ZPJa: Joint Intent Classification and Slot Filling

Skuratovich Aliaksandr
xskura01@fit.vutbr.cz

Santosh Kesiraju
kesiraju@fit.vut.cz

Abstract

In this report, I evaluated the performance of these three models on a dataset of human-robot interactions (NLU Evaluation Dataset) and compare their results in terms of accuracy and efficiency. My findings provide insights into the strengths and weaknesses of each approach. BERT-based models were trained for joint intent classification and slot filling using two types of BERT - large and base. Large BERT provided better results. I also trained Fasttext CBOW and Stanford NER models for comparison with BERT-based models. The best results were obtained with the BERT-based models, but the Stanford NER model was significantly smaller, and the accuracy decrement was not so significant. Fasttext also provided good results, but not as good as the BERT-based models, while the difference in model sizes was not as significant as between Stanford NER and BERT-based models. I also identified errors in the NLU Evaluation Dataset and errors in the data preparation pipeline of the framework being used and plan to contribute to the framework by fixing these issues.

1 Introduction

Intent classification and slot filling are important tasks in dialogue systems because they enable the system to accurately interpret and respond to user requests.

Intent classification involves identifying the intention or purpose behind a natural language statement or request. This is important because it allows the system to understand what the user wants or is trying to accomplish and to provide an appropriate response. For example, if a user asks "`what is the weather today in Brno`", the intent classification component of the dialogue system would identify the request as a "`weather forecast`" and send it to the appropriate module for handling.

Slot filling involves extracting specific pieces of information (called slots) from a natural language statement or request. This is important because it allows the system to gather the necessary information to fulfill the user's request. For example, in our example, the slots might include a specific location (`Brno`) and time (`today`). Without slot filling, the dialogue system might not have all of the necessary information about the request.

Together, intent classification and slot filling are essential for enabling dialogue systems to understand and respond to user requests effectively. They are integral to many natural language processing systems, including voice assistants, chatbots, and virtual assistants.

2 Task Definition

2.1 Intent classification

Intent classification can be defined as a supervised machine learning task where the goal is to learn a function $f : X \rightarrow Y$, where X is an input sequence and Y is a predefined category.

In the context of natural language processing, the input instances (X) may be natural language statements or requests, and the labels (Y) may represent different intentions or purposes.

This is typically done by training a machine learning model on a labeled dataset of X and Y pairs and then using the trained model to make predictions on new, unseen input instances.

Intent classification is used to be done using machine learning algorithms such as support vector machines, decision trees, and also neural networks. However, recently it becomes more popular to employ large pre-trained transformers such as BERT (Devlin et al., 2018) to compute embeddings of input utterances. The performance of the intent classification model is typically evaluated using metrics such as accuracy, precision, and recall.

2.2 Slot filling

Slot filling involves accurately identifying specific pieces of information, referred to as "slots," within

a given input sequence. This task can be represented as a function $f : X \rightarrow Y$, where X is a vector representing the input sequence and Y is a vector where each element corresponds to the mapping of a specific element in the input sequence to a target slot.

3 Method

For the task of joint intent classification and slot filling a BERT-based model was used.

BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based language model that can be fine-tuned for a variety of natural language processing tasks.

At a high level, BERT works by encoding an input sequence $X = [x_1, x_2, ..., x_n]$ into a set of continuous-valued vectors, or embeddings, $E = [e_1, e_2, ..., e_n]$. The embeddings capture the meaning and context of the words in the input sequence and are used as input to one or more downstream tasks, such as classification or sequence labeling.

To generate the embeddings, BERT first applies a word embedding layer to map each word in the input sequence to a fixed-length vector representation. It then applies a series of self-attention layers to compute the relationships between the words in the input sequence. Finally, it applies a feed-forward neural network to generate the final embeddings.

The training of a joint intent classification and slot-filling model was done an such a way, that firstly embeddings were computed using BERT model. Then these embeddings were passed into projection layers to classify an intent an slots. Mathematically speaking:

$$E = \text{BERT}(X)$$
$$p(Y|X) = \text{MLP}_i(E_{CLS})$$
$$p(Z|X) = \text{MLP}_s(E)$$

, where $p(Y|X)$ is the posterior probability distribution over intent labels, and $p(Z|X)$ is the posterior probability distribution over slot labels for every input token, MLP_i and MLP_s are feed-forward networks with two layers and softmax activation function for intent prediction and slot labeling respectively.

