Named Entity Recognition for Telugu using LSTM-CRF

Aniketh Janardhan Reddy, Monica Adusumilli,
Sai Kiranmai Gorla, Lalita Bhanu Murthy Neti and Aruna Malapati
Birla Institute of Technology and Science, Pilani, Hyderabad, India
{f20140096,f20140005,p2013531,bhanu,arunam}@hyderabad.bits-pilani.ac.in

Abstract

Named Entity Recognition (NER) for Telugu is a challenging task due to the characteristic features of the language. Earlier studies have made use of handcrafted features and gazetteers for identifying named entities (NEs). In this paper, we present a Long Short-Term Memory (LSTM) - Conditional Random Fields (CRF) based approach that does not use any handcrafted features or gazetteers. The results are compared to those of traditional classifiers like support vector machines (SVMs) and CRFs. The LSTM-CRF classifier performs significantly better than both of them, achieving an F-measure of 85.13%.

Keywords: Named Entity Recognition, LSTM, CRF, Word Embeddings

1. Introduction

NER is an interesting task in Natural Language Processing (NLP) that identifies NEs such as the name of a person, location or organization in a sentence. NER has numerous applications in NLP and is used while performing text mining, machine translation, question answering, indexing for information retrieval, automatic summarization, etc.

Telugu is one of the most spoken Indian languages and is highly inflectional and agglutinating in nature. It has one of the richest and most complex set of linguistic rules resulting in complex word forms. The task of building an NER model for Telugu language has some linguistic challenges like the unavailability of annotated corpora, no gazetteer lists, no capitalization, spelling variations, free word order, etc. NEs in Telugu cannot be identified by capitalization as is the case with English and most European languages. Inflectional suffixes can be added either to the root or to the stem of Telugu words and common nouns can also be NEs in certain cases. The words are also more diverse in nature. NER in Telugu is challenging because of these difficulties. In this paper, we develop a classifier which performs NER in Telugu using LSTM and CRFs. We then compare the performance of our approach with those of popular tools like YamCha and CRF++. The annotated data used for our work has been made available for public use.

In Section 2. we discuss related work and in Section 3. we discuss our LSTM-CRF classifier. We describe the other classifiers we tested in Section 4. and our experiments, results and their analysis is documented in Section 5. Finally, we conclude in Section 6.

2. Related Work

State-of-the-art NER models are based on LSTMs. They overcome the problems associated with approaches that use handcrafted rules or gazetteers. These approaches are labor intensive and inflexible. Lample et al. (Lample et al., 2016) proposed a language independent LSTM-CRF based classifier which used pretrained word embeddings, character-level embeddings and contextual word representations. A CRF is finally used to perform classification. Our architecture is very similar to the one proposed by Lample et al. They also proposed another LSTM-based architecture inspired by shift-reduce parsers in the same paper. Chiu et al.(Chiu and Nichols, 2015) proposed a bidirectional LSTM (Bi-LSTM) and a Convolution Neural Network hybrid model that automatically detects word and character-level features. They reported an F-measure of 91.62% on the CoNLL-2003 dataset.

Considerable amount of work has been done on NER in other Indian languages such as Bengali and Hindi. Ekbal et al. (Ekbal and Bandyopadhyay, 2008) developed an NER model for Bengali and Hindi using SVMs. It uses various features which are computed based on both the word and the context in which it occurs. They reported F-measures of 84.15% and 77.17% for Bengali and Hindi respectively. In another paper(Ekbal and Bandyopadhyay, 2009), they proposed a CRF-based NER model for Bengali and Hindi and obtained F-measures of 83.89% for Bengali and 80.93% for Hindi. A CRF-based NER model which can handle nested NEs has been developed for Tamil(Vijayakrishna and Devi, 2008), another Dravidian language. Athavale et al.(Athavale et al., 2016) used a Bi-LSTM which took Word2Vec embeddings and the parts-of-speech (POS) tags of the tokens as input and output the NE tag. They reported an accuracy of 77.48% for NER in Hindi.

Srikanth and Murthy(Srikanth and Murthy, 2008) were some of the first authors to explore NER in Telugu. They built a two stage classifier which they tested using their own dataset. In the first stage, they used a CRF to identify nouns. Then, they developed a rule-based model to identify the NEs among the nouns. A CRF-based NER model was also built which made use of handcrafted features. Gazetteers were used to enhance the performance of their classifiers. Overall F-measures between 80% and 97% were reported in various experiments. It is interesting to note that the higher scores were obtained only upon using gazetteer lists. Their work also has several limitations. Firstly, they use handcrafted rules, features and gazetteers to perform NER which is both labor intensive and inflexible. Secondly, their classifier is only capable of identifying NEs which are one word long. Finally, neither their code nor their dataset has been made available publicly. Our work overcomes all of these problems. Shishtla et.al(Shishtla et al., 2008) built a CRF based NER model with language independent and dependent features and reported an F-measure of 44.89%.

