov logic representation. Alchemy has a provision for performing a wide range of AI applications, including collective link prediction, classification, social network modeling, entity resolution, information extraction, etc.

Three major tasks can be performed using Alchemy, namely, structure learning, weight learning, and inference. Learning involves learning the structure or parameters of a given model, using training data, provided in the form of grounded atoms. Whereas inference involves inferring the probabilities of all the possible states (the most probable state having the highest probability value) of the query atoms provided, given test data and weighted rules obtained from learning.

In the project, Alchemy had been used as a black box tool for MLN, where the task was to identify entities and extract relations between them using the predicted entity types and relations obtained from the entity and relation classifiers (machine learning-based) as weak evidence.

7.2 Data Preprocessing

The same CoNLL dataset was used in MLN implementation (Alchemy), as was used in ML-based entity and relation classifiers. The data needed to be processed for training and testing files, which had grounded predicates for weak as well as strong evidence.

7.2.1 Files in Alchemy

Following are the different files used in alchemy for learning and inference. These files are discussed in detail in the further sections on inference and learning.

For training:

1) *univ.mln:* Input file containing MLN predicates and rules
2) *train.db*: Input file containing grounded predicates (evidence obtained from the classifiers and gold truth)
3) *output.mln*: Output file containing the output of Alchemy with weights assigned to each rule.

For testing:

1) *output.mln*: This is the file obtained from learning where alchemy assigns weights to all the rules.
2) *test.db*: Input file containing the grounded predicates (evidence obtained from the classifier)
3) *queries*: Input file containing the queries which are to be inferred.
4) *infer_results.results*: Output file containing the inferred strong evidence along with a probable value of that evidence to be true.

7.2.2 Forming train.db

train.db contained weak evidence (obtained from the classifiers) and strong evidence (Gold truth) predicates. When this along with the FOL rules file (*univ.mln*) was fed as input to alchemy, it assigned weights to the rules after learning. This process is discussed in detail in Section 7.3 on learning. To construct this file, the output from classifiers and gold truth needed to be formatted in the form of grounded predicates.

Alchemy has the following tags for entities:

- Peop for People
- Org for Organization
- Loc for Location
- Other for any other entity type

For relations:

- OrgBased_In
- Live_In
- Located_In
- Work_For
- Kill
- None

Data for *train.db* was obtained as follows:

- The output from entity classifier, that is sentence ID, token ID along with the predicted entity, and the gold truth was stored in a .csv file as mentioned in Section 4.4. This data was used as weak evidence for an entity type, that is, Etype. Corresponding gold truth (the actual tag associated with the entity), EFtype was obtained from the CoNLL dataset. Using this data, grounded predicates for entities were formed, which were required in train.db.

  For example, if the predicted tag for the first token in sentence 2 is People, the grounded predicate is formed as Etype(2, 1, Peop), where the first entry is for sentence ID, the second entry is for token ID and third entry is the predicted type by the classifier (weak evidence).
- In Alchemy, the BIO encoding scheme was not considered. For example, B-People and I-People both would be converted to Peop. This was followed for all the entity types including the Other type.
- Similarly, the weak evidence for the relation type was obtained. Sentence ID, token 1, token 2 and their corresponding relation from the classifier output was stored in a .csv file as mentioned in Section 5.4, which was used to obtain the weak evidence, Rtype.

  For example, if the predicted relation between token 1 and token 7 in sentence 2 is Live_In, then Rtype is formed as Rtype(2, 1, 7, Live_In).

  The first entry shows sentence ID, second and third entries are for token IDs and the last entry is for the predicted relation by relation classifier (weak evidence).
- Gold truths for relations were obtained from the data set and were stored as grounded predicates for RFtype.

The formation of grounded predicates is illustrated below.

For entity:

Table 7.1 csv example for forming train.db (Entity)

Sentence ID	Token ID	Word	Predicted	Actual
1	2	Mumbai	Loc	Loc

The actual tag is obtained from the CoNLL dataset and the predicted tag is the output of the entity classifier.

