Automatic role labeling

Emotion analysis – assignment 4

Carlotta Quensel
Felix Bühler
Maximilian Wegge

February 7, 2021

1. Introduction

As the task of sequence labelling is more advanced than whole-text emotion classification, we expect there to be more difficulties in automatic labelling as well as bigger differences between naïve and complex algorithms.

Thus, we decided to do automatic role labeling to confirm these expectations. The sequences for emotion experiencer, target or stimulus are mostly semantically determined and do not directly map onto syntactic structures, but the difference in syntactic structure between different training data might still hold an effect on the results. In this assignment, we look to answer if and how a naive and complex sequence labelling approach differ, as well as compare the influence of different training data on the results. With these objectives as our research questions, we now are able to decide on our method and data.

2. Method

Both methods are trained on the same data, so that we can compare the intersection between data and algorithm influences. Therefore, our first step in implementing a role labelling algorithm is to decide on the different data.

2.1. Training data

To keep the task mamageable, we only label one role, which is target. The semantic role of emotion target might in a naïve understanding correspond to the syntactic role of object (e.g. *I am angry at* ***you***.). This is of course not correct in many instances but still opens an interesting distinction between corpora. As the Reman corpus consists

of literary texts, GoodNewsEveryone of news headlines and Electoral tweets of tweets, these three corpora have very different syntactic styles, the news being abbreviated and the literature containing more complex syntactic contructs. While the distribution of our selected role is not ideal between the corpora (Reman only contains around 700 target instances), going forward, these are the three corpora used for training and evaluating both algorithms.

2.2. Naïve approach – Hidden Markov model

Sequence labelling in general and the role labelling of emotion target specifically is dependent on the word order, sentence semantic and therefore also syntactical information. While both approaches only consider the tokens as labelling information, while the complex method learns to hopefully detect underlying structures in the token order, the simple method only considers the order of labels. As a basic sequence labelling method, we chose to train a Hidden Markov model and then determine the best label sequence using a viterbi algorithm (the code can be found at the end of this documentation). The model is trained with the token-label pair frequencies relative to the token frequency as emission probabilities and the label bigram frequencies relative to the second token as transition probabilities. An estimated best label sequence therefore combines the most probable label sequence with the highest possibilities of labels for each individual token. The viterbi algorithm is then used to compute this most probable label sequence for a given token sequence.

While Hidden Markov models have many applications in natural language processing, as a sequence labelling algorithm, it is most frequently used for POS-tagging. In contrast to POS-tagging, a HMM in our case has a deficit, as the tokens are not connected to the tags as strongly. Even though a token can change its syntactical category according to its place in the sentence, there are generally only a few possible labels it can take on. For emotion role labelling on the other hand, a sequence like *my mother* can be the emotion target as well as the experiencer or even the stimulus. The classification mostly depends on semantic information which might be transported through syntactic structure, which our approach does not take into account, as the tokens themselves are only counted as unigrams. Thus, we use a more complex learning approach to combat the problem of missing context.

2.3. Complex approach – Transformer

For the complex approach, we chose RoBERTa. This deep learning method is an extension of the BERT transformer, which means it is pre-trained on external data. We used 'RoBERTa' to convert the tokenized sentences into an input matrix. This way we can use the whole sequence as context for each individual token. We set a maximum word amount to 100 words, because most of the sequences in our corpora are shorter. The first layer is the pre-trained RoBERTa model[1], which was not re-trained, as it has 124 million

[1]RoBERTa distribution: `https://huggingface.co/transformers/model_doc/roberta.html`

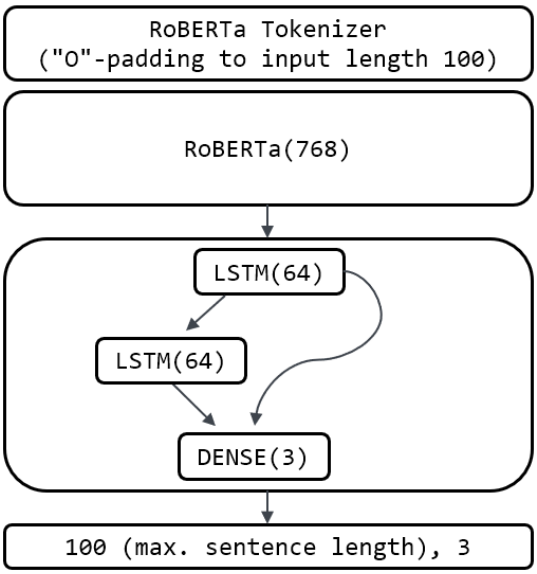

Fig. 1 Model architecture of the RoBERTa with an input layer size of 100 and an output of 3 ('O', 'B' and 'I') for each of the 100 input dimensions.

