Named Entity Recognition

by

Tanishq Chaudhary, Mayank Goel, Jerrin John Thomas

{tanishq.chaudhary, mayank.goel, jerrin.thomas}@research.iiit.ac.in

International Institute of Information Technology, Hyderabad

Abstract

The aim of this project is to extract domain entities, focusing on named entities. The major element of our project is to create a highly accurate NER for real-world data on Sports. This has high relevance in fields of sports marketing, news and user experiences, more specifically, consuming news/data about their favourite sports event.

To give an insight into different NER models, we use FIFA World Cup 2018 as an event to formulate our ideas. FIFA World Cup 2018 was chosen as it is a highly popular event and that means tweets are readily available.

We plan to use several methods to gauge the accuracy of our NEs (Named Entities). The base methods depend on POS Tagging and also a look at the chunking approach in the NLTK package. Owing to the various drawbacks in these methods when considering real world data like tweets, we built our own methods, one using ngram frequency and one using Deep Learning to gain much better results.

Introduction

The reason for working on multiple approaches was to be able to compare ideas and include cross-domain ideas. State-of-the-Art NERs exist, and for such algorithms, a string matching algorithm like Karp-Rabin's Algorithm based on hashing is used, primarily in search engines. This form of NER is used primarily by Google who also use proprietary algorithms for their own search engines.

For data like extracting NEs in Tweets or texts, we need a different approach. For k words with a combined length n where |k| = 0.01(n) (Assuming about one NE per sentence) we can gain a speed boost by tagging NE on each tweet itself and comparing the possible NE to the database. This is considering one NE per sentence. However, most tweets do not have NEs and comparing such a long database is a faulted approach. We intend to work on methodologies to fix this problem using various approaches and its potential value in other fields, focusing on the research aspect instead of trying to desperately increase accuracy of a model, most of which are heavily data-dependent. Even so, we gained much better accuracies than naive methods, or ones in NLTK, at much better speed (or at highly increased accuracy in the case of Deep Learning)

Github Repository

Specifications

Annotation Code and Dataset

Accuracy Checking

We used a dataset online which had 530k tweets collected on FIFA World Cup 2018. This was raw data that we cleaned, and then annotated. We POS tagged each sentence using NLTK, and tagged the NEs in them. The tags assigned were - Beginning and Middle Player NE, and Beginning and Middle Team NE (for purpose of multi word NEs), using the IOB format.

The data was cleaned with the help of the CSV module in Python, and was formatted using Dataframes in Pandas. The code of data cleaning is well-commented and is in the form of a Notebook on Github. We used binary search to speed up our annotation by a considerable degree.

We are also using the GMB extract on Kaggle which was already cleaned. We have converted the NE tags used to parallel our Twitter data.

We have an upper bound on the actual accuracy mainly due to the POS Tagging. POS Tagging assigns each word with a likely part of speech. The errors of the POS tagger lead to false positives and negatives during extraction. Errors in the dataset also lead to errors in domain entity extraction. 7.21% of NEs have been not tagged as NNPs, for example. This clearly leads to a loss of accuracy.

Method 1 - Pure POS Tag Approach

Notebook

The first approach is the naive way of approaching the NER task. We simply look at the POS(Part-Of-Speech) Tags of a word. If the Tag is NNP, a proper noun, we tag it as a Named entity. This approach is highly trivial and depends on the efficiency of the POS Tagger.

Interestingly enough, in the domain of tweets, sentences need not be complete and the tweets use various neologisms to denote ideas, which make POS Tagging an inadequate approach. We have used the formula below:

$$\frac{Intersection\ of\ all\ NEs\ and\ NNPs\ (which\ occurs\ with\ 92.79\%\ accuracy)}{All\ NNPs}$$

This gives us 38.81% accuracy. This is clearly inefficient and pure POS Tags can not be used, as the recall in itself is also low, considering the fact that it’s very structure dependent. We will see more methods which are more word dependent.

Method 2 - NLTK Chunking Approach

Notebook

This is the method in which we directly use NLTK, which is often used as a base target for most applications. We remark 50.07% accuracy, which is an obvious step up from our previous approach. However it is obvious this approach still has lots of drawbacks and cannot be used as any form of dependable indicator for NER tagging. For real-world datasets such as tweets, the recall in itself is very low. This is because NLTK uses capitalization as an indicator which works well in formal datasets, but some sentences may have uneven capitalizations.

