Sequence Tagging Project

Markus Gabriel
01326657

May 2018

The full source code for the project can be found at
`https://github.com/MGabr/seq-tagging-proj`.

1 Standard CRF predictor

As a standard CRF predictor I used the `CRF` implementation from `sklearn_crfsuite` with all hyperparameters left to their default values. I used the current token and the current, previous and next POS tag as input features to this CRF predictor. The evaluation results can be seen in table 1.

	Precision	Recall	$F_{\beta=1}$
ADJP	69.41%	77.16%	73.08
ADVP	78.29%	82.58%	80.38
CONJP	55.56%	62.50%	58.82
INTJ	50.00%	100.00%	66.67
LST	0.00%	0.00%	0.00
NP	92.11%	92.22%	92.17
PP	97.21%	95.29%	96.24
PRT	71.70%	73.08%	72.38
SBAR	79.07%	88.31%	83.43
VP	92.59%	93.17%	92.88
Overall	91.90%	92.29%	92.09

Table 1: Conlleval evaluation results for standard CRF predictor

2 Modern BiLSTM-CRF predictor

As a more modern predictor I chose the `LSTM` implementation from `keras.layers`. This required more preprocessing effort, additional layers and hyperparameter tuning.

2.1 Preprocessing

At first, I had to convert the features to a format supported by LSTM. A simple solution is one-hot encoding them. This is what I did for the POS tag and the chunk tag features using `Tokenizer` from `keras.preprocessing.text`. For the token feature this was, however, not possible since the dimension of the resulting dense matrix was to large to fit into memory and therefore produced a `MemoryError`. It was also not possible to use a sparse matrix since LSTM does only accept dense matrices. As a consequence of this issue I could not use the token feature from the CRF predictor "as-is" in the modern predictor as requested in the assignment. Using the token feature in it's word vector representation was, however, possible. To use it this way, it has to first be integer encoded which I did again using `Tokenizer`.

LSTM further requires a 3D matrix with shape (nb_samples, timesteps, input_dim) as input. In our case nb_samples is the number of sequences in our dataset, timesteps is the maximum length of a sequence and input_dim is the dimension of each feature (e.g. concatenation of token word vector and one-hot encoded POS tag). With sequence in the dataset I mean a sequence of (token, POS tag, chunk tag) tuples, called timesteps in the LSTM context, which are separated by empty lines from the following sequence. A first issue I had here was that the timesteps dimension was variable since the sequences had different sizes. To nevertheless be able to create a 3D matrix I padded all the sequences to the maximum sequence length with zeros using `pad_sequences` from `keras.preprocessing.sequence`. Since zero is a reserved index for the `Tokenizer` and not assigned to any word, the padding values do not conflict with existing values for words. To avoid, however, that these many padding values influence the training of the model I used a `Masking` layer which simply skips timesteps (tokens and POS tags) which are all zero. To then avoid skipping preditions for new words in the test data which would have an index of zero after tokenization, I mapped all out-of-vocabulary tokens to the index of the same dummy token. In my case with `oov_token="oov_token"` in `Tokenizer`.

2.2 Hyperparameters

To find good hyperparameters for my LSTM predictor, I followed the insights from Reimers et al. [1]. I therefore used the following hyperparameters.

- Nadam Optimizer
- Gradient Normalization with threshold 1
- CRF instead of softmax activation function
- Variational Dropout with value 0.1
- 2 bidirectional LSTM layers
- 100 recurrent units
- Batch size of 5 for training data and 1 for test data

The value 0.1 for the dropout was found as good value after some experimentation. Since it resulted in better performance I further used a `TimeDistributed Dense` layer between the last BiLSTM and the CRF layer which maps the BiLSTMs output dimension of 200 to our required output dimension (number of possible chunk tags). I trained the model for 25 epochs. Finally, it is notable that CRF itself is used here also after the BiLSTM layers.

