Comparison of Approaches in Named Entity Recognition

Andrew Deaver, Kathryn Hite

I. Introduction

The problem of named entity recognition (NER) has been of interest to researchers for many, many years. The approaches to solving this problem have ranged from classical, statistical models such as Maximum Entropy Markov Models (MEMM) to more modern, data-driven approaches, such as Recurrent Neural Networks (RNN). In this paper, we evaluate a number of approaches to solving this problem. For all of the tasks, we used the CoNLL2002 corpus. This gave us five tags with which to label all the words. They are as follows.

LOC, which represents locations
ORG, which represents an organization
PER, which represents a person
MISC, which represents an entity not covered by the previous 3 tags
O, which represents a word that is not an entity.

II. Background

NER involves recognizing entities such as locations, proper names, or organizations in natural language text documents. These documents can be structured or unstructured, and the characteristics of named entities can vary for different disciplines, leading to a wide variety of approaches.

A wide variety of methods have been explored to handle the problem of NER. Early systems involved utilizing rule-based algorithms that were built for each specific type of recognition problem. This method still remains common when there is insufficient data for a training set, but over the past decade the preferred approach has moved to statistical models and supervised classification algorithms such as neural networks. This allows much of the current research in machine learning to apply to the NER field as well. The most current advancement in this space is the exploration of semi-supervised models. These models reduce the amount of hand tagging that need to be done to create an enormous dataset for supervised learning by utilizing a small amount of tagged data in conjunction with unlabelled data. For this paper, we chose to implement and evaluate the two most commonly developed and promising approaches. The first is a Hidden Markov Model that explored the statistical modeling approach to the NER problem. We then develop a Long Short Term Memory Network combined with a Neural Network to explore methods of utilizing context vectors to improve accuracy.

III. Hidden Markov Model

A. Background

Hidden Markov Models (HMMs) are a type of Markov chain that estimates some sequence of latent states, Q, from a sequence of observed states. Such as with Markov chains, we're interested in maximizing the likelihood of a sequence of states given some transition probability between every two states, $P(q_t|q_{t-1})$. However, since we only have a sequence of observations, we must also find the probability that a state can generate the given observation. This probability is known as the emission probability and is given by $b_q(o_t)$.

Therefore, the probability of any latent state, q, is given by

$$q_t = \underset{q \in Q}{\operatorname{argmax}}(P(q|q_{t-1})b_q(o_t)) \tag{1}$$

where Q is the set of all possible latent states.

The optimal sequence of latent states is determined by the Viterbi algorithm. The Viterbi algorithm is a dynamic programming algorithm that constructs a table such that the rows represent all of the possible states and the columns represent all of tokens. Any given position in the table A_{ij} is the probability that a latent state generated a given token given the previous sequence.

B. Model

HMMs are generative models, since the emission probability represents the probability that a given latent state generated the observed data. These probabilities are typically estimated by training Gaussian mixture models. However, given the nature of the problem and the fact that we had access to labeled sequence data, we decided to modify the HMM to operate on a discriminative classifier as opposed to a generative one [1]. This difference is apparent upon examining $b_q(o_t)$. In the generative case, the emission probability is given by

$$b_q(o_t) = P(o_t|q) \tag{2}$$

In the discriminative case, the emission probability is given by

$$b_q(o_t) = P(q|o_t) \tag{3}$$

Typically, a generative model is used because the latent states are not known. This is not the case in our model, so

we will opt for a model that most closely aligns with the problem that we're trying to solve.

In order to calculate the emission probability, given a discriminative model, we used a mixture of support vector machines [2].

The transitions probabilities were calculated by counting bigrams of tags in the dataset and using the following formula

$$P(q_t|q_{t-1}) = \frac{count(q_t q_{t-1})}{count(q_t)} \tag{4}$$

C. Implementation

We used CoNLL-2003, a tagged training set of 50,000 words, for our NER training and test data.

We used the the pretrained model Python Word2Vec model from Google with the GoogleNews set of word vectors to translate our training and test data to word embedding input. Word embeddings allow us to use the same standard of input across each of the models.

We used SVMs to calculate the probability that a word represented a given tag. We iterated over each word of our training set to generate our training set. The SVM was trained with a bagging classifier in Sklearn to better parallelize the training.