For this, the grounded predicates would be:

Weak evidence: Etype(1, 2, Loc)

Strong evidence (Gold truth): EFtype(1, 2, Loc)

For relation:

Table 7.2 csv example for forming train.db (Relation)

Sentence ID	Token 1	Token 2	Predicted	Actual
1	2	7	Live_In	None

For this, the grounded predicates would be:

Weak evidence: Rtype(1, 2, 7, Live_In)

Strong evidence (Gold truth): RFtype(1, 2, 7, None)

7.2.3 Forming test.db

test.db was formed in a similar way as train.db, discussed in Section 7.2.2. In this file, just the weak evidence obtained from the classifiers was given, i.e., Etypes and Rtypes. EFtypes and RFtypes were inferred using the *test.db* and the weighted FOL rules.

7.3 Learning Weights

As mentioned in Section 7.2.1, the following files were used for learning:

- *train.db* (with weak and strong evidences)

- *univ.mln* (with FOL rules)

And the weighted FOL rules are obtained in the *output.mln* file.

Predicates and first-order formulae are specified in .mln files. For example, at the top of univ.mln, the predicates Etype, Rtype, EFtype, and RFtype are declared. The first appearance of a predicate or function in a .mln file is taken to be its declaration.

A formula terminated by a period signifies that the formula is 'hard' (i.e., worlds that violate it should have zero or negligible probability). These hard constraints are not specified in the univ file but are placed at the end in the output file generated so that they are considered during inference.

Fig. 7.1 given below shows a sample *univ.mln* file.

```
//predicate declarations
Etype(sid,tid,token)
EFtype(sid,tid,token)
Rtype(sid,tid,tid,rel)
RFtype(sid,tid,tid,rel)

Etype(s,t,Peop) => EFtype(s,t,Peop)
Etype(s,t,Loc) => EFtype(s,t,Loc)
Etype(s,t,Org) => EFtype(s,t,Org)
Etype(s,t,Other) => EFtype(s,t,Other)

Rtype(s,t1,t2,Work_For) => RFtype(s,t1,t2,Work_For)
Rtype(s,t1,t2,Live_In) => RFtype(s,t1,t2,Live_In)
Rtype(s,t1,t2,Located_In) => RFtype(s,t1,t2,Located_In)
Rtype(s,t1,t2,OrgBased_In) => RFtype(s,t1,t2,OrgBased_In)
Rtype(s,t1,t2,Kill) => RFtype(s,t1,t2,Kill)
Rtype(s,t1,t2,None) => RFtype(s,t1,t2,None)
```

Fig. 7.1 Sample univ.mln file

Grounded atoms were specified in the *train.db* file. The creation of this file has been discussed in detail in Section 7.2 on data preprocessing.

7.3.1 Weight learning

To learn the weights of formulae, the ***learnwts*** executable is executed using the following command [40]:

learnwts -g -i univ.mln -o univ-out.mln -t univ-train.db

The learn weights executable is present in the bin folder of alchemy.

Flags used:

-*g*: Generative Learning (alternatively, -d can be used for discriminative learning, generative learning was used in this project)

-*i*: Input File

-*o*: Output File

-*t*: Train File (more than one .db file can be specified by a comma-separated list)

Generative learning: Generative learning [41] is based on generative model which is a statistical model of a joint probability distribution, i.e., given an observable variable X and a target variable Y, it is a joint probability distribution on $X \times Y$, $P(X, Y)$. Symbolically, for observable X and target variable y, it can be defined as $P(X \mid Y = y)$.

For learning, 1153 train documents of the CoNLL dataset were used. The weak evidences (i.e. Etypes and Rtypes) and the gold truths (i.e EFtype and RFtype) were taken from the classifier result csv files for each document to form its train.db file. Thus, in total 1153 train.db files were generated.

