Project EIDI Report: Using artificial data generated from pattern plus lexicons for Natural Language Understanding task in the dialogue system

Taycir Yahmed [1] **Mo Yang** [2] **Zizhao LI** [3]

Abstract

Dialogue system is widely used in lots of situations including customer service, AI assistant, connected object, chatbot...... For task oriented system, A central challenge is how to let system to understand the semantic meaning of a sentence. A classical solution is to use Named Entity Recognition (NER), where the model is usually trained by supervised learning.

The problem is that labeled data can be expensive especially when facing to complex and varies dialogue situations. So an alternative solution is to use patterns plus lexicons to generate artificial data. The purpose of this project is to evaluate the impacts using artificial data for Named Entity Recognition task.

1. Introduction to the subject

Dialogue System is a hot research area for artificial intelligence product development. In general Dialogue System can be divided into two categories(1): conversation-oriented systems which aim to converse with users and provide interesting and reasonable contextually relevant responses, and task-oriented systems designed to assist users achieve specific goals (e.g. find products, restaurants or flights) (2) Our project is based on building task-oriented system.

A central problem for task-oriented dialogue system is to develop Natural Language Understanding (NLU), which is realized by concept detection. The idea is to extract the domain concepts from sentence. It is a typical Named Entity Recognition (NER) task in Natural Language Processing. Previous data-driven works for NLU task is mainly based on human-human dialogue corpus.(3) While, the disadvantages are obvious. First, it is very hard to collect large scale human-human dialogue data in every real-world situation. Second, it is very expensive to label the huge mount of data even if they are collected. Some data augmentation methods are raised to deal with such problem (4). In this project, we focus on evaluating the effectiveness of using artificial data generated from pattern plus lexicons for NER task so eventually NLU task.

[1]Telecom ParisTech, France. [2]ParisSaclay University, France [3]ParisSaclay University, France. Correspondence to: Taycir Yahmed <tyahmed@enst.fr>, Mo Yang <mo.yang@u-psud.fr>, Zizhao Li <zizhao.li@u-psud.fr>.

Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

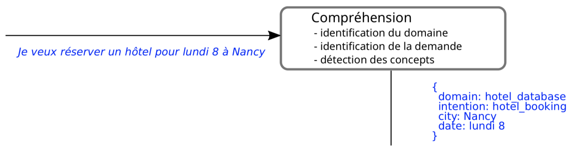


Figure 1. Concepts detection

2. Dataset description

The data is in the scenario of cooking. The understanding system is all about searching for recipes with or without ingredients or a specific family of ingredients. All the texts are generated based on question patterns plus lexicons which are crawled from world wide web (eg. wikipedia and marmiton.org) . In another word, all the texts are artificial data.

The data used in the projects is in three parts: Train set (trn), Development set (dev), Test set (test). Sentences in Train set are generated from pattern plus lexicons trn. In the Development set, sentences are generated by new patterns or new lexicons or both. The Test set follows the same principles but add some entirely new patterns. To be more detailed, here is the data set components:

For trn, dev, test, there are three categories of sub-corpus:

- ask_for_recipe
- give_cat-ingredients
- give_ingredients

The number of the three categories are hugely unbalanced.

For the dev, to effectively analyze the impacts of training from artificial data on different distribution of dev data, contrasts are supposed to be made by *new pattern* vs *old pattern*, *new lexicons* vs *old lexicons*. The following terms indicate components in different dev files:

- newP = new patterns
- oldP = patterns which has appeared in trn
- newV = new lexicons
- oldV = lexicons which has appeared in trn

In the project we aim to train a NER model and carefully choose appropriate number of examples on trn, tune the model on dev, and eventually evaluate and analyze on test.

3. Analytical tools

To analyze the impacts on results by using traditional statistical model or neural network, we also take two approaches to do the analysis.

3.1. Wapiti (CRF model)

Conditional random fields (CRFs) (5) are a kind of sequence statistic modeling method often used for structured prediction. CRFs are a type of discriminative undirected probabilistic graphical model which is used to encode known relationships between observations and construct consistent interpretations and is often used for labeling or parsing of sequential data, in which the linear chain CRF that is usually used in natural language processing in particular in NER predicts sequences of labels for sequences of input samples.

