CSCI 5832 Homework 3

Tuguluke Abulitibu

November 29, 2021

Abstract

This is a short report on homework 3: Named Entity Recognition. I spent the majority of my time trying to recreate huggingface's Transformer approach[1], and did not get a satisfying results: low F1 scores (never passed .37). Also, due to version dependencies and machine to train with pytorch, I could not even finish an end-to-end pipeline using BERT. Luckily I do have a baseline appaorch using conditonal random field, which is approved by Dr. Martin. One of the avantages via this approach is the quick training time on local machine, and without GPU trainning. Here are some of the numerical exeperimetns with a short explanations on how I failed to achieved the BERT approach.

1 Exploratory Data Analysis

After some base wrangling, we can get the following dataframe for the training set:

	Sentence	Word	Tag	Sentence #
0	1	Comparison	O	Sentence: 1
1	2	with	O	Sentence: 1
2	3	alkaline	B	Sentence: 1
3	4	phosphatases	I	Sentence: 1
4	5	and	O	Sentence: 1
⋮	⋮	⋮	⋮	⋮

Hence the unique count for each fields as

Fields	Unique Counts
Sentence	203201
Word	203189
Tag	203201
Sentence #	7398
O	179860
I	13559
B	9782

This obiviously shows that the majority of Tag is **O**. Any model will bias towards this predicitons. In fact by only labeling outout as **O**, shold return some decent F1 score for at least this tag.

2 Approach 1: Conditional Random Field

Conditional random fields (CRFs) are a class of statistical modeling methods often applied in pattern recognition and machine learning and used for structured prediction. Whereas a classifier predicts a label for a single sample without considering "neighboring" samples, a CRF can take context into account. To do so, the prediction is modeled as a graphical model, which implements dependencies between the predictions.[2]

[1] Fine-tuning Transformer for named-entity recognition

[2] Wiki

2.1 Training one the Original data set with no 'manipulation'

Although the F1 score should be on Name entity, and not Tag, but I had issues with **from nervaluate import collect_named_entities** library, so I had to refer my results to Tag level, since I figrue if that number is high enough, entity level F1 should not be very bad after all.

2.1.1 10 iterations and .25 split

	precision	recall	f1-score	support
O	0.91	0.98	0.95	36125
B	0.59	0.18	0.28	2097
I	0.62	0.34	0.44	2702
accuracy			0.90	40924
macro avg	0.71	0.50	0.55	40924
weighted avg	0.88	0.90	0.88	40924

2.1.2 10000 iterations, .2 train/test split

	precision	recall	f1-score	support
O	0.96	0.98	0.97	45228
B	0.81	0.68	0.74	2589
I	0.77	0.70	0.73	3429
accuracy			0.95	51246
macro avg	0.85	0.79	0.81	51246
weighted avg	0.94	0.95	0.94	51246

2.1.3 100000 iterations, .2 train/test split

	precision	recall	f1-score	support
O	0.97	0.98	0.97	36125
B	0.82	0.70	0.76	2097
I	0.78	0.72	0.75	2702
accuracy			0.95	40924
macro avg	0.86	0.80	0.83	40924
weighted avg	0.95	0.95	0.95	40924

We did not see a big improvement in F1 score. simplify doing more iteration will not do the trick.

2.2 How to improve the F1 score?

2.2.1 Add POS

All the online examples use POS as an addtional field for tagging, with the help of NLTK's **pos_tag** and **work_tokenize** packages, we can add a new column to the original dataframe:

	Sentence	Word	Tag	POS	Sentence #
0	1	Comparison	O	NNP	Sentence: 1
1	2	with	O	IN	Sentence: 1
2	3	alkaline	B	JJ	Sentence: 1
3	4	phosphatases	I	NNS	Sentence: 1
4	5	and	O	CC	Sentence: 1
⋮	⋮	⋮	⋮	⋮	⋮

2.2.2 10 iterations and .25 split.

Comparing to 10 iteration without POS tags,we already saw improvements of F1 scores (especially on **B** and **I**) .

	precision	recall	f1-score	support
O	0.93	0.96	0.94	45228
B	0.58	0.28	0.38	2589
I	0.50	0.46	0.48	3429
accuracy			0.89	51246
macro avg	0.67	0.57	0.60	51246
weighted avg	0.88	0.89	0.88	51246

2.2.3 10000 iterations, .25 train/test split

	precision	recall	f1-score	support
O	0.97	0.98	0.97	45228
B	0.81	0.69	0.74	2589
I	0.76	0.70	0.73	3429
accuracy			0.95	51246
macro avg	0.84	0.79	0.82	51246
weighted avg	0.94	0.95	0.94	51246

