GeoAI Challege Location Mention Recognition(LMR) from Social Media

Solution overview

Submitted by: Nabarup Maity(nabarupmaity@gmail.com)

Challenge link: https://zindi.africa/competitions/geoai-challege-location-mention-recognition-from-social-media

Preprocessing:

In the initial processing phase, I employed the spacy en-core small model to tokenize the input texts to prepare them for model compatibility. I made slight adjustments to the text tokenizer by adding a custom rule to split "@" as a separate token.

I have avoided the sentences having zero location mentioned. During the data processing stage, I have captured and retained essential information including the event name, tweet id, word, part of speech tag, start offset, and the word's target. Please refer the notebook Preprocess.ipynb for reference. This will produce 3 files train.csv, val.csv and test.csv.

Sample processed data looked like below.

	Event_Name	Sentence	Word	POS	start_offset	Tag
0	california_wildfires_2018	1061497252806414336	Please	INTJ	0	O
1	california_wildfires_2018	1061497252806414336	read	VERB	7	O
2	california_wildfires_2018	1061497252806414336	below	ADP	12	O
3	california_wildfires_2018	1061497252806414336	!	PUNCT	17	O
4	california_wildfires_2018	1061497252806414336	!	PUNCT	18	O
5	california_wildfires_2018	1061497252806414336	Another	DET	20	O
6	california_wildfires_2018	1061497252806414336	devastating	ADJ	28	O
7	california_wildfires_2018	1061497252806414336	fire	NOUN	40	O
8	california_wildfires_2018	1061497252806414336	has	AUX	45	O
9	california_wildfires_2018	1061497252806414336	hit	VERB	49	O
10	california_wildfires_2018	1061497252806414336	Northern	PROPN	53	LOC
11	california_wildfires_2018	1061497252806414336	California	PROPN	62	LOC

Data exploration:

I have started with calculating some statistics about the data like total number of sentences, the word count distribution, frequency distribution of tags. Frequency distribution of pos tags, most common tokens for the tags, maximum sentence.

Feature Engineering:

- The first step is to build a dictionary (word2index) that assigns a unique integer value to every word from the corpus. I constructed a reversed dictionary that maps indices to words (index2word).
- I have built a similar dictionary for the various tags(tag2index).
- Then I write a custom function(to_tuples()) that will iterate over each sentence, and form a tuple consisting of each token, the part of speech the token represents, and its tag. We apply this function to the entire dataset. The transformed version of the first sentence in the corpus looks like below

[('RT', 'PROPN', 'O', 721565484492197888, 0), ('@', 'ADP', 'O', 721565484492197888, 3), ('MailOnline', 'PROPN', 'O', 721565484492197888, 4), (':', 'PUNCT', 'O', 721565484492197888, 14), ('At', 'ADP', 'O', 721565484492197888, 16), ('least', 'ADJ', 'O', 721565484492197888, 19), ('41', 'NUM', 'O', 721565484492197888, 25), ('people', 'NOUN', 'O', 721565484492197888, 28), ('killed', 'VERB', 'O', 721565484492197888, 35), ('in', 'ADP', 'O', 721565484492197888, 42), ('Ecuador', 'PROPN', 'LOC', 721565484492197888, 45), ('earthquake', 'NOUN', 'O', 721565484492197888, 53), (',', 'PUNCT', 'O', 721565484492197888, 63), ('sparking', 'VERB', 'O', 721565484492197888, 65), ('tsunami', 'PROPN', 'O', 721565484492197888, 74), ('warning', 'VERB', 'O', 721565484492197888, 82)]

- Now I use this transformed dataset to extract the features (X) and labels (y) for the model. We can see what the first entries in X and y look like, after the two have been populated with words and tags. We can discard the part of speech data, as it is not n eeded for this specific implementation.

X['RT', '@', 'MailOnline', ':', 'At', 'least', '41', 'people', 'killed', 'in', 'Ecuador', 'earthquake', ',', 'sparking', 'tsunami', 'warning']
y: ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'LOC', 'O', 'O', 'O', 'O', 'O']

