Natural Language Processing - Project

Query understanding for virtual assistant

François Devrainne - Anne-Camille de Sainte Foy

This project falls under the scope of text classification and aims at both identifying query types (intents) and performing slot filling. We used about 13K queries and each query falls under one of 7 different query types: "SearchCreativeWork" (e.g. "Find a movie called Living in America"), "GetWeather" (e.g. "What is the Sri Lanka forecast for snow?"), "BookRestaurant" (e.g. "Book a restaurant for one person at three pm"), "PlayMusic" (e.g. "Play some soul music"), "AddToPlaylist" (e.g. "Add song to sleepytime"), "Ratebook" (e.g. "Give this essay a 2 out of 6"), "SearchScreeningEvent" (e.g. "Find some close by movies"). This data comes from Google's API.ai, Facebook's Wit, Microsoft's Luis, Amazon's Alexa, and Snips' NLU and were collected by Snips[1].
The objective is to build a model for each intent that is able to extract the slot from a corresponding query. For instance, for the query "Add song to sleepytime", the goal is to extract the two following slots: "music item: song" and "playlist: sleepytime".

1 Models

We proceed in two steps: a first algorithm will identify the query type and a second one will perform slot filling. For instance, this two-steps process could be a way to model the functioning of an AI virtual assistant that receives a query and would first determine the type of query and then extract the relevant attributes to provide the user with the correct answer. For the first step, we used a Long Short-Term Memomry (LSTM) neural network and Conditional Random Fiels (CRFs) for the second. We also compare our results with an algorithm, which directly performs slot filling on a merged dataset that includes all the queries. We optimise the LSTM and CRFs independently and assume that tuning the hyperparameters of both algorithms independently will give the optimal hyperparameters of the algorithm computed on the merged dataset.

Identify the type of query - LSTM neural network

In a first step, we want to identify the type of query. To do so, we first tokenised our queries, that is we converted our queries into tensors. Then, we computed a LSTM neural network. LSTM has feedback connections and enables one to explore all dependencies between the words of a query. It also avoids the vanishing and exploding gradient problem that are common problems faced by recurrent neural networks. We used Adam Gradient Descent with a learning rate of 0.0005, a batch size of 16, cross entropy as loss function and accuracy as metric. The network is trained on 10 epochs. We also added dropout to avoid overfitting.

Slot filling - CRFs

In order to predict the slot corresponding to each word in a sentence, we used a CRF model. In this model, each observation is a word in a sentence, to which we assign different features. We chose to include the position of word i in the sentence, the $i-1$ and $i+1$ words before and after the word, the number of upper cases in order to distinguish proper and common names and the length of the query. The model uses stochastic gradient descent and is trained on 200 iterations.

[1]The datasets are available here:`https://github.com/snipsco/nlu-benchmark/tree/master/2017-06-custom-intent-engines`

2 Results

Identify the type of query

After having trained the LSTM model on the train set (12,260 queries), we tested the performances of our algorithm on the test set (1,363 queries). We plotted the loss and validation loss and accuracy and validation accuracy on the train and test set both as a measure of performance and to detect potential overfitting (Cf. Appendix 1). Throughout the different epochs, the loss and validation loss decrease and the accuracy and validation accuracy incrase, which shows that our algorithm learns well. At the end of the 10^{th} epoch, our LSTM model achieves very good performances even on the test set, which is very important in order to check that the algorithm generalises well.
We also computed the confusion matrix (Cf. Appendix 2). It confirms that most observations are well predicted. The best predicted types of queries are "RateBook" and "BookRestaurant" and the worst are "SearchCreativeWork" and "SearchScreeningEvent" (resp. 100% and 96% of well predicted queries). This sounds quite logical because "SearchCreativeWork" and "SearchCreativeEvent" are the most diverse types of queries and includes requests for respectively different songs, plays, shows, journals etc. and different types of events.

Slot filling

We tested the algorithm both on specific queries and on all queries at the same time. Whatever process we choose, we get an average precision, recall and F-1 score of 95%, which shows that our algorithm performs well. When running on specific queries, the algorithm performs best on the "RateBook" queries. This is probably linked to the fact that slots for this query are quite specific (e.g. rate, rating, book...). We also computed the number of queries for which our algorithm is wrong (Cf. Appendix 3). Overall, for the vast majority of queries (2874 queries), there is no error.
We also considered combining the two algorithms, that is build a single algorithm that identifies the type of query in a first step and returns the relevant slots in a second step. It appears that this algorithm performs better: we get an average precision, recall and F1-score of 99%. This model therefore seems to be an adequate solution for our problem, especially if we want to scale the number of different types of queries.

3 Conclusion

This project aimed at identifying the type of a particular query on the one hand and extract the relevant slot from a query on the other hand. As shown by the results, LSTM and CRFs are models that perform well to solve these kinds of problems. To go further, it would have been possible to include parsing in the features of our algorithms. Yet, this method seems quite painful to implement regarding the great diversity of types of expressions we have: for instance, a song title that includes a verb could mislead the algorithm.

Appendices

Appendix 1 - Loss and accuracy

Loss	Validation loss	Accuracy	Validation accuracy
4.3%	6.0%	99.0%	98.4%

Table 1: Loss and accuracy on the training and test sets

Appendix 2 - Confusion matrix

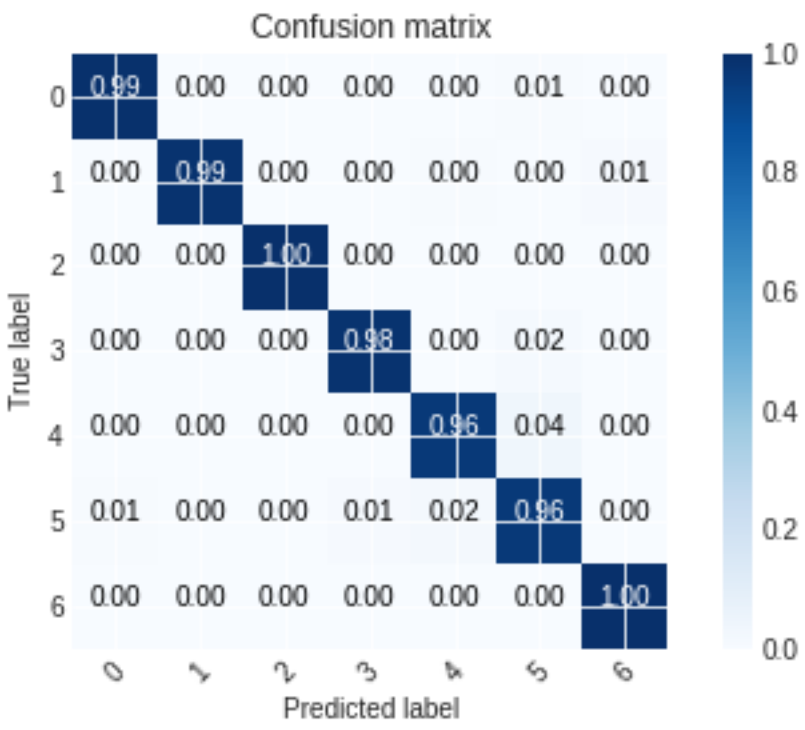

Figure 1: Confusion matrix

Appendix 3 - Number of errors for CRF model

Number of errors	Number of queries
0	2874
1	276
2	180
3	109
4	70
5	22
6	20
7	7
8	9
9	5
10	1
11	1

Table 2: Number of errors for CRF model