2.3 Usage of "classical" CRF features

As already mentioned, it was not possible to use the token feature as input to the LSTM, since it would have to be one-hot encoded and therefore take up too much memory. Therefore we can only use the other remaining POS tag features as original CRF features to input to the LSTM layers. Since we, however, have bidirectional LSTM layers which already save and use state about previous and next features, we don't even need the previous and next POS tag feature. Therefore the only input is the current POS tag. Table 2 shows the evaluation results.

	Precision	Recall	$F_{\beta=1}$
ADJP	62.26%	59.27%	60.73
ADVP	69.02%	74.36%	71.60
CONJP	60.00%	33.33%	42.86
INTJ	100.00%	50.00%	66.67
LST	0.00%	0.00%	0.00
NP	92.33%	91.87%	92.10
PP	92.15%	94.72%	93.41
PRT	31.25%	28.30%	29.70
SBAR	72.91%	68.54%	70.66
VP	89.90%	91.19%	90.54
Overall	89.73%	90.23%	89.98

Table 2: Conlleval evaluation results for modern BiLSTM-CRF predictor using "classical" CRF features

2.4 Usage of word vector features

I used the glove 6B pre-trained word vectors with 100 dimensions. They can be used to build a weight matrix for the words from the training data which can then be used directly in a not trainable `Embedding` layer after the token input. The evaluation results can be viewed in table 3.

	Precision	Recall	$F_{\beta=1}$
ADJP	72.21%	66.59%	69.29
ADVP	75.66%	79.68%	77.62
CONJP	33.33%	55.56%	41.67
INTJ	100.00%	50.00%	66.67
LST	0.00%	0.00%	0.00
NP	91.94%	92.10%	92.02
PP	97.01%	97.34%	97.18
PRT	71.43%	75.47%	73.39
SBAR	86.81%	88.76%	87.78
VP	91.89%	90.29%	91.08
Overall	91.75%	91.70%	91.72

Table 3: Conlleval evaluation results for modern BiLSTM-CRF predictor using word vectors

2.5 Usage of both features

To use both features, I simply had to add a `Concatenate` layer in front of the first BiLSTM layer. This layer concatenates the word vector feature and the POS tag feature into one feature. The evaluation results for both features can be seen in table 4.

	Precision	Recall	$F_{\beta=1}$
ADJP	78.76%	75.51%	77.10
ADVP	80.53%	84.06%	82.26
CONJP	41.67%	55.56%	47.62
INTJ	100.00%	50.00%	66.67
LST	0.00%	0.00%	0.00
NP	94.95%	94.41%	94.68
PP	96.86%	98.04%	97.45
PRT	74.76%	72.64%	73.68
SBAR	88.64%	89.14%	88.89
VP	94.45%	93.56%	94.00
Overall	94.15%	94.00%	94.08

Table 4: Conlleval evaluation results for modern BiLSTM-CRF predictor using both features

3 Comparison of evaluation results

Comparing the different predictors and feature usages, we see that as expected the BiLSTM-CRF predictor using both features performs best. The CoNLL 2000 challenge was won with a F1 score of 93.48%, a similar BiLSTM-CRF approach later got a F1 score of 94.46% using Senna word embedding initialization

or 94.13% and another approach based on a voting classifier scheme achieved 95.23% accuracy [2]. Considering these results, my results with a F1-score of 94.08% seems to be okay, but could probably still be removed with hyperparameter tuning. The best chances for further improvement would probably lie in using other word vectors as also recommended in [1].
When using only word vector features or only the POS tag feature, the results are worse than when simply using CRF. Using only the POS tag feature the performance is significantly lower than when using word vectors (F1-score 89.98% against 91.72%). Using only word vectors with the modern BiLSTM-CRF approach, the F1-score comes at least rather close to CRF with a difference of only 0.37% between 91.72% and 92.09%.

References

[1] Nils Reimers and Iryna Gurevych. Optimal Hyperparameters for Deep LSTM-Networks for Sequence Labeling Tasks *In CoRR, 2017, Pages 1-10*

[2] Zhiheng Huang, Wei Xu and Kai Yu. Bidirectional LSTM-CRF Models for Sequence Tagging *In CoRR, 2015, abs/1508.01991*