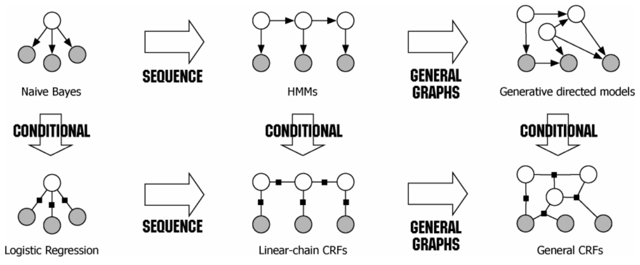


Figure 2. Diagram of the relationship between naive Bayes, logistic regression, HMMs, linearchain CRFs, generative models, and general CRFs. (6)

CRFs have been implemented in wapiti with different method of optimization such as l-bfgs which is the classical quasi-newton optimization algorithm with limited memory, sgd-l1 which is the stochastic gradient descent for L1-regularized model, bcd, the blockwise coordinate descent with elastic-net penalty and others.

In our case, we choose l-bfgs as the optimization algorithm that generally produces better models than SGD but requires a lot more memory for training.

3.2. LSTM + CRF + word embeddings

Recurrent neural networks (RNNs) are a family of neural networks that operate on sequential data. They take a sequence of vectors (x1, x2, . . . , xn) as input, in the mean time, generate another sequence (h1 , h2 , . . . , hn), named hidden layer, which represents some information about the sequence at every step in the input. The hidden layer could be single layer or multiple layers depends on task complexity. Eventually output layer is built on top of hidden layer and output sequence expected. In theory RNNs can learn long dependencies, but RNNs tend to be biased on the most recent inputs because of the phenomenon *gradient vanishing* (7). Long Short-term Memory Networks (LSTMs) aim to solve the problem by using *memory cell* and have been shown able to remain long-dependency information. As the second approach of the project we tend to try LSTMs on NER task.

For this part we have used an open source program *Named Entity Recognition with Tensorflow* (8). The program implement a NER model: a bidirectional LSTM with a sequential conditional random layer above it (LSTM+CRF) (9). The main architecture is as the figure below. In a few words bi-directional LSTM serves as feature extractor and CRF is used as tagging model and is built on top of LSTM. CRF model takes LSTM hidden layers units values as features and output the tagging decisions in each step, which eventually generate a sequence of tagging results.

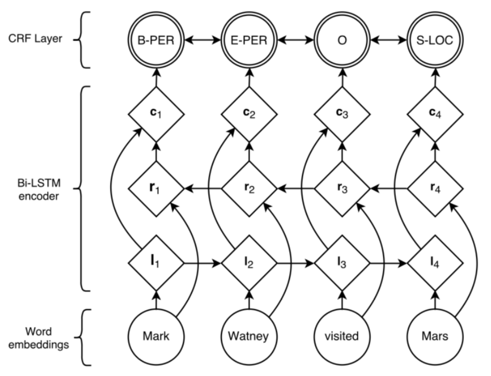


Figure 3. Main architecture of the network.Word embeddings are given to a bidirectional LSTM. l_i represents the word i and its left context, r_i represents the word i and its right context. Concatenating these two vectors yields a representation of the word i in its context, c_i.

4. Discussing the results

In this section, we will discuss the results we obtained during our experiments with two kind of methods: CRF and LSTM + CRF.

4.1. Results using Wapiti

Firstly, we want to see the influence of distribution of examples in train set to our model, we train one model by using all of the data in train set which contains 76 examples give_cat-ingredients, 487542 examples ask_for_recipe, 2284626 examples give_ingredients and then test it on development set. Then, we try to fix this unbalance on dataset, we firstly choose a subset of similar examples number of give_cat-ingredients from ask_for_recipe and give_ingredients, in other words a very small dataset using for training. Afterwords, we try to extend give_cat-ingredients and ask_for_recipe to approach the volume of give_ingredients via copying and shufflering examples. As for the pattern file used in CFG, we take into account the five words around the current token as unigrams and the two words around the token as bigrams to label this token in our basic pattern file.