Method 3 - n-gram Frequency

Notebook

Words are of varying importance in a corpus. The more important information they hold, the more frequent they tend to appear in the corpus. But some words like “the”, “of” are so common that word frequency will tend to emphasize on such words than on the keywords. Since the focus is on named entities which are nouns and using the inverse frequency will lead to less accuracy for polygrams, the term frequency is sufficient. Named entities can be of lengths of one, two, three or more words. Hence the approach of n-gram frequency has been arrived upon.

The sentences are concatenated and the resulting data is split into n-grams. The most frequent ones are added into a set. The limit for frequency is set by a function which has been taken to be linear. The sentences are taken one-by-one and POS tagged using the nltk POS tagger. Iterating through each word of the sentence, noun phrases are extracted by searching for a continuous string of nouns. This noun phrase is broken into n-grams and they are checked if they are present in the set of frequent n-grams.

Assessments and Measures

This method is quite promising as it is highly suited to Tweets or informal data. Since it works on frequent ngrams, it can also be used as a very helpful domain indicator. This method suffers from the inaccuracy of POS Tagging, however it still works as an efficient method due to its functioning with low amounts of data as well. We have an accuracy of 89.40% using this method, which can be very useful for NER along with domain entities. However, it is apparent that a statistical learning method will be best suited.

Method 4 - Forms of Machine Learning

Notebook

Memory Tagging

What would happen if we simply memorized all the tags?

We have used the scikit learn library to implement the Memory Tagger.

The Memory Tagger follows the naive and the bruteforce style of approaching the problem. It remembers each and every single word and its associated tag. In simple words, it forms a dictionary. When asked to predict, it simply predicts the tag of the word as the most frequent tag in the dictionary. If the word is not in its vocabulary, the Tagger by default predicts 'O'.

The out-sample weighted F1 score is: 90%

```
              precision    recall  f1-score   support

      B-Play       0.16      0.09      0.12     16990
      B-Team       0.53      0.20      0.29     37644
      I-Play       0.34      0.01      0.03     17251
      I-Team       0.41      0.06      0.10      7414
           O       0.94      0.99      0.96    969276

    accuracy                           0.92   1048575
   macro avg       0.47      0.27      0.30   1048575
weighted avg       0.89      0.92      0.90   1048575
```

Deep Learning

We have used the deep learning library, keras to create, compile and train the neural network.

Data Exploration

The data was successfully cleaned and put in the required format. We now start with data exploration. The notebook graphs give a general idea of the length of the data being used, the number of sentences, the number of words for both the corpora and the number of tags: both POS tags and the NE tags. The maximum length of the tweets was found to be of 62 words.

Neural Architecture

LSTM: Recurrent neural networks (RNNs) are a powerful family of Neural Networks that capture Sequential Data. They take in an input of vectors and return another sequence which represents some information. Though, in theory, RNNs are capable of capturing long-distance dependencies, in practice, they fail due to the vanishing/ exploding gradient problems.

Long Short-term Memory Networks (LSTMs) have been designed to combat this issue by incorporating a memory-cell and have been shown to capture long-range dependencies. They do so by using several gates that control the proportion of the input to give to the memory cell, and the proportion from the previous state to forget.

This still poses a problem, because only the dependencies going from left to right are captured. Thus, we use two LSTMs, one going forward and the other going backwards; which is also known as a Bi-Directional LSTM. It is able to look at the past context of a word and the future context of the word. The outputs of each of the hidden LSTMs are concatenated to form the final sequence.

CRF: One of the popular tagging approaches is to make each and every output independent of its surroundings, like we have done before. Despite this model's success in simple problems, its independent classification decisions are limiting when there are strong dependencies across output labels.

For example, in POS Tagging, a noun is most likely to follow an adjective, but not an adverb, or that a determiner will always precede a NP. We see various forms of such syntactic contexts, which is also used for other frequentist approaches as in POS tagging like in the Brill's Tagger. Such type of grammar is also present in the IOB tags we have used; for example, I-Player will not be preceded by B-Team, but by I-Player or B-Player.

For CRF training, we use the maximum conditional likelihood estimation. Training and decoding can be done efficiently by adopting the Viterbi algorithm.

