Language Understanding System final project

Marco Mancini
University of Trento
marco.mancini@studenti.unitn.it

Abstract

The following document reports all the work done for the Language Understanding System final project. It consists in the comparison between generative and discriminative model on the SLU domain. The report starts explaining what I've done in order to implement and test the models, finally a section regarding the comparison is presented.

1 Introduction

The objective of this final project for Language Understanding System (Giuseppe, 2016) is to compare generative and discriminative models on the Spoken Language Understanding domain. In particular, the model chosen for the generative family is the one analyzed in the first mid-term project, the Stochastic Finite State Transducers; regarding the discriminative models I've selected both Conditional Random Fields (CRF) and Recurrent Neural Networks (RNN). The case study on which the comparison has been developed is the same of the previous project, the movie domain. In the second section the CRFs are threaten, in particular I'll show which are the main templates of rules I've implemented for each feature-set I've decided to use. The third section regards in more details the RNNs and the test cases performed on all the feature-sets. The last section threat the comparison between all the models presented and the one already showed in the previous project, where is also possible to find the data analysis.

2 Conditional Random Fields

CRFs are discriminative models, which aim to model directly the posterior probability `P(Y|X)`. It is defined by a dependency graph G and sets of featuress $\mathrm{f_k}$, which are weighed by λ_k, these weights will be the ones to be learned during the training phase. The feature functions are defined a-priori, and with them it is possible to define any kind of relations between the features and the output labels. How it is possible to see from the formula below, the final posterior probability is defined over all the possible cliques of the graph, over all the features and it is normalized for a normalization factor `Z(x)`.

$$p(Y|X) = \frac{1}{Z(x)} \exp(\sum_{c}^{C}\sum_{k}^{K} \lambda_k f_k(y_c, x, c))$$

$$\mathrm{Z(x)} = \sum_y^Y \exp(\sum_c^C \sum_k^K \lambda_k f_k(y_c, x, c))$$

It is important to notice that the normalization factor is in common to every class and its computation is highly computational demanding. Moreover, the higher the number of feature functions, the more complex will be the computation of the posteriors. The interesting feature that differentiate the CRFs from the generative models is that with them is possible to define any kind of relations between the features and the output, this means that it is also possible to define cross relations, window feature functions able to also take into account features beyond the current one analyzed, and so on. Then, they result to be a-priori more expressive with respect to generative models. In order to craft the feature functions and use them into CRFs over the test-case datases, I've used CRF++ (Kudo, 2016).

2.1 Base features

I've developed these firsts tests on the basic dataset, composed only by words and output labels. CRF++ permits to craft templates specifying which columns to take into account (e.g. features), the width of the window to apply, the concatenations of features and the bi-gram or uni-gram model for the output label. I've decided

window	unigram/bigram	concatenations	word-window_Label	F1
[-2,0]	unigrams	No	No	62.25%
[-4,0]	unigrams	No	No	68.45%
[-2,0]	bigrams	No	No	76.73%
[-4,0]	bigrams	No	No	75.74%
[-2, +3]	bigrams	No	[0,0]	81.28%
[-2,+3]	bigrams	[0,1]	[-2,+1]	82.44%

Table 1: CRFs results on the base dataset

to start from simplest templates, which features are defined only over the previous 2 or 3 words and with unigrams on labels, until to arrive to the more complex ones defined over word-windows of bigger sizes (e.g. [-2, +3]), with bigrams on label and concatenations. The main results are showed in Table 1. The window column defines the window applied to the features (in this case only the words), the unigram/bigram column specifies which model has been applied to the output label, the concatenation is defined over the words and the word-window_Label column specify which word-window apply over the unigram/bigram model choosen. How it is possible to see, the usage of bigrams for output label generated a big gap of performances (8%), another improvement has been given from the usage of concatenations and word-windows on the labels.