Learning for Alchemy occurs in an iterative manner such that the output.mln (containing FOL rules) generated for the ith document is given as input for learning for $(i+1)^{th}$ document. Thus, after 1153 iterations, the final output.mln file is obtained after training over the 1153 documents. This file was then used for inferring the results as explained in Section 7.4

Fig. 7.2 shows the *train.db* used for learning.

```
Etype(5,0,Loc)
EFtype(5,0,Loc)
Etype(5,1,Org)
EFtype(5,1,Org)
Etype(5,2,Org)
EFtype(5,2,Org)
Etype(5,3,Org)
EFtype(5,3,Org)
Etype(5,4,Org)
EFtype(5,4,Org)
Rtype(5,4,0,OrgBased_In)
RFtype(5,4,0,OrgBased_In)
```

Fig. 7.2 Sample train.db file

Learning in alchemy can be understood as follows:

- The formula is first converted to conjunctive normal form (CNF). While learning the weights, weight is learned for each of the clauses in the formula.
- For formulas preceded by weight in the input .mln file, the weight gets divided equally among all of the formula's clauses.
- The weight of a clause is used as the mean of a Gaussian prior for the learned weight. For hard constraints i.e. formulas preceded by periods, each of the clauses in its CNF is assigned a prior weight. This weight is twice the maximum of the soft clause weights. These hard constraints are specified in the output file during inferencing.
- If a formula is neither a hard constraint nor a prior weight is specified,a default prior weight is used for each of the formula's clauses.

The results obtained after learning, that is the FOL rules in the CNF form with weights assigned to them, were stored in the output.mln file as specified in the command for learning. This file containing the weighted FOL rules was then used for performing inference. Inference is discussed in the next section.

The following Fig. 7.3 shows the *output.mln* file which contains the weighted FOL rules (in CNF form) along with the hard constraints mentioned at the end of the file.

```
10.5383  EFtype(a1,a2,Peop) v !Etype(a1,a2,Peop)

// 8.84984  EFtype(a1,a2,Loc) v !Etype(a1,a2,Loc)
8.84984  EFtype(a1,a2,Loc) v !Etype(a1,a2,Loc)

// 4.12466  EFtype(a1,a2,Org) v !Etype(a1,a2,Org)
4.12466  EFtype(a1,a2,Org) v !Etype(a1,a2,Org)

// 15.933  EFtype(a1,a2,Other) v !Etype(a1,a2,Other)
15.933  EFtype(a1,a2,Other) v !Etype(a1,a2,Other)

// 12.7747  RFtype(a1,a2,a3,Work_For) v !Rtype(a1,a2,a3,Work_For)
12.7747  RFtype(a1,a2,a3,Work_For) v !Rtype(a1,a2,a3,Work_For)

// 12.2892  RFtype(a1,a2,a3,Live_In) v !Rtype(a1,a2,a3,Live_In)
12.2892  RFtype(a1,a2,a3,Live_In) v !Rtype(a1,a2,a3,Live_In)

// 11.8412  RFtype(a1,a2,a3,Located_In) v !Rtype(a1,a2,a3,Located_In)
11.8412  RFtype(a1,a2,a3,Located_In) v !Rtype(a1,a2,a3,Located_In)

// 11.8699  RFtype(a1,a2,a3,OrgBased_In) v !Rtype(a1,a2,a3,OrgBased_In)
11.8699  RFtype(a1,a2,a3,OrgBased_In) v !Rtype(a1,a2,a3,OrgBased_In)

// 10.3628  RFtype(a1,a2,a3,Kill) v !Rtype(a1,a2,a3,Kill)
10.3628  RFtype(a1,a2,a3,Kill) v !Rtype(a1,a2,a3,Kill)

// 9.33856  RFtype(a1,a2,a3,None) v !Rtype(a1,a2,a3,None)
9.33856  RFtype(a1,a2,a3,None) v !Rtype(a1,a2,a3,None)

EFtype(s,t1,Peop) ^ EFtype(s,t2,Peop) => RFtype(s,t1,t2,Kill).

EFtype(s,t1,Loc) ^ EFtype(s,t2,Loc) => RFtype(s,t1,t2,Located_In).

EFtype(s,t1,Peop) ^ EFtype(s,t2,Org) => RFtype(s,t1,t2,Work_For).

EFtype(s,t1,Org) ^ EFtype(s,t2,Loc) => RFtype(s,t1,t2,OrgBased_In).

EFtype(s,t1,Peop) ^ EFtype(s,t2,Loc) => RFtype(s,t1,t2,Live_In).
```