3. The LSTM-CRF Classifier

We briefly introduce LSTMs, CRFs and other relevant concepts before proceeding to explain the architecture of the classifier we built.

3.1. Long Short-Term Memory (LSTM)

Recurrent Neural Networks (RNNs) are neural networks with self loops and are commonly used for building classifiers which operate on sequential data. In theory, vanilla RNNs are capable of learning long term dependencies between data points. But, it has been shown by Bengio et al. (Bengio et al., 1994) that they often fail to learn them due to vanishing gradients. LSTMs are a class of RNNs which were proposed by Hochreiter et al. (Hochreiter and Schmidhuber, 1997) to overcome this problem and are adept at learning long term dependencies. An LSTM takes sequential data of the form $(x_0, x_1, x_2...x_n)$ as input and gives a sequential output of the form $(y_0, y_1, y_2...y_n)$. LSTMs have an internal state which is updated whenever a new data point is input. This update is controlled by two gates. The forget gate determines how much of the previous state's information is to be retained. The input gate controls how much the state changes due to the new input. The LSTM finally gives an output which is determined by the output gate based on the internal state.

3.2. Character-Level Word Embeddings

The structure of a word is useful in determining if it is a named entity. For example, many Indian cities such as Hyderabad and Ahmadabad end with *-bad*. This structure can be captured through character-level word embeddings (Ling et al., 2015). Initially each character is assigned an n-dimensional vector. Each token is broken up into its individual characters which are then mapped to their corresponding vectors. Thus, a token is converted to a sequence of vectors which is then fed to a Bi-LSTM. A Bi-LSTM is composed of two LSTMs which process a sequence in opposite orders. The final states of both LSTMs are then concatenated to obtain the final character-level embedding of the token.

3.3. Generating Contextual Representations of Words using LSTMs

To make an output decision, an LSTM must store information about the previous data points. In the case of NER, the LSTM stores information about the tokens which occurred before the current token, thereby storing a contextual representation of the current token. In order to get both the left and right contexts we use a Bi-LSTM and then concatenate the internal states of the forward and backward LSTMs after processing the given token to get the token's final contextual representation.

3.4. Linear Chain Conditional Random Fields (CRFs)

When a conventional classifier is used to recognize NEs, each tagging decision occurs independent of the others. Entities can be spread across multiple tokens. For example, a person's name can be composed of two tokens consisting of his first and last names. There are also constraints on

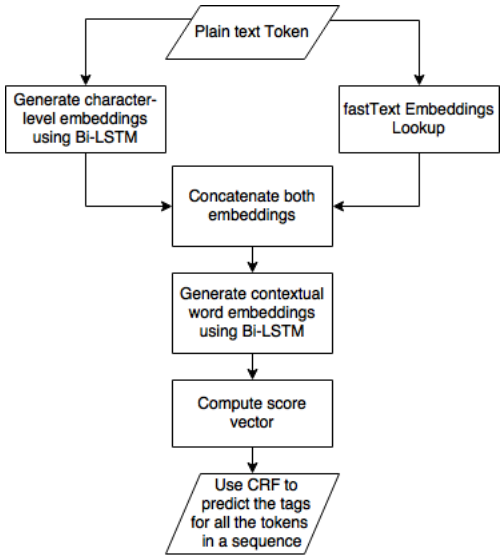

Figure 1: Main steps of our approach

the occurrences of certain tags. For example, a sequence of I-xx tags cannot occur without a corresponding B-xx tag before them. It is also highly unlikely for entities to be present one after the other without separators between them. These observations make it clear that a tagging decision must be made after taking into account the tagging decisions for all the other tokens in the sequence.

Using CRFs (Lafferty, 2001), tagging is done for the entire sequence simultaneously and each tagging decision is dependent on the others. We use a linear chain CRF which considers dependencies between adjacent words (linear dependencies). Each token t_i in a sequence of the form $(t_0, t_1, t_2...t_n)$ has a corresponding m-dimensional vector s_i where m is the number of possible tags. Another $m \times m$ transition matrix A is used to capture the linear dependencies between the tagging decisions. A_{ij} of the matrix is indicative of the probability of the i^{th} tag being followed by the j^{th} tag. s and e are two m-dimensional vectors whose values represent the confidence of a tagging sequence starting and ending with a given tag respectively. Each tagging sequence of the form $(y_0, y_1, y_2...y_n)$ assigned to the token sequence is scored as:

$$Score(y_0, y_1, y_2...y_n) = s[y_0] + \sum_{i=0}^{n} s_i[y_i] + \sum_{i=0}^{n-1} A[y_i, y_{i+1}] + e[y_n]$$

The tagging sequence which has the maximum $Score$ is output by the CRF based classifier. While training, we seek to minimize the negative log of the probability of the correct tagging sequence $\tilde{Y}$. Hence our loss function is:

$$Loss = -log(P(\tilde{Y}))$$

where $P(\tilde{Y}) = \frac{e^{Score(\tilde{Y})}}{\sum_{y_0,...,y_n} e^{Score(y_0,...,y_n)}}$

During backpropagation, the various weights and embeddings of the model are tuned to minimize the $Loss$.