2.2 Advanced features

Among all the possible combined features present in the feature set, I've selected the following four:

1. Lemma
2. Lemmas + Part of Speech tags.
3. Word + Part of Speech tags
4. Word + Part of Speech tags + Lemmas

Having these additional features allowed me to write template rules which can also work on them, using the same operations I've described before (window functions and concatenations). Like for the tests without additional features, I decided to proceed in an incremental way, starting from the more basics feature functions to the most complex ones.

The main results are showed in Table 2. The table is pretty similar to the previous one, it only differs on the added window and concatenations applied over the features, moreover the first column shows the identifier related to the selected features (it is related to the enumeration previously shown). It is interesting to notice that the best setting for the base features applied to the lemmas, produces a small drop of 0,15%. Moreover, using the whole set of features (word + pos + lemma) and playing a bit on the windows and concatenations, produced an F1-measure of 82.23%.

2.3 Enhanced dataset

The enhanced dataset was the one which gave the best results on the generative models. It was created by simply replacing the Os labels (no concept) with the word associated, the idea behind this operation was to exploit the more information as possible, since the big majority of the labels were the Os concepts. I've tried to run a CRF with very basic feature functions (e.g word-window [-2,0] and unigrams in labels), but CRF++ was completely stucked. This problem was probably due to the huge number of classes involved in this dataset, in fact there was a blow-up of the total number of features applied over all the classes (e.g 200000 total features on the base dataset vs 2000000 on the enhanced one), yielding to an highly demanding computation. For this reason I tried to reduce the number of classes generated, passing from all of them to ever smaller windows, until I reached a O-to-concept window of size 3 which resulted to be executable. Unfortunately it gave me worst results with respect to the previous cases (74% F1).

2.4 Other tests

Likewise for the generative models I've also tried to apply a frequency cut-off with different thresholds, but all the tests gave me worse results. Other tests have been done changing the `eta` parameter of the CRF model (e.g. regularization term), but its best value resulted to be the one set by default. Moreover I've also tried to use prefixes and suffixes as additional feature of the dataset. The selection of those suffixes and prefixes as been done in a greedy way, selecting the firsts and the lasts 3 character of the word. Then I've run different tests, the best one is the winner of Table 2, two rules added in order to take into account also of the single prefix and suffix of the current word. The described test gave me as result an F1 of 83.52%, which result to be the best one. I've also run a 10-cross validation over the CRF which gave me the best results and the F1-average between all the folds is 86.06%, confirming the overfitting impossibility.

3 Recurrent Neural Networks

In a traditional Neural Networks the inputs and outputs are independent of each other, then with a traditional structure it is not possible to exploit sequential information. In order to exploit those information there exist Recurrent Neural Networks in which, the weighted connections feeding a neuron, also come either from the hidden units of the previous iteration, or from the outputs; so, with these structures, it is possible to take into account sequences. The former I've just cited are also called Elman NN, the latter are called Jordan NN. Since both of them contains recursions, during the training the network is unrolled in time backwards and the backpropagation algorithm is applied in order to update the weights. Since the words cannot be passed to the network as they are, they are fed to the RNN using 1-of-n encoding, where the encoding is based on the position of the word in the vocabulary. Starting from these encodings a RNN is also able to learn representations of the words into a vector space (word embedding), where it is possible to find co-relations between them.

Regarding the Spoken Language Understanding domain, we can decide to apply an Elman RNN or a Jordan RNN, the main difference, as said, stands in the recursion. With an Elman RNN what we model as output is the posterior probability $P(Concept_t|W_t, H\_out_{t-1})$, where t is the time (i.e. the position of the word in the sentence) and H_out_{t-1} represents the output of the previous hidden layer (i.e. the hidden representation of the previous word). If we, instead, decide to use a Jordan RNN the posterior probability modelled is $P(Concept_t|W_t, Out_{t-1})$ which depends on the current word and the previous output (i.e. the label of the previous word).

