Named-Entity Recognition in Resume

Cedric Yu[♮]

Abstract

We perform a named-entity recognition task on a dataset of resumes. Our approach is based on an assignment in the Sequence Models course on Coursera, offered by DeepLearning.AI. We expand it by studying the raw data, performing a more accurate and streamlined tokenisation, before re-training a Huggingface transformer model and evaluating the model performance on the F_1 scores of the entity classes on a validation set.

1 Problem Statement

We would like to perform a named-entity recognition task on a dataset of resumes. Namely, to each word/vocabulary appearing in a resume, we would like to assign a corresponding entity, such as 'Name' and 'Degree'. The tasks discussed in this report are all performed in the script `transformer_named_entity_recognition.py`.

1.1 Datasets

We are given a labeled dataset `resume/ner.json` of size 220. The instances are line-separated. Each instance is a dictionary of the form

```
{"content": <content text string>, "annotation": [{"label":[<label>],"points":[{"
    start":<int>,"end":<int>,"text":<text>}] } ], ,"extras":null}.
```

Here, in each instance, the value of `content` is a string of the cased raw text of the resume. `annotation` is a list containing the target labels (entities); In each element of this list, `label` is the target label (entity, e.g. 'Name' and 'Degree') for the `text` which is the part of the `content` it refers to. `start` and `end` respectively denote the positions of the starting and ending characters of the `text` in `content`. `extras` is not used.

For example, the first line of the dataset partially reads

```
{"content": "Abhishek Jha\nApplication Development Associate...","annotation":[{"label
    ":["Skills"],"points":[{"start":1295,"end":1621,"text":"\n... Oracle PeopleSoft\n
    ..."}]},..., {"label":["Name"],"points":[{"start":0,"end":11,"text":"Abhishek Jha"
    }]}],"extras":null}
```

In particular, the string `"Abhishek Jha"` in `content` has label `Name`, and its starting and ending positions in `content` are respectively 0 and 11.

1.2 Evaluation Metric

We will evaluate the model performance by Precision, Recall and F_1 score of each entity label, defined as

$$\text{Precision} \equiv (\text{True Positive})/(\text{True Positive} + \text{False Positive})$$

[♮]cedric.yu@nyu.edu. Last updated: November 2, 2021.

$$\text{Recall} \equiv (\text{True Positive})/(\text{True Positive} + \text{False Negative})$$

$$\text{F}_1 \equiv 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \times \text{True Positive}}{2 \times \text{True Positive} + \text{False Positive}\text{False Negative}}.$$

This is the F_β score with $\beta = 1$. When one prefers high precision (less false positives), one chooses a small $\beta < 1$, and when one prefers high recall (less false negative), one chooses a large $\beta > 1$; using the F_1 score means we wish to strike a balance between precision and recall.

2 Pre-Processing

A few preliminary observations are in order. The raw text (`content`) of each instance contains not only alphabets, numbers and white spaces, but also symbols such as parentheses and '•'. Some of these symbols are assigned labels— ideally one would like to only assign labels to alphabets and numbers— this is a shortcoming of the source dataset. The `start` and `end` of the labels sometimes also include trailing white spaces, which we would like to disregard. Besides, in order to meaningfully evaluate model performances, we need to tokenise the raw text and assign to each token corresponding label.

We thus arrive at the following strategy for pre-processing the labeled dataset:

1. load the labeled dataset (a `.json` file) and replace, in `content`, line break `'\n'` by `' '`
2. update `start` and `end` positions of each label by ignoring the trailing white spaces
3. train-validation split
4. tokenise the text in `content` and the labels
5. map labels to indices (aka 'tags')
6. for the use of a Huggingface transformer (distilbert-base-cased):
   further tokenise with the Huggingface tokeniser
   load pre-trained model and re-train with our dataset

The first two steps are done using the functions `convert_dataturks_to_spacy` and `trim_entity_spans` from the assignment without modification (other than adding comments). Our dataset is rather small (220), so we do a random 7-3 split into training and validation sets.