Fig. 7.3 Sample output.mln file

7.4 Inference

Inference in the project can be understood as obtaining the most probable type associated with an entity and the most probable relation between two entities using the weak evidence from ML and the weighted FOL rules from learning. Inference in Alchemy gives all possible states (all possible entity types for a token and all possible relations between two tokens) each associated with a particular probability value. The predicted entity for a token and the relation between two tokens is the one with the highest probability value among the inferred values.

These probability values are inferred using the weighted FOL rules (including the hard constraints) along with test data, which is the data on which the inference is to be performed (Etype and Rtype weak evidences obtained from the classifiers) and the queries.

Inference in MLN can be performed with various algorithms, for example:

Belief propagation: Known as sum-product message passing, this algorithm is essentially used for performing inference on graphical models, such as Bayesian networks and Markov Random Fields. The marginal distribution is calculated for each unobserved node (or variable), conditional on any observed nodes (or variables). Belief propagation is commonly used in AI and Information theory [42].

Lifted Belief Propagation:[43] It is a lifted version of a scalable probabilistic inference algorithm, belief propagation. In this algorithm, first, a lifted network is constructed, where each node represents a set of ground atoms that all pass the same messages during belief propagation. Belief propagation is then run on this network.

To perform inference, the ***infer*** executable was executed using the following command [44]:

infer -i output.mln -e test.db -r infer_results.results -q queries

The infer executable is present in the bin folder of alchemy.

Flags used:

-i: .mln file as input (*output.mln file* obtained from learning containing the weighted FOL rules. Hard constraints, that is, the FOL rules with periods are to be appended at the end of this file)

-*e*: The evidence test.db file. Contains weak evidences Etype and Rtype obtained from the classifiers

-*r*: final results file which would have the probability values

-*q*: specifies the query formulae, they can either be separated by a semi-colon or can be given in a separate file, say query file. In the implementation, queries would be the final types of entities EFtypes and the final relation between those entities RFtype.

If the .db file contains at least one grounding, that is it has at least one evidence from the classifier output, then predicate defined can be considered as evidence predicate.

Two basic types of inference are supported by Alchemy: probabilistic and MAP/MPE. The current implementation contains four probabilistic inference algorithms: (Lifted) Belief Propagation (flag -bp), MC-SAT (flag -ms), Gibbs sampling (flag -p) and simulated tempering (flag -simtp). When inference is performed, the probabilities of the query atoms being true is written in the results file. The flag -maxSteps is used to specify the maximum number of steps in the algorithm. In the project, lifted belief propagation has been used for inference.

To perform inference, 288 sentences from the CoNLL test dataset were used. The inference was also performed on sentence level, where weak evidence Etype and Rtype from a single sentence were used to form test.db and was inferred using the weighted FOL rules. Likewise, all 288 sentences were inferred separately, and then the results were combined into a single .results file for further analysis.

7.5 Getting the Final Types

The inferred results were obtained as a .results file gave all possible entity types for a token and possible relations between two tokens along with a probability value associated with each state.

To get the final types for entity and relation, the one with the highest probability value amongst all the possible tags was chosen and that was considered as the final type for the entity and relation.

Fig. 7.4 shows the Predicates along with their Probabilities obtained after inference.