In order to apply them to the movie-domain test case I've used the python project of (Bayer, 2016), given during the LUS course. Since the Neural Networks are subject to overtraining problems, which can lead to overfitting problem, what is used during the training in order to check the F1-progress is a validation set. In our dataset there was no present a validation set, for this reason I had to create it from the training set but, since the training set was already composed by few sentences (3300) I decided to only take 300 sentences from it (10%), so to have anyway a good point to start from for the comparisons. In the tool provided there is the possibility to play with the following hyperparameters:

- learning rate: the rate used in the gradient descent.
- context windows: number of words which will be used (concatenated) to determine the context of each word.
- batch-size: the number of utterances to use for each backpropagation step.
- hidden units: the number of hidden units in the hidden layer
- seed: the random seed used to shuffle the data
- word embeddings dimension: dimension of the word embedding space.
- number of epochs: max number of backpropagation steps

3.1 Base features

The following tests have been developed on the basic dataset. I've tested both Elman and Jordan RNN varying the parameters starting from the simplest ones, the best results are showed in Table 3. During the testing I've noticed that the context size plays the most important role for the performances (e.g. change it from 9 to 7 produce a drop of 3/4% in F1) and also the embedding size produces similar changes. Also in these tests I've tried to apply the frequency cut-off to the data but, like for the CRFs and the SFSTs, it only produced a drop of the performances (1/2% with threshold 1).

selected features	word window	Pos-Window	Lemma-window	unigram/bigram	word-concatenations	word-pos concatenations	word-window_Label	F1
1	[-2,+3]	No	No	bigrams	[0,1]	No	[-2,+1]	82.30%
2	[-2,+3]	[0]	No	bigrams	[0,1]	No	[-2,+2]	82.60%
3	[-2,+3]	[-1,0]	No	bigrams	[0,1]	[0]	[-2,1]	82.93%
4	[-2,+3]	[-1,0]	[0]	bigrams	[0,1] | [-1,+1]	[0]	[-2,1]	83,23%

Table 2: CRFs results on advanced features

type	learn_rate	context_win	batch_size	hidden_units	seed	embedding	epochs	F1
Elman	0.1	9	5	100	3842845	100	25	74.45%
Jordan	0.1	9	5	100	3842845	100	25	77.94%

Table 3: RNN best results on basic feature set

3.2 Advanced features

The datasets used for these tests are the same used with the CRFs. In order to use the additional features with the RNNs I've simply concatenated them (like for the SFSTs). In this case I started from the settings which gave me best results in the previous tests, and they give me the best results also here. In fact I've tried to play a bit on the parameters, but all the tried combinations resulted in a drop of performances. In Table 4 it is possible to see the best result for each feature set, the best one is given by the word-pos concatenation with the Elman RNNs. It is interesting to notice how the usage of all the features concatenated (i.e. word + PoS + lemma) have produced a big drop of performances, probably due to a sparseness problem.

3.3 Enhanced dataset

These last tests have been deployed on the enhanced dataset. Unlike for the CRFs, I've been able to test the RNNs over the complete enhanced dataset, and the results can be seen in Table 5. The Jordan RNN performed better on this dataset, this is probably due to the fact that the enhanced dataset is meaningfull in the output labels, which are exploited by the Jordan RNNs. Also in this case I've tried to change some hyperparameter and play with the cut-off thresholds, but the results only worsened.

4 Comparison

The comparison will be performed over the main performance measures, which are F1, accuracy, precision and recall. I've also decided to compare the training time needed for the models, since they differ significantly one to each other.

4.1 Base features

In this first subsection I'm going to compare the best results reached by each model on the base feature dataset. How it is possible to see from Table 6 the CRFs have been the ones to give the best F1 measures on this set. Moreover, if we look at the training time needed, they take only 1/3 of the time needed to train a RNN.

4.2 Advanced Features

The following results show the comparison between the discriminative and generative models over the advanced feature set. I've selected only the best result for each model and, how it is possible to see from Table 7, the CRFs have the best ones, again. Moreover, the 82.23% of F1 measure reached results to be the best between all the results.