parameters and therefore superseeds our spacial and temporal constraints. The output is then fed into a bidirectional LSTM with 64 units. Its output is in turn fed into another bidirectional LSTM of the same make-up. Then we used a Dense-Layer to combine all the features. Thus, the model has the architecture shown in figure 1. Adding a residual connection between the first LSTM and the Dense-Layer improved our accuracy. These layers are used in a Time-Distributed-Layer to produce a prediction for each token. For learning we also added 'ReduceLROnPlateau' to reduce the learning-rate for internal metrics when learning stagnates.

A difficulty with this approach is the sentence length. Since all inputs have the same length, we pad shorter sequences with additional 'O's, thus the model predicts 'outside' at a disproportionate frequency considering the real label counts in the corpora. Reducing the amount of words could improve the predictions, but important information could be cut off (only a problem with the Reman data, as the sentences are very complex and long). Using RoBERTa also influenced the time requirements of our training, e.g. training on "Good News Everyone" took about 3 hours.

3. Evaluation

In opposition to token-wise evaluation, evaluation of sequences can be ambiguous. In this approach to evaluation, we count intersections between the gold and predicted sequence as correct labels (True Positives) without differentiating between 'B' and 'I'. When a gold label sequence spans more than one predicted sequence, we only count one of these multiples and ignore the rest.

Analogously, incorrect predictions are defined as empty intersections: False Negatives

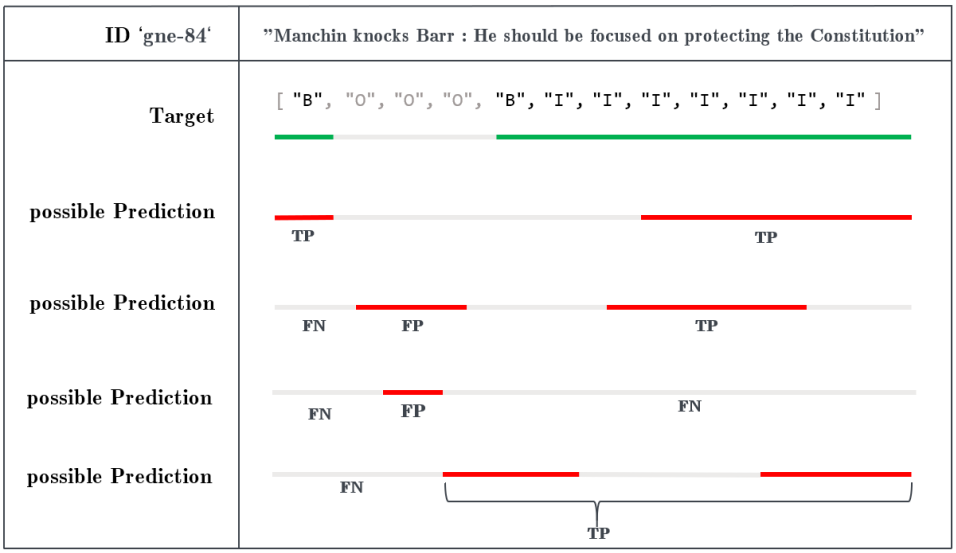

Fig. 2 Example predictions for the target sequence and their categorization as correct and incorrect.

are counted if the model does not predict a target sequence, False Positives if it predicts a non-target sequence. In this evaluation, we do not count True Negatives (correctly predicted non-target sequence), as they do not contribute to precision, recall and f-score for the target sequence and would add further ambiguity to the evaluation process. Figure 2 illustrates this approach with some examples (note that the predictions are created manually to best show different cases and do not represent the actual output of our models). In applying this method, we yield the following results for both models and all three training corpora respectively.

		Gold	
		In	**Out**
Predicted	**In**	2132	5788
	Out	1559	n.a.

precision:	0.269
recall:	0.578
f-score:	0.370

(a) trained on GNE

		Gold	
		In	**Out**
Predicted	**In**	4365	11739
	Out	1026	n.a.

precision:	0.271
recall:	0.810
f-score:	0.406

(b) trained on Elec. Tweets

		Gold	
		In	**Out**
Predicted	**In**	6335	11087
	Out	1099	n.a.

precision:	0.364
recall:	0.852
f-score:	0.510

(c) trained on Reman

Fig. 3 All three Hidden Markov models with viterbi predictions that are trained on one corpus, and evaluated on the other two.