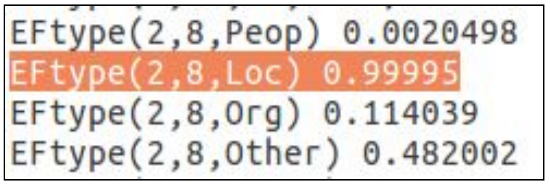

Fig. 7.4a Predicted Final types for entity-1

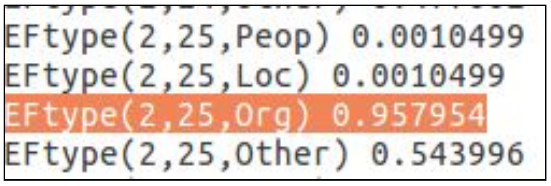

Fig. 7.4b Predicted Final types for entity-2

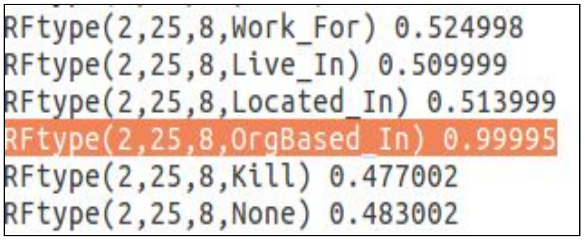

Fig. 7.4c Predicted Final types for relation

Chapter 8. RESULTS

Criteria: As stated in Section 3.3 the underlying goal of the platform was to improve the results of the classifiers for the essential relations (i.e. relations apart from "None"). Keeping this criterion in mind, the final results obtained for comparison of the two methods (i.e. machine learning and MLN) did not include those annotated documents whose Gold Truth relation was the "None" relation.

8.1 Comparison of Classifiers and Alchemy

Table 8.1 shows the classification report obtained for machine learning based Relation Classifier after testing for 288 sentences of CoNLL dataset.

Table 8.1 Classification Report for Machine Learning based Relation Classifier

	Precision	Recall	F1 Score	Support
Kill	0.74	0.74	0.74	47
Live_In	0.73	0.49	0.59	100
Located_In	0.84	0.77	0.80	94
OrgBased_In	0.86	0.68	0.76	105
Work_For	0.86	0.49	0.62	76
Accuracy			0.63	422
macro avg	0.67	0.53	0.58	422
Weighted avg	0.81	0.63	0.70	422

Table 8.2 shows classification report obtained for Alchemy based Relation Classifier after testing for 288 sentences of CoNLL Dataset.

Table 8.2 Classification Report for Alchemy based Relation Classifier

	Precision	Recall	F1 Score	Support
Kill	0.76	0.79	0.77	47
Live_In	0.75	0.59	0.66	100
Located_In	0.83	0.81	0.82	94
OrgBased_In	0.86	0.68	0.76	105
Work_For	0.86	0.49	0.62	76
Accuracy			0.66	422
macro avg	0.67	0.56	0.60	422
Weighted avg	0.81	0.66	0.72	422

Inferences:

1. Based on the results it was inferred that the relations which were misclassified by the Relation Classifier were corrected using Alchemy.
2. A report was generated to analyze some of the sentences that were corrected by Alchemy as shown in Table 8.3. It was observed that the Relation Classifier had misclassified many of these relations as “None”, although these sentences were annotated correctly by the Entity Classifier. These sentences were correctly classified by Alchemy which improved the overall accuracy of the Relation Classifier.

