HW2 NLP WET - REPORT

Yonathan Mizrahi – 209948728 & Raphaël Bellahsen – 931188684

Task division

Yonathan implemented models 1 and 2 while Raphaël worked on model 3.

Train

Description of the trained model

For model 1: In this model, we implemented an SVM model to predict the labels. First, we split our train and dev set into words and labels. We renamed our labels to 0 or 1. Then, for each word in the corpus, we compute its feature representation using the pre-trained Glove model Twitter-200[1] (previously imported). If a word in the corpus wasn't represented by the Glove model, we represented it by a vector set to 0. Then we trained our SVM using the feature representation of the train corpus and we tested it on the dev corpus.

For model 2: In this model, we implemented an FF model to predict the labels. We mostly used the code seen in tutorial 5. As we do in model 1, we reshaped our train and dev set into words and labels and we renamed our labels to 0 or 1. Then, for each word in the corpus, we compute its feature representation using the pre-trained Glove model Twitter-200[2] (previously imported). If a word in the corpus wasn't represented by the Glove model, we represented it by a vector set to 0. Then we trained a simple neural network with one hidden layer. The model was trained for 5 epochs using the Adam optimizer on the train.tagged set and tested on the dev.tagged set. To improve our model, we look if exists for unknown words similar words to the truncated words. If yes, it would take the mean vector of the 3 similar words. Unfortunately, this technique wasn't successful for us (lower f1 score results).

For model 3[3]: A DL model to predict named entity recognition using BLSTM and GloVe word embeddings using PyTorch. Two models have been trained using the corpus. The difficulty in correctly predicting the NER tag comes from encountering unknown words. If a word in not in the corpus, the model cannot be expected to correctly classify the NER tag for that word. However, strategies in handling unknown words can be utilized to lessen the severity of this issue. For example, adding *<UNK>* tags, and other variants such as *<UNK-UPPERCASE>* and *<UNK-NUMBER>*, assists the model in learning how to handle unknown words.

Let's transform `['@paulwalk', 'It', "'s", 'the', 'view', 'from', 'where', 'I', "'m", 'living', 'for', 'two', 'weeks', '.', 'Empire', 'State',`

[1] We also tested to implement our Glove model and also several others pre-trained models (), but we got our bests results using the twitter-200 Glove model.

[2] Again, the best results we got were using the twitter-200 Glove model.

[3] Reference to some of the idea discussed in this part: https://www.kaggle.com/code/rejpalcz/best-loss-function-for-f1-score-metric/notebook & https://arxiv.org/abs/2108.10566 .

```
'Building', '=', 'ESB', '.', 'Pretty', 'bad', 'storm', 'here', 'last',
'evening', '.'], ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'B-location', 'I-location', 'I-location', 'O', 'B-
location', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
```
into something more simple like : `['@paulwalk', 'It', "'s", 'the', 'view', 'from', 'where', 'I', "'m", 'living', 'for', 'two', 'weeks', '.', 'Empire', 'State', 'Building', '=', 'ESB', '.', 'Pretty', 'bad', 'storm', 'here', 'last', 'evening', '.'], ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'X', 'X', 'X', 'O', 'X', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']` something like only `'X'` something for the name entity and `'O'` for others.

Adding feature to our index :

```
  0 - PAD = special token used for the word ''
  1 - ALL_LOWER = when the word is all lower case, such as 'cat'
  2 - ALL_UPPER = when the word is all upper case, such as 'IBM'
  3 - FIRST_UPPER = when the first character is capitalized, such as
'John'
  4 - OTHERS = all other words that did not fit the categories above
```

The *CustomDataset* that we are instantiating is responsible for converting the data as retrieved from the function read_data() into pytorch tensors of indices . It will pad all sentences to the same length and convert words to indices using a voc. Lookup.

For example, the first sentences will be mapped into the dataset like this:

```
tensor([ 1,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
1, 16,1, 13, 17, 18, 19, 20, 21, 22, 13,  1,  1,  1,  1,  1,  1,  1,
1,  1, 1,  1,  1,  1,  1]),
tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 2, 1, 1,
1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
['@paulwalk', 'It', "'s", 'the', 'view', 'from', 'where', 'I', "'m",
'living', 'for', 'two', 'weeks', '.', 'Empire', 'State', 'Building',
'=', 'ESB', '.', 'Pretty', 'bad', 'storm', 'here', 'last', 'evening',
'.', '', '', '', '', '', '', '', '', '', '', '', '', '', '', ''],
```
we can see that the sentences is completed with padding ‘ ’

```
 tensor([1, 3, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 4, 3, 3, 3, 4, 2, 4, 3,
1, 1, 1, 1, 1, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Test

Results on Model 1

F1 score on the dev set for Model 1: 55.32 %

```
The classification report of our prediction is:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97     14483
           1       0.90      0.40      0.55      1250

    accuracy                           0.95     15733
   macro avg       0.93      0.70      0.76     15733
weighted avg       0.95      0.95      0.94     15733

The f1 score on the file dev.tagged is: 0.5532150776053215
```

We can see that we have "imbalanced data" since we have many more 0s samples than 1s. The fact that we have a “relatively small” number of True 1s samples in our datasets affects the performance of the classifier. Generally, to deal with that kind of issue, we can generate fraud words with label 1 to ‘re-balance’ the data[4]. However, in our case, the main issue came from the fact that a large number of words weren’t represented as a vector, and the fact we had to set them as a vector of 0[5] led to a low recall score.

Results on Model 2

F1 score on the dev set for Model 2: 58.36%

```
----------
Best Validation F1 Score on the test set during the train: 0.583680
The F1 Score on all the dev.tagged file is: 0.583680
```

As it was in model 2, the imbalanced data and also a large number of unknown words in our corpus affect the performance of the classifier.

Results on Model 3 / Competition:

F1 score on the dev set for Model 3: 64 to 72 % (depends on the train stage)

After implementing an LSTM model, we observed that the loss was decreasing but the f1 score was also decreasing. To address this issue, we added weight to the loss function. The optimal loss function for this situation would be the f1 score itself, but it is non-differentiable and therefore cannot be used as a loss function. We experimented with different weights and parameters for the SGD optimization algorithm and were able to achieve a max f1 score of 0.72. This score can potentially be improved by incorporating pos features from the NTLK library or by using a different loss function that takes the f1 score into account. We also tried using different activation functions but did not see any improvement in the results.

```
Now let's test the saved model2: CustomModel_CPU
Classification Report (including f1 score) on file dev
              precision    recall  f1-score   support

           1       0.96      0.99      0.98     14483
           2       0.86      0.53      0.66      1250

    accuracy                           0.96     15733
   macro avg       0.91      0.76      0.82     15733
weighted avg       0.95      0.96      0.95     15733
```
[6][7]

[4] We tested this solution in the model 2 however it led to almost the same f1 score on the dev.tagged set.
[5] During our test, we clearly saw that when we’re setting those unknown vectors with different values, it has a direct impact on our f1 score.
[6] Example from one of our runs (not the best one).
[7] Label 1 is for tag O and label 2 is all others tags

Estimation for competition files: 60-65%.