Tokenisation

Next, to tokenise the text in `content`, we first split the string white spaces and symbols:

```
list(filter(None, re.split(r'(\W\b|\b\W|[\s]+)', content)))
```

This is followed by dropping the white space tokens, then assigning the corresponding labels (entities) to the tokens (or `'Empty'` if there is no matching label). This step is achieved in the function `tokentize_dataset_tags`, which returns a list of the tokenised content and a list of the corresponding labels. (This step was also performed in the assignment, but we found some mismatch in the labels, so we use our own implementation.)

Note that the tokenised sequences have different lengths; we will truncate and pad using the DistilBert tokenizer.

From the tokenised training set, we find the label counts as follows:

```
Empty                   85440
Skills                   8273
Email Address            1930
Designation              1015
Degree                    938
College Name              889
Companies worked at       766
Location                  362
Name                      335
Graduation Year           161
Years of Experience        98
UNKNOWN                     7
```

Expectedly, the great majority of the tokens has `'Empty'` as label. Thus we have a very imbalanced class; accuracy is not a good evaluation metric.

DistilBert Tokenizer

Since we will use a pre-trained transformer model, we need to further tokenise the strings using a matching distilbert tokeniser. In particular, we will do this using `DistilBertTokenizerFast` together with the config from `distilbert-base-cased`. After this second tokenisation, we need to align the target tags. That is, the DistilBert tokeniser further divides some words into sub-words, and we assign to these sub-words the same tag is the original words. This is done in the function `tokenize_and_align_labels`.

Given the different sequence lengths, after this second tokentisation, we truncate/pad sequences into length 512. This will truncate some of the instances in the datasets. To evaluate the model performance, for those cases we will consider the tags associated to the truncated sequences, because that is what the model can predict. It turns out that of the 154 training instances, 105 (i.e. about two-thirds) need to be truncated.

3 Hugging Face Transformer

We use `TFDistilBertForTokenClassification` and the pre-trained parameters from `distilbert-base-cas` of the Hugging Face Transformers [1].

3.1 TFBertForSequenceClassification Model

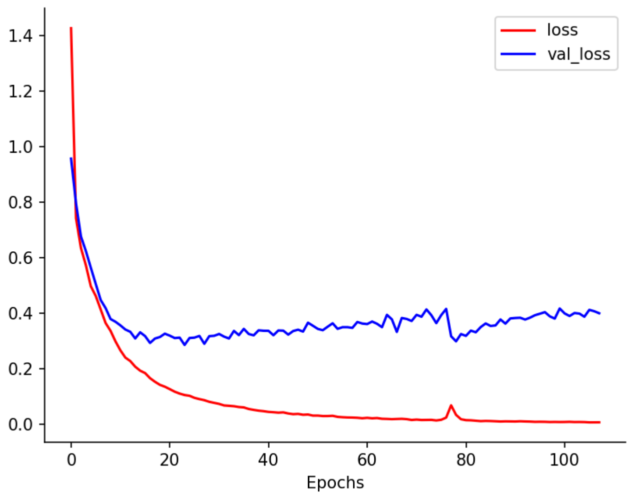
We load the transformer model `TFDistilBertForTokenClassification` with the pre-trained weights by

```
model = TFDistilBertForTokenClassification.from_pretrained('pre-trained-transformer-
    distilbert-base-cased/', num_labels=len(unique_tags))
```

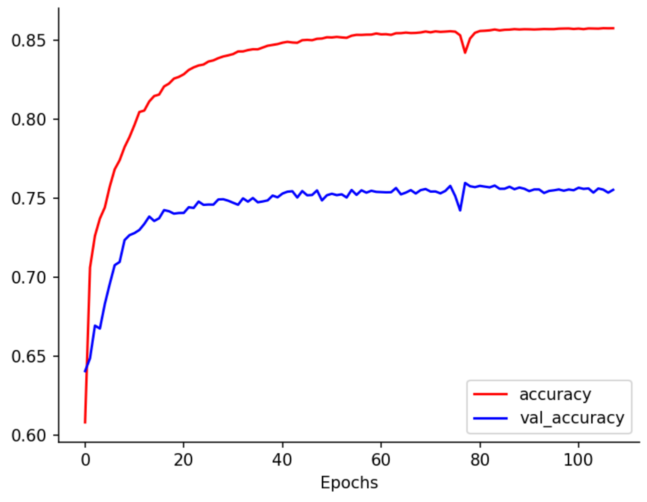
Next, we need to re-train the model with our training set. Note that this model predicts logits, so we need to add to the loss function and metric the parameter `from_logits=True`. With Adam at `learning_rate=5e-6`, we re-train the model for 100 epochs with batch size of 4 (with a larger batch size the model cannot fit into the VRAM of the local machine) with early stopping on the validation accuracy (I don't know how to use F_1 score for early stopping with Bert). The re-training stops at

```
# Epoch 108/200
# 39/39 [==============================] - 7s 187ms/step - loss: 0.0066 - accuracy:
    0.8575 - val_loss: 0.3992 - val_accuracy: 0.7552
```