Table 8.3 Sentences Corrected by Alchemy

Sentence	Classifier Output	Alchemy Output	Gold Truth
By year 's end , 6 million acres had burned in the West and <B-Loc> Alaska </B-Loc>, making 1988 the worst fire season in 30 years , and , in terms of firefighting resources committed, the most expensive in <B-Loc> U.S. </B-Loc> history, Sacher said.	None	Located_In	Located_In
On April 14 , while attending a play at the Ford Theatre in Washington, <B-Peop> Lincoln </B-Peop> was shot in the head by actor John Wilkes <I-Peop> Booth </I-Peop> , who cried out ` ` Sic Semper Tyranus ' ' (` ` Thus Ever to Tyrants , ' ' the motto of Virginia)	None	Kill	Kill
There are even hopeful signs in Chicago 's schools, branded 14 months ago as the nation 's worst by <B-Loc> U.S. </B-Loc> Secretary William J. <I-Peop> Bennett </I-Peop>	None	Live_In	Live_In

Chapter 9. CONCLUSION

Based on the results mentioned above it is concluded that the main objective of using Markov logic networks built on top of the machine learning models in order to enhance their accuracy is fulfilled. In the next sections the limitations, as well as the future scope and applications of using this technique are mentioned.

9.1 Contributions

The main motivation of this project was to unify the approach of machine learning and MLN to improve the results obtained from ML alone for entity and relation extraction, from a given corpus of documents. By combining logic and probability, Markov logic provided an excellent framework for this approach. The classical approach for information extraction was discussed in chapters 4 and 5 on ML-based entity and relation classifiers. Further, the unifying approach had been discussed in detail in chapter 7 on the implementation of MLN using alchemy, where Markov logic was successfully applied to the results obtained from ML-based classifiers (weak evidences) for information extraction.

As stated in chapter 8, the classification report for ML-based relation classifier and MLN derived relation results, showed that alchemy (MLN model) was able to correct the relations misclassified by ML-based classifiers. Hence, using this approach MLN was able to improve the results obtained by the relation classifier. In this dissertation, it is shown that the limitations and assumptions associated with the conventional ML-based approach for information extraction, can be overcome by combining them with the probabilistic graphical model through MLN. The MLN model codifies the domain knowledge to form rules which can be used in inferencing to deal with real-world scenarios where data is uncertain and has complex relations in it.

9.2 Limitations of This Work

1. **Rigid Criteria for Testing:** The initial criteria for testing this hypothesis which included only essential relations was rigid. Based on the first-order rules used by Alchemy it was observed that several sentences whose “Gold Truth” relation was

"None" were misclassified by Alchemy. Alchemy generated several false positives for the essential relations ("Not None" relations), which reduced the precision for the essential relations.

2. **Restrictive dataset**: The CoNLL dataset had a small set of entities (4) and relations (6). Hence the models were not trained to obtain more generalized results. This led to a majority of entities being labelled as "Other" and a majority of relation being labelled as "None".
3. **Limitations of Alchemy:**
   a) ***Training of Alchemy:*** It was observed that after training of Alchemy the weights assigned to the FOL rules were proportional to the corresponding F1 scores of Entity and Relation Classifier. However, no significant changes in those weights were observed as the training set of documents was increased from 100 upto 1153 documents. The Alchemy based Markov network trained on 1153 documents gave similar results to the Alchemy Markov network that was trained for 100 documents.
   b) ***Propagation of Mis-Classifications:*** Based on the FOL rules Alchemy was dependent on the results of the Entity Classifier. Entity Classifier results i.e. Etypes were propagated as EFtypes which included few mis-classifications of the Entities. These were used to obtain RFtypes for Relation Classifier generating few erroneous classifications for the relations. The Entity Classifier formed the base upon which Alchemy corrected the Relation Classifier. No FOL rules were used to correct the Entity Classifier using the RFtypes of Relation Classifier.

9.3 Future Directions / Enhancements

1. **Modifications in Testing of Relation Classifier:**
   a) **Machine Learning Model:** To obtain results for purely the relation classifier, independent of the Entity Classifier, the sentences can be annotated using Gold Truths of entities of the test dataset. These sentences

can then be provided as input to the Relation Classifier for testing and the results can be obtained purely of the relation classifier.

b) **Improvements using Alchemy:** Similarly, during Testing of Alchemy the Gold Truths of the entities in the test dataset ("EFtypes") can be given as inputs rather than Predicted Labels of the Entity Classifier (Etypes), to obtain the results.