4.3 Enhanced dataset

As previously explained, was not possible to run the CRFs over the full enhanced dataset, for this reason in Table 8 there are only the comparisons between SFSTs and RNN. In this case the generative model won, but the results reached do not beat anyway the best ones reached with the CRFs. It is interesting to notice how the Jordan RNN, in this case, reached the best accuracy of the whole set of tests; however, in this case, its training lasted in 40/45 minutes, this was probably due to the increased number of classes to predict.

5 Conclusions

I've compared a generative model (SFSTs) and two discriminative models (CRFs and RNNs) in the context of Spoken Language Understanding. The comparison has been done mainly on the F1-measures reported by the tests, but I've also looked for accuracy and the time to train the models. I can state that the best F1 measure have been reached

type	dataset	learn_rate	context_win	batch_size	hidden_units	seed	embedding	epochs	F1
Jordan	1	0.1	9	5	100	3842845	100	25	76.50%
Jordan	2	0.1	9	5	100	3842845	100	25	77.84%
Elman	3	0.1	9	5	100	3842845	100	25	78.44%
Elman	4	0.1	9	5	100	3842845	100	25	75.07%

Table 4: RNN results on the advanced dataset

type	learn_rate	context_win	batch_size	hidden_units	seed	embedding	epochs	F1
Jordan	0.1	9	5	100	3842845	100	25	77.15%
Elman	0.1	9	5	100	3842845	100	25	76.10%

Table 5: RNN results on the enhanced dataset

Model	Accuracy	Recall	Precision	F1	Training time
SFST	92.68%	74.34%	78.51%	76.37%	∼2/3 mins
CRF	94.20%	88.20%	78.18%	82.44%	∼10 mins
RNN (Jordan)	94.68%	81.92%	74.34%	77.94%	∼30 mins

Table 6: Comparison on base set

Model	Dataset	Accuracy	Recall	Precision	F1	Training time
SFST	Lemma + PoS	92.19%	72.23%	77.56%	74.80%	∼2/3 mins
CRF	Word + PoS + Lemma	94.65%	80.53%	87.90%	83.52%	∼10 mins
RNN(Elman)	Word + PoS	95.30%	76.72%	80.25%	78.44%	∼30 mins

Table 7: Comparison on advanced set

Model	Accuracy	Recall	Precision	F1	Training time
SFST	94.96%	83.04%	82.44%	82.74%	∼2/3 mins
RNN(Jordan)	95.32%	80.29%	74.24%	77.15%	∼40/45 mins

Table 8: Comparison on enhanced dataset

by the CRFs on the word+PoS+Lemma feature set (83.52%), using additional features like prefixes and suffixes; this result is close to the SF-STs results on the enhanced dataset (82.74%). Instead the model with the best accuracy is the Jordan RNN (95.32%). Regarding the training time I can confirm what expected, in fact the generative models are really faster to train, instead the generative ones are slower, especially the RNNs because of their more complex structure and also because of the backpropagation algorithm. In all the 3 tested models I can confirm that both the frequency cut-off and lemmatization did not improve the results, probably this is due to the fact that the corpus was of small-medium size. Personally, I've found the CRFs the more flexible models and the ones in which is easy to embed complex and specific rules. How reported by other papers (GrÃľ-goire Mesnil, 2013) (Christian Raymond, 2007), I think that with a more extended corpus (5000 utterances at least) the discriminative models can further outperform the generative ones.

Appendices

The project and some of the models used can be found on GitHub at `https://github.com/marcomanciniunitn/Final-LUS-project`

References

ALi Orkan Bayer. 2016. Recurrent neural network python tool.

Giuseppe Riccardi Christian Raymond. 2007. Generative and discriminative algorithms for spoken language understanding.

Riccardi Giuseppe. 2016. Language understanding system course [unitn]. `http://disi.unitn.it/~riccardi/page7/page13/page13.html`.

Li Deng Yoshua Bengio GrÃľgoire Mesnil, Xiaodong He. 2013. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding.

Taku Kudo. 2016. Crf++. `https://taku910.github.io/crfpp/`.