Improved Error Detection for Model Based Interactive Semantic Parsing (MISP)

Pulkit Arya arya.35@osu.edu
Sujan Kakumanu kakumanu.3@osu.edu
Arpita Saha saha.139@osu.edu

Abstract

Model-based Interactive Semantic Parsing (MISP) presents a solution to the interactive semantic parsing problem by incorporating a model-based intelligent agent. A key part of that agent is a probability threshold based error detector which determines when a user should be prompted for clarification on a semantic parse. It was determined that this is a point of improvement for the MISP model. A simple probability threshold does not take into account the effect of features such as the current machine language clause being evaluated. We propose another method for error detection through a Feed-forward Neural Network (FNN) model. This solution was reached after constructing various classifiers and then testing the model performances on a test set against that of MISP's probability threshold performance. Compared to a probability threshold of 0.95, the value equipped by MISP with EditSQL for human evaluation, our FNN based error detection results in better string and token accuracy.

1 Introduction

A part of the Model-based Interactive Semantic Parsing (MISP) framework is an error detector that determines whether human intervention is needed for a part of a semantic parse. This error detector, based on an experiment outlined in the MISP paper, relies on a probability threshold (Yao et al., 2019). Based on this probability threshold, the model determines whether it needs to query the user for clarification or not. The goal of this project is to build a more robust error detector, which along with the probability of prediction, would also consider other features such as the type of SQL clause in the semantic unit to determine the need for interaction with the user.

2 Experimental Methodology

In order to tackle creating a new error detection method, we had to construct classifiers and evaluate performances before finally incorporating our new method into MISP. This section details those intermediate steps.

2.1 Experimental Setup

The MISP report states, in Section 5.1, that SQLova and EditSQL are used as base semantic parsers and they both share similar performances (Yao et al., 2019). Since the goal of our experiment is to improve error detection, we decided to select one base semantic parser to run our experiments: EditSQL. In addition, data needed to be extracted from running MISP with EditSQL. Every run of MISP results in a JSON file containing natural language prompts and their corresponding feedback records, among other items. From these feedback records, we extracted tag names, associated probabilities, the index of the current tag in the feedback record, and the user's response to the uncertain tag. Since the interaction was designed to be performed by a simulated user, we assumed the feedback to be "yes" if the response matched the gold token and "no" otherwise. We ran MISP with a probability threshold of 1.0 to ensure that a user's feedback is solicited for all tags.

From Figure 1 below, we can say that the data set is balanced since the target values are almost evenly distributed between 0 and 1, values responding a user interaction of 'no' and 'yes' respectively. So, accuracy can be a good measure of the evaluation metric.

2.2 Feature Prominence

We used XGBoost Algorithm to classify the features according to their importance value. XGBoost does this by comparing importance explic-

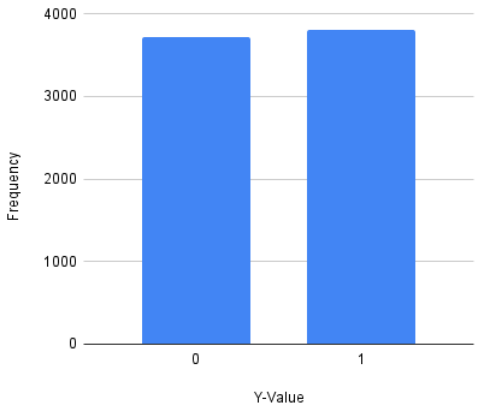

Figure 1: Distribution of target values

itly for each attribute and then comparing these importance values to rank them in decreasing order. For each decision tree, importance is calculated by the amount that each attribute split point enhances the performance measure, weighted by the number of records under that node. Then the mean feature importance is taken across all decision trees within the model (Morde, 2019). The greater an attribute's usage in making key decisions within decision trees, the higher its relative importance. It also reflects how valuable a feature was in the construction of a boosted decision tree within the model. As shown in Figure 2, it can be seen that the tag probability, which was the probability associated with the tag and compared with the threshold was the most important feature. The second most important feature was the tag name, which signified the type of clause ('SELECT', 'WHERE', etc) specified in the semantic unit. The other two features, namely "#InFeedback", which indicates the number of semantic token currently being parsed and the probability threshold did not hold much significance and thus were later dropped out of the training mechanism.

