Report for EMNLP HW2: Feature-based Linear Model and RNN for Event Trigger Detection

Abstract

In this homework, I implement 2 models for the event trigger detection: a feature-based Linear Model trained with the Perceptron Algorithm and an RNN using pretrained word embeddings.

1 Introduction

I take the event trigger detection task as a Sequence tagging task in this homework and implement a Linear Model and an RNN. For the linear model, I extract features directly from the text and train it with plain Perceptron Algorithm. For the RNN, the pretrained word embeddings, GloVe (Pennington et al., 2014), is used. The dataset contains 3439 training sentences with an average length of 23.72, 157 validation sentences with an average length of 21.03 and 426 test sentences with an average length of 27.05. Event triggers are labeled as 1, other tokens 0. There are 1.38 triggers in 1 sentence on average in train data, 1.42 in valid data and 1.61 in test data. Table **??** shows information about the dataset.

Dataset	size	avg length	avg trigger num
train	3439	23.72	1.38
valid	157	21.03	1.42
test	426	27.05	1.62

Table 1: Information about the dataset.

2 Methods

2.1 Feature-based Linear Model

A fundamental method to do the task is building a feature-based Linear Model. First, features are extracted from the text for each token in a sentence, including the token itself, its position, whether it is capitalized, whether it is a number, whether it is a punctuation, its Part of Speech(POS) tag, and its neighbors' features. The feature can be represented as an indicator function, for example:

$$f_i(x, y) = \begin{cases} 1, & \text{if } y = 1 \text{ and } POS(x) = \text{VB} \\ 0, & otherwise \end{cases}$$

To predict whether a token is an event trigger, scores are calculated for both cases:

$$score(x, y) = \sum_i \lambda_{f_i(x,y)} f_i(x, y)$$

$$,y \in \{0, 1\}.$$

$$y^* = \arg\max_y score(x, y).$$

Then, in training and testing stage, each token is treated as an instance. I use the Perceptron Algorithm to train the model, which makes it reasonable to use features directly without compressing them.

Algorithm 1 The Perceptron Algorithm

Input:

Training set $(x_k, y_k)_{k=1}^n$, $\lambda = [0, ..., 0]$

for each $kin[1, n]$ **do**
  $y_k^* = \arg\max_{y \in \{0,1\}} score(x_k, y)$
  if $y_k^* \neq y_k$ **then**
    $\lambda = \lambda + f(x_k, y_k) - f(x_k, y_k^*)$
  end if
end for

Output:

λ

2.2 RNN

RNN is an approproate structure to do sequence tagging task. However, the feature vectors used in Linear Model has a quite big dimension, which makes them hard to be reused in RNN. Therefore, I use pretrained word embeddings from GloVe instead. At each time step t, a prediction y_t^* is pro-

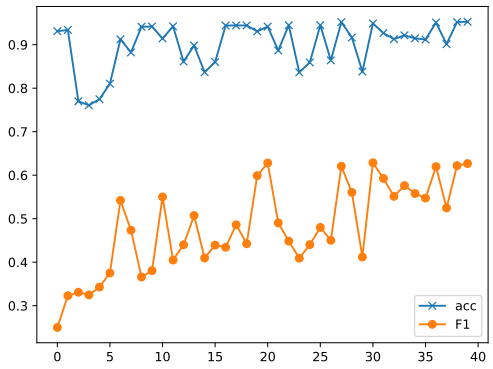

Figure 1: Validation results of Linear Model.

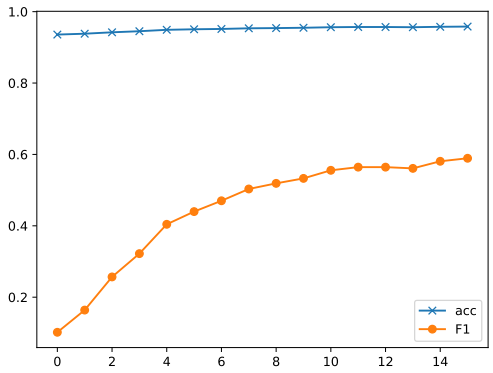

Figure 2: Validation results of RNN.

duced given a token x_t as input:

$$
\begin{aligned}
e_t &= Embedding(x_t), \\
i_t &= [e_t, h_{t-1}], \\
h_t &= W_1 \cdot i_t, \\
o_t &= Sigmoid(W_2 \cdot i_t), \\
y_t^* &= 1 \text{ if } o_t > 0.5 \text{ else } 0.
\end{aligned}
$$

Different from the case in my Linear Model, I treat each sentence as an instance when training and testing the RNN.

3 Experiment and Analysis

Model	Precision	Recall	F1
Linear Model	69.74	**66.91**	**68.29**
RNN	**83.86**	50.51	63.04

Table 2: The results of the 2 models.

For the feature-based Linear Model, I set the epoch number to 40. For RNN, I set the epoch number to 16, learning rate 0.01 and use MSE loss and SGD optimizer. Pretrained word embeddings are trainable here. Precision, recall and F1 are 3 evaluation metrics calculated at token level. The validation results in training state are shown in Figure 1 and 2. The loss curve is shown in Figure 3. The results are shown in Table 2. RNN has a high precision but a low recall. Linear Model is more balanced between precision and recall and therefore achieves a better F1 score. The curve of the Linear Model jitters a lot in training, while the curve of RNN is much more stable.

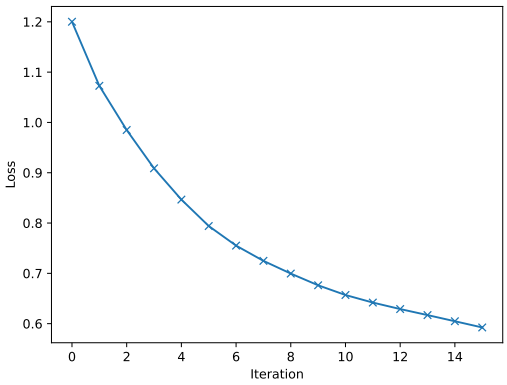

Figure 3: The loss curve of RNN.

Model	Type	Sent Length	Trigger Num	Most Freq POS
Linear	FP	27.72	1.44	VBD
	FN	31.08	2.19	NN
RNN	FP	28.48	1.45	NN
	FN	28.97	2.11	NN

Table 3: Analysis of results. For each wrongly predicted instance, calculate its sentence length, the number of triggers in this sentence and record its POS.

Besides, I check some information of the falsely predicted instances, which is shown in Table 3. Comparing with Table 1, it can be found that sentences which have wrongly predicted tokens tend to be longer. Longer sentences may includes more triggers and be harder to interpret by model. FP instances tend to locate in sentences with less triggers and FN instances tend to locate in sentences with more triggers. Tokens with POS tag `NN` are wrongly predicted most often, especially in FN instances, because verbs are more likely to be triggers.

4 Conclusion

In this homework, I implement a feature-based Linear Model and an RNN for the event trigger detection task and make a comparison between them. It can be inferred from the analysis that if the model can get rid the side effect of the POS tag, it may achieve better performance.

References

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.