The training curves are shown as follows:



(a) Training (red) and validation (blue) loss.



(b) Training (red) and validation (blue) auc.

Figure 1: (Re-)training curves for the pre-trained DistilBertTokenizerFast model.

So, apparently we have an overfitting problem, which can be attributed to the small dataset size.

Model Performance

Note that the accuracy is evaluated with the twice-tokenised sequences. What we want is the evaluating metrics on once-tokenised sequences. To this end, we re-combine the Bert-tokenised sub-words, and take the tag of the first sub-word of each word to be the predicted label. Using `seqeval.metrics.classification_report`, we find the Precision, Recall and the F_1 scores of the training and validation sets to be:

```
training set:
                       precision    recall  f1-score   support

                Name       1.00      1.00      1.00       155
 Years of Experience       1.00      1.00      1.00        33
              Degree       0.76      0.81      0.78       105
         Designation       0.97      0.96      0.97       284
              Skills       0.97      0.99      0.98      2128
```

```
       Email Address       0.99       1.00       0.99      1804
               Empty       1.00       1.00       1.00     42167
            Location       0.99       0.99       0.99       239
        College Name       0.77       0.87       0.82       101
  Companies worked at      0.97       0.96       0.97       336
     Graduation Year       0.93       0.67       0.78        82

           micro avg       0.99       1.00       1.00     47434
           macro avg       0.94       0.93       0.93     47434
        weighted avg       0.99       1.00       1.00     47434


validation set:
                Name       0.97       0.95       0.96        64
  Years of Experience      0.54       0.64       0.58        11
              Degree       0.59       0.64       0.61        53
         Designation       0.61       0.70       0.65       123
              Skills       0.93       0.73       0.82      1426
       Email Address       0.86       0.89       0.88       682
               Empty       0.96       0.98       0.97     16042
            Location       0.81       0.72       0.76        90
        College Name       0.41       0.61       0.49        56
  Companies worked at      0.67       0.63       0.65       139
     Graduation Year       0.67       0.36       0.47        33

           micro avg       0.94       0.95       0.94     18719
           macro avg       0.73       0.71       0.71     18719
        weighted avg       0.94       0.95       0.94     18719
```

This shows that the performance on the validation set is worse than on the training set. The overwhelmingly largest class ‘`Empty`’ nonetheless has high scores. But we are more interested in other non-trivial classes. To this end, we look at the macro average: the validation set has an F_1 score of 0.71, significantly less than that (0.93) of the training set. Looking at the individual classes, we find that the model performs well on the classes `Name`, `Skills`, `Location`, and worst on `Graduation Year`, `College Name` and `Years of Experience`. It is possible that the model has a hard time learning the numeral year tokens. The truncation of the tokenised sequences, as well as possible mis-labeled tokens from the dataset set may have also contributed to the low scores.

4 Outlook

We have analysed the problem with only the raw texts and labels given. In other applications, other useful features are given too, such as the parts-of-speech of the tokens (e.g. the Reuters Corpus [2]). It is instructive to explore the incorporation of theses extra features, perhaps in a custom LSTM-based network with embedding vectors, and see whether the model performance is better than just using the raw text.

5 Acknowledgment

Some of the codes are adopted (with modifications and optimisations) from the assignments of the Sequence Model course on Coursera, offered by DeepLearning.AI. We also acknowledge the authors from various sources online, whose tools and techniques were borrowed and implemented in our codes. (I didn't keep track of the references.)

References

[1] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR* **abs/1810.04805** (2018) , `arXiv:1810.04805`. `http://arxiv.org/abs/1810.04805`.

[2] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *ArXiv* **abs/1910.01108** (2019) .