Bi-LSTM CRF: Now, we construct our final model.

```
Model: "sequential"

_________________________________________________________________

Layer (type)                 Output Shape              Param #

=================================================================

embedding_1 (Embedding)      (None, 62, 64)            3319552

_________________________________________________________________

dropout_1 (Dropout)           (None, 62, 64)            0

_________________________________________________________________

bidirectional_1              (None, 62, 256)           197632

(Bidirectional)

_________________________________________________________________

dropout_2 (Dropout)           (None, 62, 256)           0

_________________________________________________________________

Time_distributed_1           (None, 62, 5)             1285

_________________________________________________________________

crf_1 (CRF)                 (None, 62, 5)             65

=================================================================

Total params: 3,518,534

Trainable params: 3,518,534

Non-trainable params: 0

_________________________________________________________________
```

Training the Network

Word Embeddings

It is obvious that the network does not understand words. It can only input numbers. So, we need to convert the given words to vectors.

First, we convert all the words to arbitrary numbers, from 1 to 51,686, the number of words. At this point, the words can have any integer value associated with them. In the first layer of our deep learning model, we have an embedding layer. This maps all the 51,686 words to a 64 dimensional vector for each and every single word of the sentence. The mathematical reasoning followed is beyond the scope of this project.

Dropout

To mitigate overfitting, we apply the dropout method to regularize our model. We apply it after embeddings and the Bi-LSTM Layers. The dropout rate is set to 50% after both the layers. We obtain significant improvements on our model performance after using dropout.

Optimization Algorithm

We have used the Adam optimizer with parameters:

lr=0.0005, beta_1=0.9, beta_2=0.999

'lr' is the learning rate, and the beta values set the decay rate. Their reason for use lies in Differential Calculus, the specific details of which are irrelevant to the project

Implementation Level Details

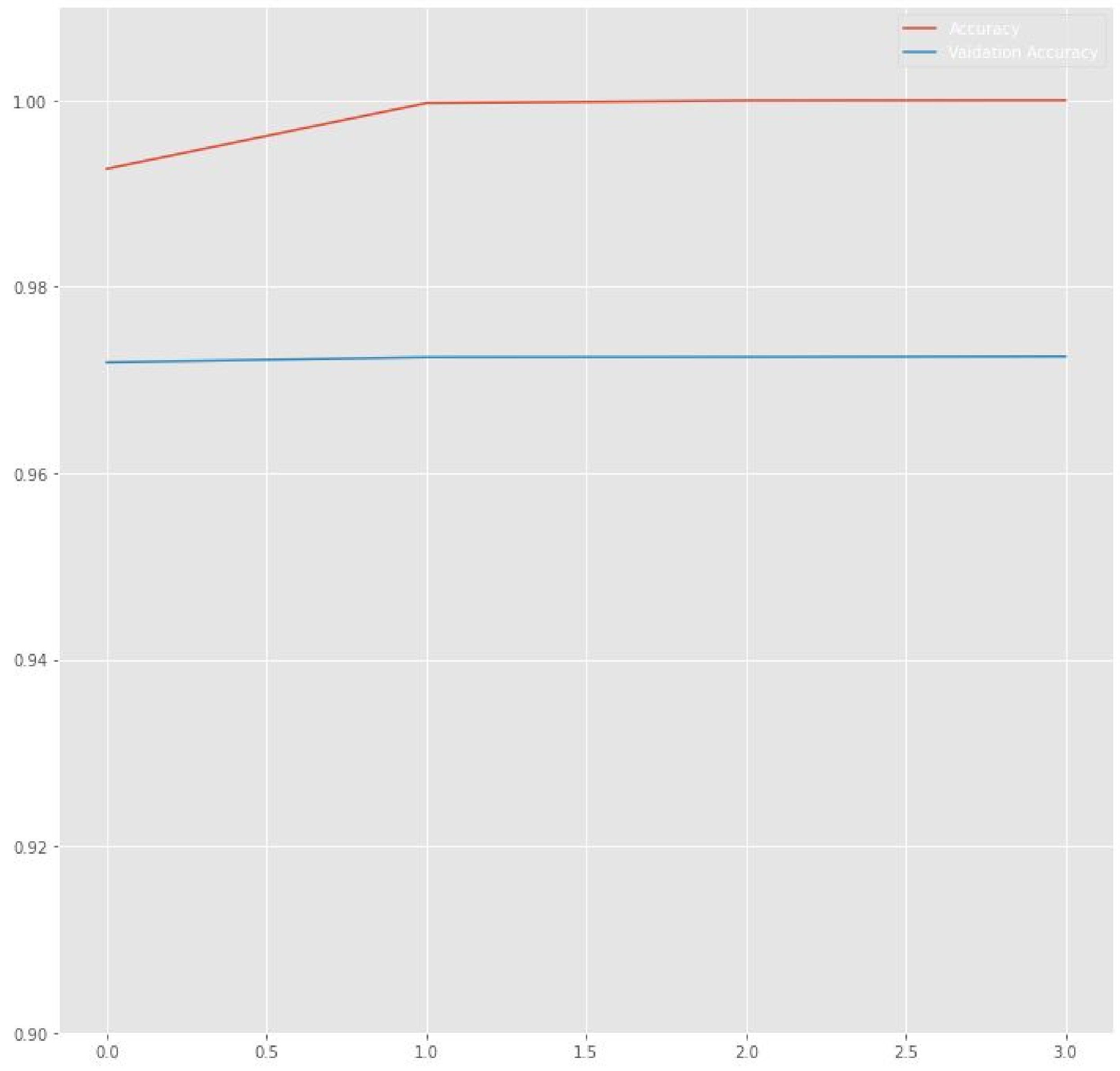
The notebooks under the Approaches/Deep learning/ have properly commented code for the Memory Tagger and Deep Learning.