Total loss $\mathcal{L}$ is computed in the following way:

$$\mathcal{L} = \lambda \mathcal{L}_i + (1 - \lambda)\mathcal{L}_s$$

, where $\mathcal{L}_i$ is the cross-entropy loss for intent classification, $\mathcal{L}_s$ is the loss for slot labelling, and λ is the weight of intent classification loss.

4 Experimental Setup

There is a pipeline describing how models have been trained.

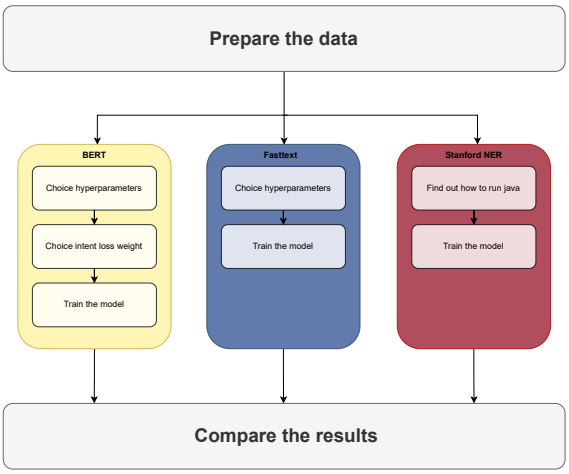

Figure 1: The pipeline for model training and comparison

In order to adequately evaluate the performance of each model, the initial data set was partitioned into ten distinct subsets for the purpose of cross-validation. Subsequent to this preprocessing step, the data was transformed into a format compatible with the NVIDIA NeMo, Fasttext, and Stanford NER frameworks. The subsequent phase of the experimental procedure involved the training of each model and the optimization of hyperparameters specific to each model. Finally, the resulting scores obtained from each model were compared and analyzed to determine their relative efficacy.

4.1 Dataset

NLU-Evaluation-Data (Xingkun Liu and Rieser, 2019) is a data set for human-robot interaction in the home domain. This dataset was used for the training. For more accurate results it has been chosen to use 10-Fold cross-validation. Each fold contains 64 intent values and 54 slot types. The train set contains approximately ten thousand utterances and a thousand test utterances.

4.2 BERT-based models

4.2.1 The first step - choosing hyperparameters and intent loss weight

NVIDIA NeMo framework was used (Kuchaiev et al., 2019) for experiments. It does not support

hyperparameter optimization techniques such as WandB sweeps[1] out of the box.

As the first step, several models were trained with the fixed intent loss weight of 0.4, and different hyperparameters to choose the best.

The best parameters were:

- Learning rate: $1e-6$. This parameter is one of the most important parameters. The default value of the learning rate provided in NeMo recipe was quite high. I compared several different values of the learning rate, and the best results were obtained with the learning rate of $1e-6$.
- Batch size: 128. Smaller batch sizes bring more noise to the training process, yet the model becomes more robust to overfitting. The optimal batch size for a dataset of this size (10k samples) is the smallest possible, as said in Goodfellow's book (**?**). However, it would increase training time by a lot.
- Optimizer: Adam. I only experimented with the Adam optimizer with default parameters.
- Weight decay: 0.003.
- Dropout: 0.8. Surprisingly enough, this value does not affect the accuracy of predictions nor prevent overfitting.
- Epochs: 300

Then, to choose Intent Loss Weight parameter models were trained on the first fold out of ten folds because it is only an internal comparison. In case of statistically insignificant differences between models with different weights of intent loss, it does not matter which model will be picked up for further experiments. On average, models trained with intent loss weight of 0.4 gave the best precision, as seen in Table 1.