3.5. Neural Network Architecture

Figure 1 gives an overview of our architecture. Our classifier makes use of global word embeddings generated from raw text so as to capture the context of words in unseen text. In specific, we use the 300-dimensional fastText pretrained Telugu word embeddings provided by Facebook Research[1] (Bojanowski et al., 2016). We also generate 200-dimensional character-level embeddings for each token as described in Section 3.2.. The global word embeddings and the character-level embeddings are then concatenated for each token to represent them as vectors. Using the training data now represented in the form of sequences of vectors, 600-dimensional contextual word embeddings are generated for each token as described in 3.3.. The contextual word embeddings are then used to compute the scores for a word using a $600 \times m$ weight matrix W and an m dimensional bias vector b where m is the number of classes.

The score for each word is an m dimensional vector given by:
$s = W \cdot h + b$, where h is the contextual embedding of that word

After getting the scores for an entire sequence, we use a linear chain CRF to make the tagging decisions for the whole sequence as described in Section 3.4..
Our code and dataset is made publicly available for reproduction of results and future research[2]. Parts of the code were adapted from the code written by Guillaume Genthial[3].

4. Other Classifiers

We compare our classifier with two other language independent classifiers which are publicly available, YamCha[4] and CRF++[5]. This section gives a brief description of these tools. We tried out many configurations for these classifiers. Here, we describe the ones which performed the best.

4.1. YamCha

YamCha is an SVM-based open source toolkit which can perform many NLP tasks such as POS tagging, NER, base noun phrase chunking and text chunking. The features to be used and the parsing direction can be customized. In our study, we use YamCha with the following combination of static and dynamic features:

1. Current token and its POS tag
2. The two tokens preceding and the two tokens following the current token along with their POS tags.
3. NE tags of the two previous tokens

For each token, the aforementioned features are generated and supplied to YamCha for training. While testing, the tags assigned by the classifier to the previous two tokens are used as the last feature.

4.2. CRF++

CRF++ is an open source implementation of CRFs for segmenting or labeling sequential data. It is also language independent and like YamCha, it can be used for performing several NLP tasks. CRF++ is written in C++ and it uses the popular Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm to perform training. We used the following features in our study:

1. Unigram Features:
   (a) The current token, the two tokens which precede it and the two tokens which follow it.
   (b) Combination of the current token and the token before it.
   (c) Combination of the current token and the token after it.
   (d) The POS tags of the current token, the two tokens which precede it and the two tokens which follow it.
   (e) Combination of the POS tags of the two tokens which precede the current token.
   (f) Combination of the POS tags of the current token and the token which precedes it.
   (g) Combination of the POS tags of the current token and the token which follows it.
   (h) Combination of the POS tags of the two tokens which follow the current token.
   (i) Combination of the POS tags of the current token and the two tokens which precede it.
   (j) Combination of the POS tags of the current token, the token which precedes it and the token which follows it.
   (k) Combination of the POS tags of the current token and the two tokens which follow it.
2. Bigram Feature - Combination of the the current token and the NE tag of the previous token.

These features are generated for each input token and CRF++ uses them for training an NE tagger.

5. Experiments

5.1. The Corpus

We accumulated the Telugu text by crawling Telugu newspaper websites. It was then manually tagged by us. NEs belonging to four classes, namely, person (PERSON), location (LOC), organization (ORG) and other miscellaneous NEs (MISC) were manually identified and tagged using the standard IOB scheme. The data consists of 47966 tokens out of which 6260 are NEs and Table 2 shows the number of NEs belonging to each of the four classes. The tagset is given in Table 1 below.