It is observed that this technique removes any propagation of mis-classification as mentioned in Section 9.2, and hence results in 100% accuracy of Alchemy based relation classifier i.e all sentences having essential relations are correctly classified by Alchemy improving upon the results of the machine learning model.

Thus, this technique can be used as a brute force approach for correcting the Relation Classifier having known the Gold Truth of the two entities that are annotated in the sentence of the test dataset, to determine the correct relation.

However, this technique is not much applicable for real-life examples where the Gold Truth of several entities are unknown.

2. **Testing on Other datasets:**

The CoNLL dataset had just 4 entities and 6 relations. To improve the current results further by using generalized FOL rules for the MLN, other such datasets having more such entity and relation labels can be used. Some of these datasets used for NLP based information extraction include:

- ACE dataset.
- DREC dataset.

BIBLIOGRAPHY

[1] https://en.wikipedia.org/wiki/Statistical_relational_learning - *Statistical relational learning*

[2] https://www.cs.purdue.edu/homes/neville/courses/CS590N.html - *CS590N: Statistical Relational Learning*

[3] https://data-science-blog.com/blog/2016/08/17/statistical-relational-learning/ - *Statistical Relational Learning – Data Science Blog*

[4] https://www.datanami.com/2018/07/03/can-markov-logic-take-machine-learning-to-the-next-level/ - *Can Markov Logic Take Machine Learning to the Next Level?*

[5] H. Poon and P. Domingos. Joint inference in information extraction. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence*, pages 913–918, Vancouver, Canada, 2007. AAAI Press.

[6] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.

[7] F. Wu and D. Weld. Automatically refining the Wikipedia infobox ontology. In *17th International World Wide Web Conference (WWW-08)*, Beijing, China, 2008.

[8] J. Wang and P. Domingos. Hybrid Markov logic networks. In *Proceedings of the TwentyThird National Conference on Artificial Intelligence*, Chicago, IL, 2008. AAAI Press.

[9] W. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Census Bureau, 1999.

[10] P. Singla and P. Domingos. Entity resolution with Markov logic. In *Proceedings of the Sixth IEEE International Conference on Data Mining*, pages 572–582, Hong Kong, 2006. IEEE Computer Society Press.

[11] P. Singla, H. Kautz, J. Luo, and A. Gallagher. Discovery of social relationships in consumer photo collections using Markov logic. In *Proceedings of the CVPR-2008 Workshop on Semantic Learning and Applications in Multimedia*, page To appear, Anchorage, Alaska, 2008.

[12] http://www.cse.iitd.ac.in/~parags/papers/thesis.pdf - Parag. *Markov Logic: Theory, Algorithms and Applications*

[13] Giannis Bekoulis, Johannes Deleu, Thomas Demeester, Chris Develder. *Joint entity recognition and relation extraction as a multi-head selection problem*, Ghent University-imec, IDLab, Department of Information Technology, Technologiepark, Zwijnaarde 15, 9052 Ghent, Belgium

[14] Guoliang Ji, Kang Liu, Shizhu He, Jun Zhao. *Distant Supervision for Relation Extraction with Sentence-Level Attention and Entity Descriptions,* National Laboratory of Pattern Recognition (NLRP), Institute of Automation, Chinese Academy of Sciences, Beijing, 100190, China

[15] Makato Miwa (Toyota Technological Institute Nagoya, 468-8511, Japan), Mohit Bansal (Toyota Technological Institute at Chicago, Chicago, IL, 60637, U.S.A) *End-to-End Relation Extraction using LSTMs on Sequences and Tree Structures*

[16] Mathew Richardson and Pedro Domingos. *Markov Logic Networks* Department of Computer Science and Engineering, University of Washington, Seattle, WA, 98195-250, U.S.A