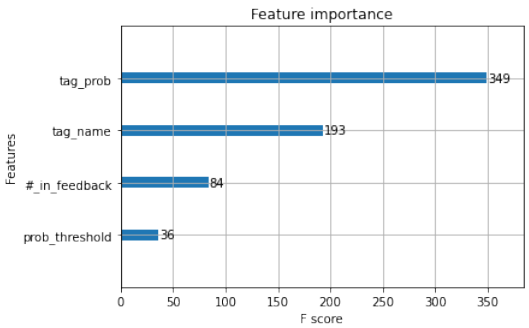

Figure 2: Feature Importance According to XGBoost

2.3 Construction and Evaluation of Preliminary Classifiers

We determined that basic machine learning algorithms should be trained to establish a baseline performance to which more advanced algorithms, along with MISP's existing error detection, can be compared. Figure 3 below lists these algorithms along with their performance on data extracted from MISP with a probability threshold of 1.0.

Classifier	Features	Test-Train Split	Log loss	Accuracy Score (Jaccard)
Decision Tree	Tag Probability, Tag name	0.2	8.956334767923734	0.7406914893617021
SVM	Tag Probability, Tag name	0.2	11.482484615819072	0.6675531914893617
Logistic Regression	Tag Probability, Tag name	0.2	11.138011183170734	0.6775265957446809
Gausian Naive Bayes	Tag Probability, Tag name	0.2	15.018861541386894	0.5651595744680851
K Nearest Neighbors	Tag Probability, Tag name	0.2	9.806044018270645	0.7160904255319149

Figure 3: Preliminary Classifier Attributes and Performances

As Figure 3 shows, the Decision Tree classifier performs the best on the test set, with an accuracy of 0.74. This might owe to the fact that decision trees are known to perform better in small to medium tabular datasets. Despite this result, it is known that this classifier is also prone to overfitting and upon further inspection, this was confirmed to be the case. Moreover, due to some error, we were not able to incorporate the Decision Tree Model into MISP to check if it improved performance of the Error Detector.

2.4 Construction and Evaluation of Advanced Classifiers

The Feed Forward Neural Network, FNN for short, consists of two hidden layers with a sigmoid activation function and a final output layer with sigmoid activation as well. Each hidden layer has 200 neurons. The model expects as input a tensor of size of 13. The first 12 columns are one hot encoding of the tag name of the semantic unit and the last column is the probability of the semantic unit. We create an indexer from our training data to assign indices for each type of semantic unit. As it can be seen in Figure 4,

3 Incorporation into MISP

Since the error detector can be trained separately, the FNN based error detector was trained while being hosted on Google Colaboratory. This trained

Model	Accuracy on Test Set
FNN	0.678
Prob = 1.0	0.493
Prob = 0.995	0.516
Prob = 0.95	0.499
Prob = 0.90	0.486
Prob = 0.80	0.467
Prob = 0.75	0.459
Prob = 0.65	0.451
Prob = 0.50	0.434
Perfect	N/A

Figure 4: FNN Accuracy on the Test Set, compared to MISP run on different probability thresholds.

model was saved as a pickle file - a way to serialize Python objects - so that it could be brought into MISP. In MISP, we added an option to run the code with an FNN based error detector. This new subclass of ErrorDetector is called ErrorDetectorFNN. On execution, –err_detector fnn can be passed as the command line argument to run ErrorDetectorFNN as the ErrorDetector running under the agent.