[1] https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md
[2] https://github.com/anikethjr/NER_Telugu
[3] https://github.com/guillaumegenthial/sequence_tagging
[4] http://chasen.org/~taku/software/yamcha
[5] https://taku910.github.io/crfpp/

Named Entity Tag	Example
PERSON	
	స్మృతి B-PERSON
	ఇరానీ I-PERSON
	(Smriti Irani)
LOC	
	సౌదీ B-LOC
	అరేబియా I-LOC
	(Saudi Arabia)
ORG	
	బిర్లా B-ORG
	ఇన్స్టిట్యూట్ I-ORG
	ఆఫ్ I-ORG
	టెక్నాలజీ I-ORG
	అండ్ I-ORG
	సైన్స్ I-ORG
	(Birla Institute of Technology and Science)
MISC	
	రూ. B-MISC
	35 I-MISC
	లక్షలు I-MISC
	(Rs. 35 lac)

Table 1: Named Entity Tagset

Class	Number of NEs
PERSON	1563
LOC	1915
ORG	778
MISC	2004
Total	6260

Table 2: Distribution of NEs

5.2. Training

Our classifier is trained using the Adam optimizer (Kingma and Ba, 2014) with the learning rate equal to 0.001 and the learning rate decay factor as 0.9. The batch size was equal to 20 and we also set a dropout rate of 0.5 while training. The training was carried out for 20 epochs.

5.3. Evaluation

10 sets of training and testing data were generated using the annotated corpus. 80% of the sentences present in the corpus comprise the training set and the remaining 20% comprise the test set. This split is done randomly and sentences are not repeated in the training and testing data. We then use these 10 splits to evaluate our classifier. The standard CoNLL evaluation script[6] is used to compute various evaluation metrics.

Approach	Precision	Recall	F-measure
YamCha	80.61	76.07	78.27
CRF++	80.75	74.92	77.72
LSTM-CRF	87.15	83.22	85.13

Table 3: Overall metrics for the various classifiers

Class	Yamcha	CRF++	LSTM-CRF
PERSON	76.61	76.08	80.07
LOC	81.21	80.52	88.11
ORG	54.32	55.86	68.02
MISC	82.10	84.56	92.68

Table 4: Comparison of the F-measures of the classifiers for each class of named entities

5.4. Results and Analysis

Tables 3 and 4 show the results we obtained. These metrics are computed after averaging the metrics over the 10 runs. The LSTM-CRF classifier clearly outperforms the other classifiers based on all the metrics. In fact, its performance is greater by approximately 7% based on overall F-measure. This is because of three main reasons. Firstly, the use of CRFs makes tagging more accurate because the classifier is able to discern the dependencies which exist between the individual tags. These dependencies can not be captured by an SVM, thereby lowering its performance. Secondly, character-level embeddings allow the classifier to learn the general form of an NE. Hence, our classifier is able to handle unknown NEs because it is able to identify them based on their structure. This cannot be accomplished when using tools like CRF++. Finally, Bi-LSTMs are great at learning long term dependencies and further augment the classifier's ability to learn dependencies between tokens which is not possible when using either SVMs or CRF++.

6. Conclusion

In this paper, we described an LSTM-CRF based approach for NER in Telugu. The approach makes use of pretrained fastText embeddings and character-level embeddings generated using a Bi-LSTM in order to learn contextual word embeddings using another Bi-LSTM. A linear chain CRF is finally used to perform the tagging based on the contextual word embeddings. The use of various word embeddings and a CRF allows the classifier to capture more contextual information and tagging dependencies respectively. The overall F-measure achieved using the LSTM-CRF classifier is approximately 7% greater than the F-measure obtained using SVM and CRF.

7. Bibliographical References

Athavale, V., Bharadwaj, S., Pamecha, M., Prabhu, A., and Shrivastava, M. (2016). Towards deep learning in hindi

[6]https://www.clips.uantwerpen.be/conll2003/ner/

ner: An approach to tackle the labelled data scarcity. *arXiv preprint arXiv:1610.09756*.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, March.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Chiu, J. P. and Nichols, E. (2015). Named entity recognition with bidirectional lstm-cnns. *arXiv preprint arXiv:1511.08308*.

Ekbal, A. and Bandyopadhyay, S. (2008). Bengali named entity recognition using support vector machine. In *IJCNLP*, pages 51–58.

Ekbal, A. and Bandyopadhyay, S. (2009). A conditional random field approach for named entity recognition in bengali and hindi. *Linguistic Issues in Language Technology*, 2(1):1–44.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Lafferty, J. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. pages 282–289. Morgan Kaufmann.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. *CoRR*, abs/1603.01360.

Ling, W., Dyer, C., Black, A. W., Trancoso, I., Fermandez, R., Amir, S., Marujo, L., and Luís, T. (2015). Finding function in form: Compositional character models for open vocabulary word representation. In *EMNLP*.

Shishtla, P. M., Gali, K., Pingali, P., and Varma, V. (2008). Experiments in telugu ner: A conditional random field approach. In *Proceedings of the IJCNLP-08 Workshop on Named Entity Recognition for South and South East Asian Languages*.

Srikanth, P. and Murthy, K. N. (2008). Named entity recognition for telugu. In *IJCNLP*, pages 41–50.

Vijayakrishna, R. and Devi, S. L. (2008). Domain focused named entity recognizer for tamil using conditional random fields. In *IJCNLP*, pages 59–66.