Concept Sequence Tagging for a Movie Domain with CRF Language Understanding Systems, Second Project

Davide Pedranz
Mat. number 189295
davide.pedranz@studenti.unitn.it

Abstract

Concept Sequence Tagging is a fundamental task for any Spoken Language Understanding (SLU) system. Concepts can be used to understand the semantic of user's requests and give appropriate replies. In this project, we trained Conditional Random Fields (CRF) classifiers to extract concepts from sentences taken from the movie domain. Then, we compared the performances with the Weighted Finite State Transducers (WFST) generative models developed in the mid-term project.

1 Introduction

Concept tagging can be defined as the extraction of concepts out of a given word sequence, where a concept represents the smallest unit of meaning that is relevant for a specific task. The extracted concepts can be used by the following blocks of a SLU pipeline, like a dialog management system, to understand the semantic of user's requests and build a appropriate replies.

In this work, we trained CRF classifiers to extract concepts sentences in the movie domain, build using the `CRF++`[1] toolkit. We will give a brief description of CRF models. Then, we will present the data set and the features used for the classification. We will present the different experiments performed and the obtained results. Finally, we will compare the CRF to the WFST models of the mid-term project.

2 Conditional Random Fields

CRF are log-linear models for labelling and segmenting sequential data (Elkan, 2008). In general, log-linear models describe the probability of the label y given the example x as:

$$p(y|x;w) = \frac{exp \sum_{j=1}^{J} F_j(x,y)}{Z(x,w)},$$

where w is a vector of weights that describes the importance of each feature and $Z(x,w)$ is a normalization factor. F_j are feature functions, i.e. measures of the compatibility between example x and label y. In CRF, the features are functions of the sentence x, the current label y_i, the previous label y_{i-1} and the position i; the output is usually binary, either 0 or 1. An example of feature could be: "1 if the word x_i is capitalized and the label y_i is `actor.name`, 0 otherwise".

In CRF anything can be a feature. It is possible to combine multiple features together to obtain new more complex ones or even generate new features from the data set. For instance, if one observes that all words starting with a given prefix correspond to the concept A, one can add a feature that takes word prefixes into account.

3 Data Set

The data set used is a corpora of sentences from the movie domain. The provided features are: words, Part of Speech (POS) tags and words stems. The aim is to extract concepts such as the actor name or the movie release date in the Inside, Outside, Beginning (IOB) format. Please checkout the mid-term project for a detailed description of the data set and the concepts distribution.

3.1 Preprocessing

Before training the classifiers, some additional features were computed from the available data:

- the word's stem is capitalized;
- the word is a language;

[1] https://taku910.github.io/crfpp/

- the word prefix (of 1, 2 and 3 letters);
- the word suffix (of 1, 2 and 3 letters).

4 Experiments

4.1 Words

The first trial was to train a CRF using only the words of the sentence. The result was used as baseline for the performances evaluation of more complex models. We experimented using different window sizes, i.e. creating a feature function for the current word and the preceding and following n words, with $n \in [0, 6]$. In addition, we tried to enable the bigram templates of `CRF++`, i.e. considering also the label of the previous word to classify the current one.

n	F1, no bigram	F1, bigram
0	51.60%	72.11%
1	69.18%	79.71%
2	75.52%	81.39%
3	77.21%	80.99%
4	77.69%	80.21%
5	76.76%	80.23%
6	76.70%	80.04%

Table 1: Performances of the models using only the words. For each window size, the model is trained with and without the bigram template.

Table 1 summaries the performances of the different models. In all cases, the bigram template helps to get significantly higher performances. The best model is the one with bigrams and window size 2. The concept of the previous word seems to be very important to better classify the current one (bigram template). The performances improve window a size up to 2, then degrades for larger ones. The words very distant from the current one seems to have little or no effect on the semantic of the current one. Taking them into account makes the model overfit the training data.