ErrorDetectorFNN has two attributes, the FNN model and an indexer. The indexer is used to map the names of semantic units to numbers that were used to convert tag_name of semantic unit into one hot encoded vector representation. There are a total of 12 semantic units. We use one column for each of the semantic units, and one column for the probability value attached to the semantic unit. The input is constructed using the name of the semantic unit and the probability attached with it in the form of a Torch Tensor.

4 Results

Running MISP using our FNN based error detector lead to an improvement over the threshold based error detector. We achieved a token accuracy of 79.79% while the token accuracy of threshold based error detector is 73.75% for a probability threshold value of 0.8, and 76.98% for a probability threshold value of 0.95. We also conducted other tests with varying probability threshold values to see the impact of higher probability threshold values on token accuracy. As it can be seen in Figure 5, our error detector was able to perform as well as the threshold based error detector running with a probability threshold of 1.0. However, setting the threshold probability to 1.0 means the error detector marks each semantic unit as uncertain and asks the user for clarification. Thus, making it a trivial case as an error detector. Our error detector also achieved a string accuracy of 55.22% as compared to 35.4% for probability threshold of 0.8 and 45.33% for probability threshold of 0.95.

Model	String Accuracy	Token Accuracy
FNN	55.22	79.77
Prob = 1.0	56.29	80.01
Prob = 0.995	50.58	78.33
Prob = 0.95	43.33	76.98
Prob = 0.90	39.26	75.58
Prob = 0.80	35.4	73.75
Prob = 0.75	34.53	73.04
Prob = 0.65	33.17	72.25
Prob = 0.50	31.14	71.01
Perfect	56.29	80.01

Figure 5: String and token-wise accuracy of the FNN model, compared with MISP run with different probability thresholds

User interaction is an important feature of MISP, however, a high accuracy must be obtained while asking the least number of questions to the user. The threshold based error detector offers a high accuracy at the cost of asking more questions to the user. This correlation can be observed as a higher probability threshold entails a higher average number of questions per example.

From Figure 6 we see that our error detector asks an average of 3.3 questions per example, which is quite high, however, the threshold based error detector need 4.0 questions per example to obtain the same accuracy as us. We reduce the number of questions required while achieving the high accuracy offered by setting a high probability threshold.

Model	No. of Questions	No. of Questions / example
FNN	3488	3.373
Prob = 1.0	4213	4.074
Prob = 0.995	1929	1.866
Prob = 0.95	1240	1.199
Prob = 0.90	938	0.907
Prob = 0.80	606	0.586
Prob = 0.75	503	0.486
Prob = 0.65	337	0.326
Prob = 0.50	123	0.119
Perfect	1241	1.2

Figure 6: Number of questions asked during interaction with the FNN model, compared to MISP run with different probability thresholds

To gauge a better understanding of our results, we also compared our error detector with a perfect error detector that is a part of MISP. The perfect error detector already has knowledge of the gold tokens and uses that to identify situations that require user interaction. Our error detector offers the same accuracy as the perfect error detector, while requiring more questions. It is expected that our

error detector would require more questions as it does not have knowledge of the gold tokens and in real user interaction scenarios the error detector will not have knowledge of gold tokens.

5 Conclusion and Future Work

For the sake of these experiments, MISP was only run on 10 percent of the Spider dataset, generating a smaller amount of data than possible. Our models can be trained and tested on data collected from MISP being run on the entirety of Spider. In addition, we only ran MISP with EditSQL; much more data can be collected if MISP were to be run with SQLova as well.

Other models such as Many-to-Many Recurrent Neural Networks can be explored e.g. by feeding the probability response and tag name of a token at each time step, denoted by the number of semantic unit we are currently considering in the semantic unit. The output from each time-step will determine whether it is a gold token or not. Since the hidden layer from previous time step is fed into the current time step, this will efficiently be able to capture the temporal relationship between the semantic units in the feedback record.

Convolutional Neural Networks, which are performing well on tabular data, such as Soft1DCNN (Villanueva, 2021) can be explored. With greater amounts of data, these deeper models might unleash some promising directions for us to work on in the future.