		Gold	
		In	**Out**
Predicted	**In**	x	x
	Out	x	n.a.
precision:		0.x	
recall:		0.x	
f-score:		0.x	

(a) trained on GNE

		Gold	
		In	**Out**
Predicted	**In**	15	57
	Out	4552	n.a.
precision:		0.208	
recall:		0.003	
f-score:		0.006	

(b) trained on Elec. Tweets

		Gold	
		In	**Out**
Predicted	**In**	112	330
	Out	7322	n.a.
precision:		0.253	
recall:		0.015	
f-score:		0.028	

(c) trained on Reman

Fig. 4 Three RoBERTa models each trained on GNE, Electoral-Tweets, or Reman and evaluated on the other two corpora.

3.1. Comparison of models

As shown in the figures 3 and 4, the naïve approach of our Hidden Markov model surprisingly produces significantly better results than the more complex RoBERTa. This can be partly explained by both methods' training requirements. The HMM bases the predicted sequence partially on the most likely tag transitions, which means that 'B' or 'I' is more often than not followed by another 'I', resulting in longer and more predicted sequences. The transformer on the other hand needs an input of constant length, as explained in section 2.3. By evaluating the models together with the above explained evaluation method, the HMM gets an advantage over the transformer.

However, the HMM predicts too many and too long sequences, as can be seen by the low precision in contrast to the relatively high recall. Due to our evaluation method, a model predicting the whole text as one sequence will lead to a precision and recall of 1, as whole sequences are always counted as correct if they intersect with the gold sequence, even if they also intersect large sequences of outside tokens. In opposition to the transformer, it is too greedy.

The transformer learns too many 'O's from the padded sequences and will therefore predict much less sequences and often predict no emotion target in the sentence at all. Thus, the recall is bad (between 0 and 1% of target sequences are found) while the precision is worse but comparable to that of the HMM.

The faults of both models are clearly visible when comparing results directly. The sentence below (ID: gne-1547) has a gold target sequence marked in bold which for human annotators would be fairly simple to annotate.

These Artists Want to Blow Up the ***Whole Financial System***

HMM:

B I I I I I I I I I

RoBERTa:

O O O O O O O O O O

When comparing the predictions of the Hidden Markov model and the transformer, both of which are trained on the same data from the Reman corpus, we see the opposition of the models' problems. While the HMM predicts the whole sentence to be a target, the RoBERTa model predicts no target sequence at all. For further evaluation, we can therefore conclude that both models struggle with a meaningful recall, either the score is not informative or too low.

As the precision for both models is quite low, we might consider that a token only approach is not the best method for this task of sequence labelling. Instead, the model could additionaly take POS-tags or even simple syntactical sequences (chunks) as input. The Hidden Markov model is an Ngram model and could thus be extended to consider token bigrams instead of unigrams in its emission probabilities.

3.2. Comparison of training corpora

For the simpler model, the training data that leads to the best results is the Reman corpus. This is somewhat surprising, as it includes the least amount of annotated emotion targets and is thus very sparse for training. On the other hand, as the texts in the corpus are sentences from literature, the original writer likely structured them deliberately, leaving less room for errors that result from bad grammar or colloquial language. Reman significantly improves the precision with regard to the other training corpora, as the biggest flaw of annotating too many or too long sequences is somewhat reduced with this training corpus.

The research question did not include a hypothesis for which corpus might be the best training data to prevent biased evaluation, but this result nonetheless contradicts our expectation. Not only is Reman sparse in texts with target annotations, the longer texts also means that it includes less sequences in relation to text length. In combination with the complex sentence structures, this leads to the assumption that the corpus might only perform well with other literary texts but not on texts as different in genre as colloquial tweets and telegraph style news headlines. The training corpus with the worst HMM is GoodNewsEveryone. As its texts are news headlines, they are the shortest in comparison to the other data sets, which hightens the possibilities for a target sequence. Conversely to the influence of Reman texts combatting the greediness of the model, the short text length here amplifies it.

For the transformer, this same reasoning results in Reman perfoming the worst out of all training data, as the problem of sparse input matrices is amplified by the corpus' inherent sparseness. Inversely, here GoodNewsEveryone leads to the best results. This might originate from the pre-trained RoBERTa layer, which could map particularly well onto the terms and sentence structure of the news headlines and therefore result in more efficient training through compatible data.