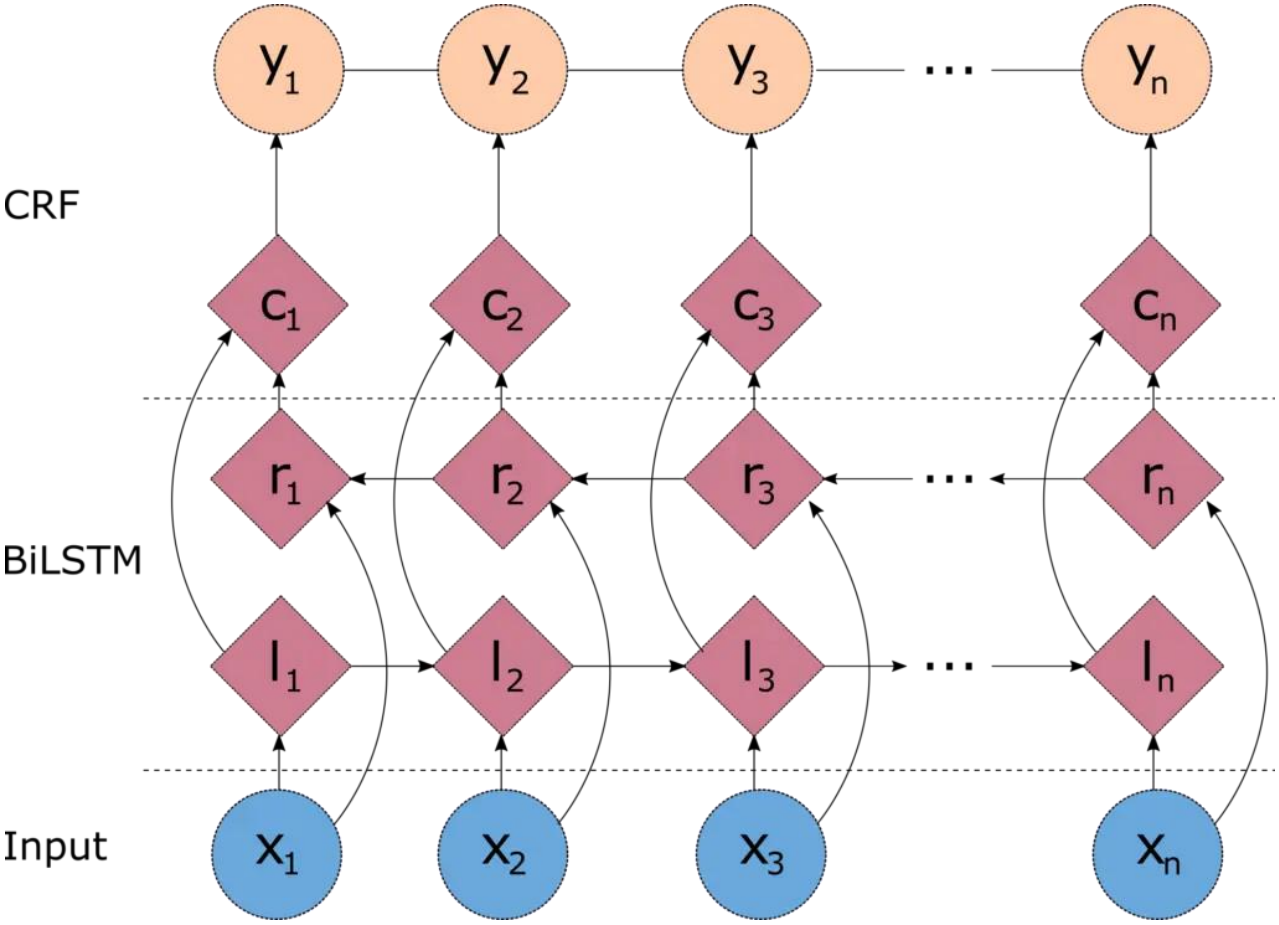
- We also need to replace each word with its corresponding index from the dictionary.

X [6755, 23736, 22875, 9466, 8302, 6313, 14810, 540, 7756, 14396, 16916, 23832, 6968, 2581, 15548, 2379]
y: [2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2]

- We see that the dataset has now been indexed. We also need to pad each sentence to the maximal sentence length in the corpus, as the LSTM model expects a fixed length input. This is where the extra "--PADDING--" key in the dictionary comes into play.
- And finally, I have converted everything into numpy arrays, as this makes feeding the data to the model simpler.

Modelling:

For the best solution in the LB I have used BiLSTM+CRF model(Lample et al 2016). The bidirectional LSTM consists of two LSTM networks - one takes the input in a forward direction, and a second one taking the input in a backward direction. Combining the outputs of the two networks yields a context that provides information on samples surrounding each individual token. The output of the BiLSTM is then fed to a linear chain CRF, which can generate predictions using this improved context. Its architecture is shown in below picture.



Please refer notebook model.ipynb for reference. It will take the processed data that we generate after preprocessing (train.csv, val.csv and test.csv) and as a step by step it consists of feature engineering, modelling and inferencing.

Data standards:

- For the given code it uses the train data for training model and validation data for model validation and test data for submission.
- As an external data I have used the pre-trained word vectors of glove twitter embedding with dimension of 100 that is of size <1 GB. Data is available at the following link (https://nlp.stanford.edu/projects/glove/)
- Packages: I have used the most recent version of python packages

Dependencies:

1. Spacy en_core_web_sm model(python -m spacy download en/ python -m spacy download en_core_web_sm)
2. Jsonlines(!pip install jsonlines)
3. plot_keras_history(!pip install plot_keras_history)
4. tensorflow_addons(!pip install tensorflow_addons)