The transition probabilities were computed by running through the training data and tracking the probability of each tag being followed by each other tag.

Given the calculated emission and transition probabilities from the tagged dataset, we decode the optimal sequence of states for the test dataset using the Viterbi algorithm.

The Viterbi algorithm uses the emission and transition probabilities to calculate the probability of each tag for a given word in the test set based on its emission probability for the word embedding as well as the most probable tag of the word preceding it.

D. Results

With the implementation described above, we achieve a 93.64% accuracy on our test dataset of 10,000 words. The following table shows the confusion matrix output from this test.

TAG	LOC	MISC	O	ORG	PER
LOC	0.36	0.00	0.33	0.26	0.05
MISC	0.00	0.27	0.64	0.00	0.09
O	0.00	0.00	1.00	0.00	0.00
ORG	0.00	0.03	0.19	0.73	0.06
PER	0.00	0.00	0.07	0.00	0.93

Our results show that the HMM was most proficient in identifying the tags O, PER, and ORG. LOC was often confused for ORG, which we predicted given the lack of context vectors in this model to help determine the sentence context that would show the difference between, for example, the location Boston, or the organization Boston from The Boston Globe.

IV. Recurrent Neural Network

A. Background

Recurrent neural networks (RNNs) are a very powerful class of machine learning models, usually used in tasks related to sequence modeling. They're especially useful in encoding information into a deeper format. A very powerful addition onto RNNs was later invented in 1997 by Hochreiter and Schmidhuber called Long Short-Term Memory (LSTM) [4]. LSTMs are a way of methodically updating and forgetting values since vanilla RNNs typically have problems with gradients becoming to overpowering or vanishing completely, especially in long sequences. LSTMs currently sit on the cutting edge of sequence modeling, which makes it a worthy candidate for comparison against the tried and true methods with HMMs mentioned earlier.

B. Model

Our model for NER with RNNs follows very closely to that of Lample, Guillaume, et al. [5]. This model consists of two major components, an encoder and a decoder.

1) Encoder: The encoder uses a bidirectional LSTM to encode the context of words. Simply put, we run an LSTM going both directions on the sentence to capture the context of the word, which might give clues as to whether or not a word is an important named-entity. This LSTM is run on the word embeddings, which are taken from the Google News Corpus. The task of this LSTM is to predict the next word in the sentence.
The outputs of each word from each of the encoders are concatenated into a new word embedding of dimensionality 600x1. These outputs become the inputs for a scoring function that score the likelihood of each tag.

2) Decoder: The decoder is a conditional random field model (CRF) [6] which is a probabilisitc model that works similar to an HMM. The idea of using this as a decoder is that it does not make local decisions, but instead, takes into account the context in which each tag occurs. The inputs to the CRF are the scores generated by the final step of encoding.

C. Implementation

We used the same CoNLL-2003 data set to train and evaluate our model. Both the encoder and decoder were written in Python using Google's TensorFlow library, which provides a number of utilities for quickly implementing this type of model. We used a lot of help from Guillaume Genthial's blog [7].

D. Results

We were unable to produce results for our LSTM-CRF model. However, Lample, Guillaume, et al. were able to achieve an F_1 score of 90.94 [5].

REFERENCES

[1] Kruger, Sven E., et al. "Mixture of support vector machines for hmm based speech recognition." *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on. Vol. 4. IEEE, 2006.*

[2] Collobert, Ronan, Samy Bengio, and Yoshua Bengio. "A parallel mixture of SVMs for very large scale problems." *Advances in Neural Information Processing Systems. 2002.*

[3] Nadeau, David. "A Survey of Named Entity Recognition and Classification." *National Research Council Canada*, nlp.cs.nyu.edu/sekine/papers/li07.pdf.

[4] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation 9.8 (1997): 1735-1780.*

[5] Lample, Guillaume, et al. "Neural architectures for named entity recognition." *arXiv preprint arXiv:1603.01360 (2016).*

[6] Lafferty, John, Andrew McCallum, and Fernando CN Pereira. "Conditional random fields: Probabilistic models for segmenting and labeling sequence data." *(2001).*

[7] Genthial, Guillaume. "Sequence Tagging with Tensorflow." *Guillaume Genthial Blog, 4 Apr. 2017*, guillaumegenthial.github.io/sequence-tagging-with-tensorflow.html.