4. Conclusion

While the task of sequence labelling is complex, this comparison shows that a complex task does not translate to a need for a complex model. This is particulary important when considering the training restrictions placed on the model, as a simple method might perform similar with more or less training data or time, while a deep learning model will suffer significantly from too little training iterations or data. Likewise, in this task the transformer's performance relied heavily on the chosen training data while the simple model had similar (but significantly differing) results regardless of training data. This shows the increased care necessary when dealing with complex methods overall. From the low precision of both methods, a need for further information can be derived. As explained above, this can take the form of a longer pipeline that includes POS-tags or chunks, as well as more training epochs specifically for the pre-trained RoBERTa. The difference in results regarding the training corpora once again shows the importance of balanced and representative training data that fits the intended use and the method.

A. Code – HMM and Viterbi

The simple model is implemented from scratch as the code below. As the transformer mainly consists of application of library methods, the code is not shown.

```
class HiddenMarkovModel:
    transitions = dict()
    emissions = dict()
    prior = dict()

    def __init__(self):
        self.transitions = {'O': {'O': 0, 'B': 0, 'I': 0},
                            'B': {'O': 0, 'B': 0, 'I': 0},
                            'I': {'O': 0, 'B': 0, 'I': 0}}
        self.prior = {'O': 0, 'B': 0, 'I': 0}

    def train_probabilities(self, train_set: pandas.DataFrame):
        trans = {'O': {'O': 0, 'B': 0, 'I': 0},
                 'B': {'O': 0, 'B': 0, 'I': 0},
                 'I': {'O': 0, 'B': 0, 'I': 0}}
        emi = dict()
        prior = {'O': 0, 'B': 0, 'I': 0}
        for index, row in train_set.iterrows():
            # The current sequence consists of (word, tag) pairs
            seq = list(zip(row['tokens'], row['target']))

            for i in range(len(seq)):
                tok = seq[i][0]
                tag = seq[i][1]
                # First tag: Prior count
                if i == 0:
                    prior[tag] += 1
                # Every other tag: Transition count from the previous tag
                else:
                    trans[tag][seq[i - 1][1]] += 1
                # Emission count from token-tag-pairs
                if tok in emi:
                    emi[tok][tag] += 1
                else:
                    emi[tok] = {'O': 0, 'B': 0, 'I': 0}
                    emi[tok][tag] += 1
        # Converting the absolute emission/transition/prior frequencies in
        for tag in {'O', 'B', 'I'}:
            freq = sum(trans[tag].values())
```

```
            for prev_tag in trans[tag]:
                trans[tag][prev_tag] /= freq
            prior[tag] /= sum(prior.values())

        for tok in emi:
            freq = sum(emi[tok].values())
            for tag in {'O', 'B', 'I'}:
                emi[tok][tag] /= freq
        # Unknown words get a random tag
        emi['$OOV'] = {tag: sum(trans[tag].values()) /
                            sum([sum(trans[tag2].values())
                                 for tag2 in {'O', 'B', 'I'}])
                            for tag in {'O', 'B', 'I'}}
        self.transitions = trans
        self.emissions = emi
        self.prior = prior


def viterbi(sequence: list[str], model: HiddenMarkovModel) -> list[str]:
    viterbi_table = [[0 for _ in range(3)] for _ in range(len(sequence))]
    max_labels = list()
    # Go through the sequence token by token
    for i_tok, token in enumerate(sequence):
        # For each possible Tag, multiply
        for i_tag, tag in enumerate(['O', 'B', 'I']):
            # the emission probability of the tag and current token
            if token in model.emissions:
                viterbi_table[i_tok][i_tag] = model.emissions[token][tag]
            else:
                viterbi_table[i_tok][i_tag] = model.emissions['$OOV'][tag]
            # with the prior probability of the tag for the first token
            if i_tok == 0:
                viterbi_table[i_tok][i_tag] *= model.prior[tag]
            # or with the transition probability for best tag of the last
            else:
                viterbi_table[i_tok][i_tag] *= max_v * model.transitions[t
        max_v = max(viterbi_table[i_tok])
        max_t = ['O', 'B', 'I'][viterbi_table[i_tok].index(max_v)]
    indices = {0: 'O', 1: 'B', 2: 'I'}
    # Read the best tags from the table and return the sequence
    for i in range(len(sequence)):
        max_labels.append(indices[viterbi_table[i].index(max(viterbi_table
    return max_labels
```