[17] Pedro Domingos, Stanley Kok, Hoifung Poon, Matthew Richardson, Parag Singla *Unifying Logical and Statistical AI.* Department of Computer Science and Engineering University of Washington Seattle, WA 98195-2350, U.S.A

[18] Sachin Pawar[1,2] Pushpak Bhattacharya[2,3], Girish A Palshikar[1]
[1]TCS, Research, Tata Consultancy Services, Pune-411013
[2] Dept. of CSE, Indian Institute of Technology Bombay, Mumbai-400076, India
[3]Indian Institute of Technology Patna, Patna-801103, India.
End-to-End Relation Extraction using Markov Logic Networks

[19] https://www.depends-on-the-definition.com/lstm-with-char-embeddings-for-ner/ - *Enhancing LSTMs with character embeddings for Named entity recognition*

[20] https://www.depends-on-the-definition.com/attention-lstm-relation-classification/ - *LSTM with attention for relation classification*

[21] https://www.microsoft.com/en-us/research/wp-content/uploads/2016/10/poon_thesis.pdf - Hoifung Poon. *Markov Logic for Machine Reading*

[22] Pedro Domingos and Daniel Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool, San Rafael, CA, 2009.

[23] Matt Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006

[24] Lise Getoor and Ben Taskar, editors. *Introduction to Statistical Relational Learning*. MIT Press, Cambridge, MA, 2007.

[25] M. R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, San Mateo, CA, 1987.

[26] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, CA, 1988.

[27] Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82:273–302, 1996.

[28] W. R. Gilks, S. Richardson D. J. Spiegelhalter, and eds., editors. *Markov Chain Monte Carlo in Practice*. Chapman and Hall, 1996.

[29] Hoifung Poon and Pedro Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. *In Proceedings of the Twenty First National Conference on Artificial Intelligence*, pages 458–463, Boston, MA, 2006. AAAI Press.

[30] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, 1996.

[31] https://ix.cs.uoregon.edu/~lowd/lmmaa.pdf - *Markov Logic: A Language and Algorithms for Link Mining*

[32] Stanley Kok and Pedro Domingos. Learning the structure of Markov logic networks. In *Proceedings of the Twenty Second International Conference on Machine Learning*, pages 441–448, Bonn, Germany, 2005. ACM Press.

[33] Stanley Kok and Pedro Domingos. Learning markov logic network structure via hypergraph lifting. In *Proceedings of the Twenty Sixth International Conference on Machine Learning*, pages 505–512, Montreal, Canada, 2009. Omnipress.

[34] Stanley Kok and Pedro Domingos. Learning markov logic networks using structural motifs. In *Proceedings of the Twenty Seventh International Conference on Machine Learning*, Haifa, Israel, 2010. Omnipress.

[35] Lilyana Mihalkova and Raymond Mooney. Bottom-up learning of markov logic network structure. In *Proceedings of the Twenty Fourth International Conference on Machine Learning*, pages 625–632, Corvallis, Oregon, 2007.

[36] J. Besag. Statistical analysis of non-lattice data. The Statistician, 24:179–195, 1975.

[37] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, Williamstown, MA, 2001. Morgan Kaufmann.

[38] M. Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8, Philadelphia, PA, 2002. ACL.

[39] https://alchemy.cs.washington.edu/ - *Alchemy - Open Source AI*

[40] https://alchemy.cs.washington.edu/user-manual/3_2Weight_Learning.html - *3.2 Weight Learning*

[41] https://en.wikipedia.org/wiki/Generative_model - *Generative Model*

[42] https://en.wikipedia.org/wiki/Belief_propagation - *Belief propagation*

[43] P. Singla and P. Domingos. Lifted first-order belief propagation. In *Proceedings of the Twenty-Third National Conference on Artificial Intelligence*, Chicago, IL, 2008. AAAI Press.

[44] https://alchemy.cs.washington.edu/user-manual/3_4Inference.html - *3.4 Inference*