4.3 Baseline models

4.4 Joint intent classification and slot filling

It has been decided to train two types of models for the task: with *bert-large-uncased* and *bert-base-uncased* for generating embeddings of input sentences. These models differ in the number of parameters. bert-large-uncased has 335M parameters,

[1] https://wandb.ai/site/sweeps

λ	**intent_f1**	**slot_f1**	**Mean**	**WMean**
0.40	92.007	94.402	93.205	93.444
0.30	91.264	94.555	92.909	93.563
0.25	90.985	94.763	92.874	93.817
0.35	90.985	94.610	92.798	93.342
0.20	90.892	94.694	92.793	93.933
0.15	90.799	94.624	92.712	94.051
0.45	91.264	94.083	92.673	92.814
0.50	91.264	93.902	92.583	92.583
0.55	91.078	93.694	92.389	92.255
0.65	91.357	93.360	92.359	92.058
0.60	91.078	93.610	92.344	92.091
0.70	91.264	93.068	92.166	91.805
0.10	87.918	94.652	91.285	93.977

Table 1: Models trained with different intent loss weight parameters.
Mean is a mean of intent_f1 and slot_f1 scores.
WMean is a mean of these scores weighted by λ

while `bert-base-uncased` has only 109M parameters. It affects the speed of the forward propagation of the model: on average, a model with a smaller BERT was 3.77 times faster than the model with `bert-large-uncased`.

As it can be seen in Table 2, `bert-large-uncased` showed better results. While the difference in **Slot F1** score is not very big (only 1.002%), the difference in **Intent F1** is 4.171%, which could be caused by more accurate embeddings provided by a larger BERT.

	Intent F1		**Slot F1**		**Mean**	
	base	**large**	**base**	**large**	**base**	**large**
Mean	87.915	92.086	93.422	94.424	90.668	93.255
Std	1.307	0.634	0.509	0.457	0.756	0.405
Min	85.281	91.263	92.645	93.690	90.380	92.477
Max	89.870	93.401	94.057	95.005	91.638	93.863

Table 2: `bert-large-uncased` and `bert-base-cased` models using 10-Fold cross-validation

4.5 Intent classification

Fasttext framework was chosen (Joulin et al., 2016) to train an intent classifier. The model uses CBOW for word embeddings and a linear classifier on top of it. The model was trained using ten-fold cross-validation for 700 epochs with a learning rate of 1.0 and a 6-gram language model.

The statistics of the results are the following.

	F1
Mean	85.764
std	1.006
Min	84.201
Max	87.266

Table 3: Fasttext CBOW text classifier results for ten folds

4.5.1 Slot filling

I chose a Stanford NER classifier (Finkel et al., 2005) based on conditional random fields to perform this task. Results are worse than the results achieved by the BERT-based system. However, the model weighs only 18 megabytes, which is much smaller in comparison with a model with the BERT-large model, which weighs 1.35 gigabytes.

The results achieved on this model are the following.

	F1
Mean	75.138
STD	1.379
Min	72.840
Max	77.580

Table 4: Stanford NER classifier results for ten folds

5 Results and Analysis

It has been shown (not surprisingly), that BERT-based models outperform both fasttext (linear classifier) and Stanford NER models (CRF classifier). However, when it comes to the memory footprint of the models and their inference speed fasttext and Stanford NER models could still be the choice. However, it is possible to train smaller BERT models via knowledge distillation.

6 Conclusion

- It must be recapped that there is space for improvements in the NeMo framework. I plan to integrate WanbB sweeps into the framework in the future to make it more convenient to work with the framework.
- Since the data set I was using has some labeling errors: for instance, "`how would I bake the best tasting cookies ever found on this earth`" should be "`how would I bake the best tasting [dish_type : cookies] ever found on this earth`". Also, some labels are presented only in a few examples, there is a need to have someone fix the data set and increase the number of samples for different slots (e.g. by adding more song artists or book writers).

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 363–370, Ann Arbor, Michigan. Association for Computational Linguistics.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

Oleksii Kuchaiev, Jason Li, Huyen Nguyen, Oleksii Hrinchuk, Ryan Leary, Boris Ginsburg, Samuel Kriman, Stanislav Beliaev, Vitaly Lavrukhin, Jack Cook, et al. 2019. Nemo: a toolkit for building ai applications using neural modules. *arXiv preprint arXiv:1909.09577*.

Pawel Swietojanski Xingkun Liu, Arash Eshghi and Verena Rieser. 2019. Benchmarking natural language understanding services for building conversational agents. In *Proceedings of the Tenth International Workshop on Spoken Dialogue Systems Technology (IWSDS)*, pages xxx–xxx, Ortigia, Siracusa (SR), Italy. Springer.