4.2 POS

The next step was to try the other features initially provided, i.e. POS and word stems. Given the results presented in Section 4.1, we considered only templates using the bigrams.

We started to add POS features to the best template in the previous section (windows size of 2), using again the window approach with $n \in [0, 4]$. The results are shown in Table 2. As expected, the POS features improved the performances of the classifier. The best model was obtained for $n = 2$. Similarly to the previous models, a large window seems to overfit the training data.

n	prec.	rec.	F1
0	87.25%	76.54%	81.54%
1	87.34%	76.54%	81.58%
2	87.50%	76.35%	81.55%
3	86.97%	76.44%	81.37%
4	87.30%	76.26%	81.41%

Table 2: Performances of the models using words and POS tags for different window sizes.

4.3 Stems

We repeated the same procedure for the word stems, using a window size $n \in [0, 4]$. The stem feature helped to obtain a slightly improvement, even though not significant. A window size of 0 made the classifier performances slightly better, while all other attempts gave a worsening. The results are reported in Table 3.

n	prec.	rec.	F1
0	87.36%	76.63%	81.64%
1	86.98%	76.54%	81.42%
2	87.02%	76.17%	81.23%
3	87.02%	76.17%	81.23%
4	87.30%	76.26%	81.41%

Table 3: Performances of the models using words, POS tags and stems for different window sizes.

4.4 Additional Features

Since many concepts spans on multiple words, we added features for sequences of successive 2 words, POS tags and stems. Then, we tried to combine words and POS tags of closed words. Unfortunately, the performances decreased in both cases.

Since both POS and Stems seemed to be useful for the classifier, we tried combine larger window sizes for words, POS and stems with the new addition features. After some attempts, we were able to increase the performances of the model to F1 of 82.63%.

Finally, we added the features generated from the available ones, as described in section Section 3.1.

The features "stem is capitalized" and "word is a language" did not help the classifier at all. On the other hand, prefixes and suffixes improved performances, getting a F1 score of 83.44%

4.5 Regularization

We tried different regularization parameters (`c` option in `CRF++`) for the best model found so far. Table 4 show the performances for different `c` values. We can notice that the parameter slightly influences the final performances.

c	prec.	rec.	F1
0.5	86.72%	80.20%	83.33%
0.75	86.70%	80.66%	83.57%
1	86.44%	80.66%	83.45%
1.25	86.37%	80.75%	83.47%
1.5	86.22%	80.84%	83.44%
1.75	86.16%	81.03%	83.51%
2	86.09%	81.12%	83.53%

Table 4: Performances of the the best CRF model, trained using different regularization parameters.

4.6 Genetic Algorithm

In order to select the best feature functions for the concept tagging task, we decided to the features selection as a black-box optimization problem. We developed a genetic algorithm using the Python `DEAP`[2] library (Fortin et al., 2012). Following the genetic analogy (Battiti and Brunato, 2014), we treated a CRF template as an individual and a feature function as a gene. A set of individuals form a population.

4.6.1 Initialization

The algorithm is divided in two phases. The first one consists in the population initialization. In our case, the initial population is composed by the union of the best scoring CRF models with some randomly generated individuals. To generate a new individual, it is sufficient to randomly select a set of genes. The bigram options is automatically added to the entire population to guarantee better performances.

4.7 Evolution

The second phase is the evolution. The evolution lasts several epochs. Each epoch generates a new population starting from the current one, trying to select the best individuals. After the last iteration, the population contains the best individuals found during the search. At each iteration, the following steps are performed.

Selection The population is evaluated using the fitness function, in our case the F1 score on the test data. The individuals are paired using some stochastic scheme. The best ones are selected for the following phases, while the others are discarded.

Reproduction The selected individuals have a certain probability mate with each other. The mated pairs generate a child by randomly choosing among their genes, in this case the feature functions. The new individual is then added to current the population.

Mutation All individuals in the population have a certain probability to mutate, i.e. change some of their genes. Possible mutations are: gene deletion, gene insertion and gene substitution. Multiple mutations can append at the same time. The mutated population is used for the next epoch.