For testing on development set & test set, we separately test the model on different dataset with different kinds of patterns or lexicons, we also mix all the data and test the model on it. The evaluation on development set is based on the built-in f1 score and ne-scoring-gen on test set. The results are showed in Table 1 - 3.

From Table 1, we can see that Pattern is a more important factor than Lexicon in our case. The performance of dataset with patterns appeared in train set (old patterns) is much better than those with new patterns. The occurrence of lexicons doesn't matter comparing the result of newPoldV to the result of newPnewV. However, the unbalance of train set have big influence on such kind of statistical model where we notice that the f1 score of *cat-ingredient* is always NAN when we predict with normal model. This situation has improved when we balanced the dataset using subset data, while the global result is not very well cause we took a small dataset to train. So another idea comes up, we extend the trainset by copying examples to build a balanced model with 7006896 examples in total. From Table 2, we observe that the global error has decreased, meanwhile some *cat-ingredient* labels have been detected.

We also tried other methods like finding the stem of named entity, deleting some repeated named entity in *give_ingredients*, however, as we said in last paragraph, the lexicon doesn't play an important role in this case, so here we didn't present the result of methods related to string processing.

We have done some experiments trying to add the PoS tag of token in .bio file, because of the previous experiences using Wapiti, we find that add PoS tag in pattern file could help us to get a better model. We have passed two days to tagger the middle size file *ask_for_recipe-trn.bio* using our laptop, however, the program didn't finish yet when we stopped it. So due to the limited computation capacity of laptops, we finally gave up to add the PoS tag in .bio file and modify basic pattern file.

4.2. Results using LSTM + CRF

The model used is concretely end to end. Training a neural network is a very tedious work and will take a lot of time. We think that fine tuning the model is not the most important thing in this project, so we basically follow the default hyper parameters (optimization function, dropout.....). The only variable each time we train a model is the number of examples. There are two reasons for this: First we want to figure out what impacts on dev set scores when using different scale of train set. Second we want to figure out what impacts on different kind of dev sets when using different distribution of each kind in train set.

We have trained three models. The first model is trained by small scale of training examples (100 ask_for_recipe, 78 givecat-ingredients, 100 give_ingredients, total number of examples: 278). The second model is trained by a bit more training examples (500 ask_for_recipe, 78 givecat-ingredients, 500 give_ingredients, total number of examples: 1078). The third model is trained by relatively large scale training examples ((10000 ask_for_recipe, 78 givecat-ingredients, 10000 give_ingredients, total number of examples: 20078))

For each model, we have separately tuned the model and evaluate the result on different dev and test set, which are:

- dev and test set of type newPnewV
- dev and test set of type newPoldV
- dev and test set of type oldPnewV
- dev and test set of give_cat-ingredients
- whole dev and test set

We use built-in f1 score to evaluate the results. The results are as below:

It is not surprise to find out that the approach neural network does not give a good result in any cases. While, there are several causes. First, as we know in advance, there is domain transfer between trn, dev, test. They are from very different distributions. Second, Neural network has huge amount of parameters to train, so normally it demands for large-scale train data. Because we do not have powerful

Table 1. Experimental results for normal models of original data and short model based on data subsets(76 give_cat-ingredients examples, 100 ask_for_recipe, 100 give_ingredients) tested on dev set: oldPnewV, newPoldV, newPnewV and a mixture of them. The evaluation is based on the built-in f-measure.