2.2.4 100000 iterations, .2 train/test split

But even after this many iterations, we still did not see huge improvement on F1.

	precision	recall	f1-score	support
O	0.97	0.98	0.97	36125
B	0.82	0.70	0.76	2097
I	0.76	0.71	0.74	2702
accuracy			0.95	40924
macro avg	0.85	0.80	0.82	40924
weighted avg	0.95	0.95	0.95	40924

2.3 Drop some 'O', so that 'I' and 'B' can get more representation

A lot of sentence only consists of **O**, dropping it, we downsize the dataset by 50%(121486$rows$). then

2.3.1 100000 iterations, .2 train/test split

	precision	recall	f1-score	support
O	0.97	0.98	0.97	36125
B	0.82	0.70	0.76	2097
I	0.76	0.71	0.74	2702
accuracy			0.95	40924
macro avg	0.85	0.80	0.82	40924
weighted avg	0.95	0.95	0.95	40924

2.3.2 Best result so far for 'I' and 'B'

	precision	recall	f1-score	support
O	0.96	0.96	0.96	19245
B	0.83	0.81	0.82	1972
I	0.80	0.78	0.79	2904
accuracy			0.93	24121
macro avg	0.86	0.85	0.86	24121
weighted avg	0.93	0.93	0.93	24121

with only 1225 iterations in fact.

```
Iter 1225 time=0.11  loss=5094.43  active=21467 feature_norm=110.93

L-BFGS terminated with the stopping criteria
Total seconds required for training: 81.587

Storing the model
Number of active features: 21467 (48900)
Number of active attributes: 16105 (39870)
Number of active labels: 3 (3)
Writing labels
Writing attributes
Writing feature references for transitions
Writing feature references for attributes
Seconds required: 0.076


```

2.4 Other idea I did not get the time to try

- Instead of just using IOB labels, we can try using the medical tagging from spacy or any fancy library to bread down the tagging, hence more features to learn, I am sure that will increase the accuracy.
- It is all about the traninng data, so we can scrap more online Gene dataset and inject into the training.
- Talking about feature learning, it is always good to try to learn more (until a certian point), so far online resource has the following features:

```
   features = {
  'bias': 1.0,
  'word.lower()': word.lower(),
  'word[-3:]': word[-3:],
  'word[-2:]': word[-2:],
  'word.isupper()': word.isupper(),
  'word.istitle()': word.istitle(),
  'word.isdigit()': word.isdigit(),
  #          'postag': postag,
  #          'postag[:2]': postag[:2],

```

We can definitely add some neighbour embedding and maybe even tag embeddding features, and learn from those.

- Dealing with unknown words in test set. I used groupby

```
df_gen_1 = pd.DataFrame(df_gen_test.groupby(['Sentence #'])['Word'].apply(list)).
    reset_index()
df_gen_1['word_string'] =df_gen_1['Word'].str.join(" ").apply(nltk.word_tokenize).apply
    (nltk.pos_tag)

```

then left join back with sentence number, so to (or I mistakenly think I do) get around special character tokenizations. There must be a clear way to deal with this, I just don't have time to figure out how. I did a quick cheat by assining the unknown ones all as O tag.

3 Output

I make two txt output, since I did not see the use of comma in the output format ($linenumber < tab > token < tab >, IOBtag$). So I had one with comma, one without for the testing purposes. You can see it in the name format.

4 Approach 2: Transformer and BERT

Failing gracefully. I should've listen and start with CRF first, or team up with someone (harder for me since remote and full-time).

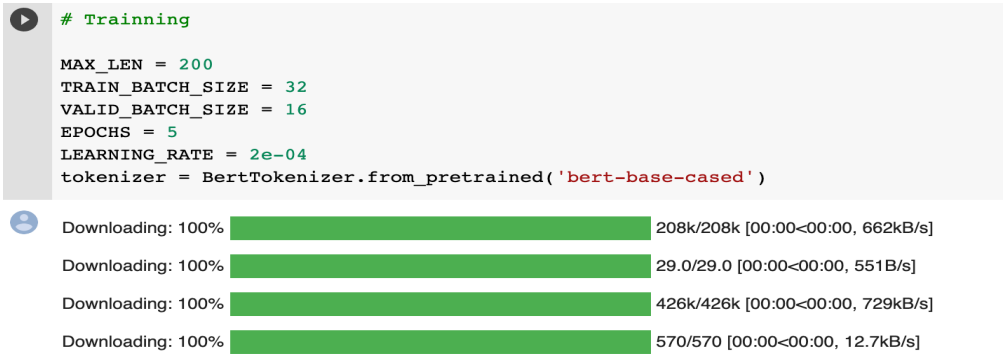

Figure 1: Even 5 epoch takes long time to run