Here are some of the decisions we made:

1. The length of the input sentences is restricted to 62 words, which is the maximum length for the twitter data. Sentences with length greater have their words dropped and the sentences with length less have been added with padding words. This helps us make the data of uniform length.
2. The number of neurons in the Bi-LSTM Layer is 128. The activation function is sigmoid and the recurrent activation function is the hyperbolic tangent function.
3. The time distributed dense layer takes in all the sequential data and converts it to meaningful information that can be used by the CRF Layer. It is actually a wrapper over a Dense layer.
4. The CRF layer is the final layer which takes into account the surroundings of the output tags to make the final prediction.
5. The batch size used for training is 256 samples (requires a high amount of ram, we used 32GB).
6. The number of epochs its runs for is 4.
7. The running time can be upto 1-2 hours, depending upon the computational speed.

   Note: The loss goes negative since the crf loss is calculated in such a way. The particulars are irrelevant for the purpose of this report, as this is used mostly in an abstracted form.

Assessments and Measures



The red line is the viterbi accuracy on the twitter training data.

The blue line is the validation viterbi accuracy on the GMB data.

The accuracies steadily rise during the first epoch and the validation accuracy is 97.19% at the end of the epoch. It rises to 97.24% at the end of the second iteration over the training data and approximately stays there for the rest of the training. This while, the training viterbi accuracy goes to 100%.

We can clearly see that the bi-directional dependencies given by the LSTM Layer and the contextual information encompassed by the CRF Layer are able to exhibit significant improvements over the simple Memory based Tagger. We have an outstanding validation accuracy of 97.25%, on the out-of-sample-data.

Conclusion

Traditional methods of NER as seen online will have much reduced accuracies in Tweets or Text data. Modern approaches should focus on context-dependent approaches, especially when concerning events or themes, such as movies or sports events. Best forms of NERs can be made highly efficiently using Deep Learning methods, and a combination of n-gram frequencies can help reduce possible errors in POS Tagging. Simple rule based approaches have low precision, and this can be eliminated by a data-driven approach, since modern Tweets provide high volumes of data.

References

- Ma, X. and Hovy, E. (2016). *End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF*. Retrieved from https://arxiv.org/abs/1603.01354
- Lample, G. and Ballesteros, M. and Subramanian, S. and Kawakami, K. and Dyer, C. (2016). *Neural architectures for named entity recognition*. Retrieved from https://arxiv.org/abs/1603.01360
- Jiampojamarn, S. and Cercone, N. and Keselj, V. *Biological named entity recognition using n-grams and classification methods*. Retrieved from http://vlado.cs.dal.ca/~vlado/papers/pacling05a.pdf
- Rituparna. (2018, September). FIFA World Cup 2018 Tweets, Version 4. Retrieved April 1, 2020 from https://www.kaggle.com/rgupta09/world-cup-2018-tweets
- Djam. (2018, June). FIFA WORLD CUP 2018 Players, Version 1. Retrieved April 1, 2020 from https://www.kaggle.com/djamshed/fifa-world-cup-2018-players
- Becklas, A. (2018, April). FIFA World Cup, Version 5. Retrieved April 1, 2020 from https://www.kaggle.com/abecklas/fifa-world-cup
- Walia, A. (2017, March). Annotated Corpus for Named Entity Recognition, Version 4. Retrieved April 15, 2020 from https://www.kaggle.com/abhinavwalia95/entity-annotated-corpus