LABELS	OLDPNEWV(NORMAL)	NEWPOLDV(NORMAL)	NEWPNEWV(NORMAL)	OLDPNEWV(SHORT)
TOKEN ERROR	1.08%	13.84%	12.97%	2.81%
SEQUENCE ERROR	8.28%	72.05%	70.36%	19.97%
O	1.00	0.96	0.96	0.99
B-RECIPE	0.99	0.61	0.61	0.90
I-RECIPE	0.98	0.91	0.92	0.95
B-INGREDIENT	0.99	NAN	NAN	1.00
I-INGREDIENT	0.88	NAN	NAN	0.76
B-NEG_INGREDIENT	0.99	0.57	0.55	1.00
I-NEG_INGREDIENT	0.95	0.03	NAN	0.92
B-CAT-INGREDIENT	NAN	NAN	NAN	1.00
I-CAT-INGREDIENT	NAN	NAN	NAN	NAN
B-NEG_CAT-INGREDIENT	NAN	NAN	NAN	0.94
I-NEG_CAT-INGREDIENT	NAN	NAN	NAN	NAN

Table 2. Experimental results for balanced models based on copied dataset (2284560 give_cat-ingredients examples, 2437710 ask_for_recipe, 2284626 give_ingredients) tested on dev set: oldPnewV, newPoldV, newPnewV and a mixture of them. The evaluation is based on the built-in f-measure.

LABELS	OLDPNEWV	NEWPOLDV	NEWPNEWV	MIXTURE
TOKEN ERROR	0.99%	10.54%	9.72%	6.71%
SEQUENCE ERROR	6.98%	47.82%	43.05%	32.57%
O	1.00	0.97	0.97	0.98
B-RECIPE	0.99	0.62	0.61	0.74
I-RECIPE	0.98	0.94	0.93	0.95
B-INGREDIENT	0.99	NAN	NAN	0.99
I-INGREDIENT	0.86	NAN	NAN	0.57
B-NEG_INGREDIENT	0.99	0.78	0.83	0.87
I-NEG_INGREDIENT	0.97	0.05	NAN	0.43
B-CAT-INGREDIENT	0.22	NAN	NAN	0.22
I-CAT-INGREDIENT	NAN	NAN	NAN	NAN
B-NEG_CAT-INGREDIENT	0.77	0.37	0.13	0.87
I-NEG_CAT-INGREDIENT	NAN	0.67	NAN	0.67

GPU we can hardly use more training data than 20000 examples. Third, nerual network is easy to over-fitting, sometime it overfit within one epoch. The model need to be further tuned. But in all cases we can find out there are several places worth to notice:

First, model trained with large dataset has almost opposite result compared to model trained with relatively small dataset. Model trained with large dataset has a much better score on oldPnewV than model trained with raletively small dataset. And in contrast, model trained with large dataset does much worse on newPnewV and newPoldV than those trained with small dataset. It seems that LSTM can learn better patterns when there are more training data, and does not recognize the patterns that it has never seen.

There is one interesting thing, in most cases model does worse in test than dev, which is obvious because dev set is still artificial data, while test set obviously have more variance. But model trained with large data set have much higher score on test than dev (almost zero). This might indicate that LSTM may have surprising performance on dealing with real-word high variant data, if it learns more patterns.

The results of all these models on 'oldP' dev and 'oldP' test are very close.

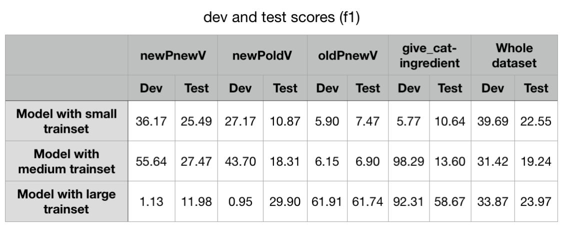
What's more, LSTM seems to be not sensitive to the unbalanced distribution in the training set. This might be explained that neural network is good at dealing with some sparse features, because it has much more parameters or intuitively, larger memory.

From the results we can find that 'Patterns' and 'number

Table 3. Experimental results for balanced models tested on test set: oldPnewV, newPoldV, newPnewV, a mixture of them and a more complicated case. The evaluation is based on *ne-scoring-gen*.