4.7.1 Results

In few epochs, the genetic algorithm managed to achieve the same performances of the best manually written model. With some additional epochs of training, the algorithm managed to outperform it, reaching a F1 score of 84.15%. The generated template contains many features presented in the previous sections as well as some more complex features selected by the genetic algorithm. It is interesting to notice that some features are duplicated in many individuals generated during the search. The removal of the duplicates causes a slightly worsening of the performances. The `CRF++` internal optimization routine was probably not able to find the optimal weights for the feature functions, so the duplicates helped to compute better weights.

We believe that it is possible to obtain even better results by tuning the meta-parameters of the genetic algorithm and running the algorithm for some additional epochs. This exploration is left as a future work.

Changing the regularization parameter did not improve the performances of the best model. The genetic algorithm seems to be able to automatically pick the best features for the given regularization parameter.

[2] https://github.com/DEAP/deap

5 Comparison

5.1 Baseline

The best WFST model in the mid-term project used a 4-gram language model for the concept tags and achieved a F1 score of 82.74%. The baseline CRF model described in Section 4.1 achieved a F1 score of 81.39%. Both models used only the words feature. Due to the `CRF++` limitations and the model complexity, CRF models can only take into account the current and the previous label for the classification. In the WFST model, this would correspond to a 2-gram language model for the concepts, which performed 79.45% F1 score. WFST models are simpler to build and can thus use more complex language models that can outperform more complex generative methods.

5.2 Complex Models

The most advanced CRF models trained described in Section 4.4 achieved a F1 score of 83.57%. The model combines words, POS tags, stems, prefixes and radixes. This shows how discriminative models can exploit additional information that can not be easily modeled with generative models. Thanks to this additional information, CRF outperformed WFST on the concept tagging task.

algorithm	prec.	rec.	F1
WFST (4-gram)	82.44%	83.04%	82.74%
CRF (manual)	86.72%	80.66%	83.57%
CRF (genetic)	87.52%	81.03%	84.15%

Table 5: Comparison of the performances of the best models for WFST and CRF.

Table 5 compares the performances of best WFST and CRF models. We can notice that the WFST model has a significantly higher recall than both CRF models. WFST tends to be more conservative in the classification, probably due to the more complex language model (4-grams) than tends to prevent false positives. On the other hand, CRF models exploit multiple features and are able to find more concepts, at the cost of some extra false positive.

5.3 Training Complexity

WFST are very easy to train, since the training only requires to estimate some probability distributions from the training data. This can be done by simply compute frequencies and normalizing them to probabilities. Some effort should be put into the choice of the meta-parameters such as the window size for the n-gram language model and the smoothing method.

In contrast, CRF require to manually specify the features to use. As described in Section 4, this can be very difficult due to the exponential number of possible combinations. The problem can be partially solved using optimization techniques such as genetic algorithms. These methods automatically pick the best features combination, at the cost of some additional computations in the training phase. In our case, the genetic algorithm managed to improved significantly the model performances, as described in Section 4.6.

6 Conclusion

The most sophisticated CRF models outperformed the WFST models developed in the mid-term project, thanks to the discriminative nature that allowed to easily combine multiple features together. On the other side, CRF resulted much more difficult to train than WFST due to the exponential number of possible feature functions combinations. Some heuristics can be used to select the good features, then optimization methods can help to pick the best ones. In particular, our genetic algorithm managed to generate a population of CRF models which outperformed the best model trained by hand, reaching the F1 score of 84.15%.

References

Roberto Battiti and Mauro Brunato. 2014. *The LION way. Machine Learning* plus *Intelligent Optimization. Version 2.0*. LIONlab, University of Trento, Italy.

Charles Elkan. 2008. Log-linear models and conditional random fields. *Tutorial notes at CIKM* 8:1–12.

Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research* 13:2171–2175.