We can also try out Few Shot Learning (Dhanya, 2021) useful for training on small quantities of available data. The model will be trained to learn the similarities and differences between the trend of probability values that output 1 for gold token and 0 otherwise. Using this the model will be able to accurately predict whether 1 should be output or 0.

6 Group Contributions

- Pulkit: After reading and discussing the MISP report with professor Sun, formulated the problem statement to improve upon the error detector. Formulated structure of data to be generated from MISP required to train error detectors. After successful data collection and preliminary model training, noted over fitting of Decision Tree after inspecting the structure of trained model. Further, tweaked the FNN model to increase accuracy from 42% to 67.8% on test set, via hyperparameter. To increase model performance, converted tag name category to one hot encoding, added hidden layers, tested different widths for hidden layers, in combination with various loss functions and optimizers. After developing the model with best test set accuracy, saved the model to make usable with MISP. Created a new error detector subclass for the MISP agent to use called ErrorDetectorFNN. Worked on section 2.4, parts of section 3, and section 4.

- Sujan: Initially, read and compiled notes on the MISP report and set up meetings and online folders for collaboration. For the midterm report, helped in narrowing down the problem statement and creating a timeline of work for the team to follow. For the final report, created a Python parser to extract data from the MISP output JSON files. This data, along with the parser, was used to train the models mentioned in the above experiments. Initially built an FNN that only achieved 42% test accuracy but this code was handed off to Pulkit who made significant changes and improvements. Set up the Overleaf LaTeX file for our report by creating the section and subsection structure. Worked on the Abstract, Sections 1, 2, parts of 2.1, parts of 3, parts of 5, and Section 7. Also set up the PowerPoint file to be used for the presentation.

- Arpita: Read the MISP paper and ran and went through the MISP code; contributed to problem definition and formulation before midterm and later to brainstorming on how to use the output from MISP to generate give the features we are seeking for; ran MISP multiple times with different probability threshold to help in gathering data that was later used by Sujan to parse into a tabular format; ; ran various preliminary classifiers and used XGBoost for feature importance and identifying the most important features that were later used by Sujan and Pulkit to train and build a robust FNN model; tried playing with deeper models [Soft1DCNN, known to perform well on tabular data, which was giving 67% test accuracy and 0.582 validation loss on 70-10-20 train-validation-test set that was lower than the reported FNN accuracy due to shortage

of data (only around 7k rows in total) so it was not reported in the main text and has been left for future study]; wrote parts of section 1, parts of section 2.1, parts of section 5 (second, third and fourth paragraphs), section 2.2 and section 2.3 in the report.

- Time Breakdown: We had Zoom and in-person meetings throughout the semester that took 23 hours. We spent about 2 hours on the midterm report and 7 hours on compiling the final report and final presentation. Collectively, collecting data, generating the models, and running experiments took 17 hours.

7 Code Repositories

- MISP Experiment Repository: This contains the code used for our experiments, such as the data parser, initial construction of the classifiers, and pickled models.

- Fork of the MISP Repository with new Error Detector: This contains MISP code that has been augmented with the FNN Error Detector.

Acknowledgments

Thank you to Dr. Huan Sun and our TA, Ron Chen, for helping us navigate this project and providing guidance on MISP.

References

Shree Dhanya. 2021. Fewshot learning article. analyticsvidhya.com/blog/2021/05/an-introduction-to-few-shot-learning/.

Vishal Morde. 2019. XGBoost algorithm description. https://towardsdatascience.com/edd9f99be63d. Accessed: 2021-12-11.

Martín Villanueva. 2021. Soft1dcnn article. https://medium.com/spikelab/4abdd67795b6.

Ziyu Yao, Yu Su, Huan Sun, and Wen-tau Yih. 2019. Model-based interactive semantic parsing: A unified framework and A text-to-sql case study. *CoRR*, abs/1910.05389.