ERROR	OLDPNEWV	NEWPOLDV	NEWPNEWV	MIXTURE	REAL CASE
SLOT ERROR RATE	6.8%	100.5%	119.8%	74.0%	160.7%
CORRECTS	94.8%	31.0%	55.1%	57.9%	7.0%
INSERTS	0.0%	61.4%	94.4%	50.2%	91.8%
DELETES	5.2%	0.0%	0.0%	0.0%	38.5%
SUBSTITUTIONS	9.4%	69.0%	44.9%	42.1%	54.5%
F-MEASURE	92.8%	23.7%	37.4%	46.3%	5.5%

of training examples' are the key-points to our task. If we want to augment the dev or test score using LSTM, we must first have large enough training set. We also must try to increase the percentage of 'oldP', in another word, add rich patterns in the training set. If possible, balance the training data distribution.


dev and test scores (f1)

	newPnewV		newPoldV		oldPnewV		give_cat-ingredient		Whole dataset	
	Dev	Test	Dev	Test	Dev	Test	Dev	Test	Dev	Test
Model with small trainset	36.17	25.49	27.17	10.87	5.90	7.47	5.77	10.64	39.69	22.55
Model with medium trainset	55.64	27.47	43.70	18.31	6.15	6.90	98.29	13.60	31.42	19.24
Model with large trainset	1.13	11.98	0.95	29.90	61.91	61.74	92.31	58.67	33.87	23.97

Figure 4. Results of LSTM + CRF

5. Work contribution

Taycir Tahmed focuses on the first project: entity normalization. The second project of NLU task in dialogue system are realized by Mo Yang et Zizhao Li. Zizhao Li takes care of the wapiti part and Mo Yang uses the LSTM+CRF model.

6. Conclusion

In this project we have focused on analysing and discussing the impacts of using artificial data to NER task. We have analyzed from different perspectives, including different models, different number of training examples, different distribution of training examples, the impacts on different types of dev or test set......

We have effectively implemented two approaches, wapiti(CRF model) and LSTM+CRF. It is not easy to compare between the two approaches since LSTM takes a long time to train and we can not use more training examples and further fine tune. But both the results of the two approaches reveal the same fact that the most effective amelioration is to rich the patterns as much as possible in the training set. Thus potential improvement is to effectively get rich patterns to generate artificial data. One way is to find a way to extract patterns from real-world conversations.

Besides, LSTM seem to have some good properties like its impressive ability to learn patterns and ability to deal with sparse features. So in the future we might continue to ameliorate this method, get state-of-art performance.

Reference

[1] Su, P. H.; Gasic, M.; Mrki, N.; Barahona, L. M. R.; Ultes, S.; Vandyke, D.; Wen, T. H.; and Young, S. 2016. On-line active reward learning for policy optimisation in spoken dialogue systems. In Meeting of the Association for Computational Linguistics (ACL).

[2] Henderson, M.; Thomson, B.; and Williams, J. 2014. The second dialog state tracking challenge. In 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue, 263.

[3] Chen, Y.-N.; Wang, W. Y.; Rudnicky, A.; et al. 2013. Unsupervised induction and filling of semantic slots for spoken dialogue systems using frame-semantic parsing. In Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on, 120125.

[4] Itsumi Saito, Jun Suzuki, Kyosuke Nishida, Kugatsu Sadamitsu, Satoshi Kobashikawa, Ryo Masumura, Yuji Matsumoto, Junji Tomita . 2017. Improving Neural Text Normalization with Data Augmentation at

Character- and Morphological Levels, Proceedings of the Eighth International Joint Conference on Natural Language Processing, Volume2: short papers.

[5] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[6] Charles Sutton, Andrew McCallum, et al. An introduction to conditional random fields. *Foundations and Trends® in Machine Learning*, 4(4):267–373, 2012.

[7] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. Neural Networks, IEEE Transactions on, 5(2):157166.

[8] Opensource program Named Entity Recognition with Tensorflow . https://github.com/guillaumegenthial/sequence_tagging.

[9] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, Chris Dyer. 2016. Neural Architectures for Named Entity Recognition, Proceeding of NAACL 2016.