Towards Automated Error Analysis: Learning to Characterize Errors
Tong Gao, Shivang Singh, Raymond J. Mooney

Department of Computer Science
University of Texas at Austin
\{gaotong, sssingh, mooney\}@cs.utexas.edu

Abstract
Characterizing the patterns of errors that a system makes helps researchers focus future development on increasing its accuracy and robustness. We propose a novel form of “meta learning” that automatically learns interpretable rules that characterize the types of errors that a system makes, and demonstrate these rules’ ability to help understand and improve two NLP systems. Our approach works by collecting error cases on validation data, extracting meta-features describing these samples, and finally learning rules that characterize errors using these features. We apply our approach to ViLBERT, for Visual Question Answering, and RoBERTa, for Commonsense Question Answering. Our system learns interpretable rules that provide insights into systemic errors these systems make on the given tasks. Using these insights, we are also able to “close the loop” and modestly improve performance of these systems.

1 Introduction
Although deep-learning systems have made remarkable progress in recent years, systems still make significant errors on complex and diverse tasks. An important step in engineering highly-accurate, robust systems is error analysis, which has been defined as the process of examining development set examples misclassified by the algorithm and understanding the underlying causes of those misclassifications. This process helps engineers prioritize and address critical problems and improve system performance and robustness.

Unfortunately, such analysis usually needs manual inspection and reasoning, which is an onerous, time-consuming and hit-or-miss process. Pure manual analysis may lead to a biased conclusion, as common features appearing in both successful and failed classifications could be misunderstood as the root cause of failure (Rondeau and Hazen, 2018). While others have proposed improved approaches to error analysis (e.g. Wu et al., 2019; Chung et al., 2020), some are not scalable to large-scale datasets, hindering their practical application, while others require a deep understanding of the errors even before conducting the proposed method. In contrast, our pipeline (1) makes weak prior assumptions about the attributes of failures, (2) is scalable to large-scale datasets, (3) groups errors with interpretable and globally unbiased rules, enabling fast manual inspection and analysis. Developers can benefit from such a systematic view and develop patches addressing different problems disclosed by different sets of rules.

Specifically, we propose using machine learning to help automate error analysis by inducing interpretable rules that characterize the errors that a system makes. First, our approach runs a predictive model on a set of held-out validation data and records which examples the system classifies correctly or incorrectly. Next, we characterize each example using a set of “meta-features” that describe the problem (e.g. for question answering, a meta-feature could be: question starts with “How many”). Finally, we learn interpretable rules from this data which predict failure using these meta-features (Figure 1). To demonstrate the effectiveness of our approach, we present results applying our approach to ViLBERT (Lu et al., 2019) for Visual Question Answering v2.0 (VQA) (Goyal et al., 2018), and RoBERTa (Liu et al., 2019) for CommonsenseQA (CSQA) (Talmor et al., 2019). We induce human-interpretable rules providing insights into systemic errors these systems make on these tasks. We then show how to “close the loop” and modify the systems to improve their performance using some of these insights. A longer version of this paper with more details is available at https://arxiv.org/abs/2201.05017.

2 Automatic Error Characterization
Figure 1 characterizes our approach. It first runs a pretrained system on development data not used during training and collects its predictions. Next, these examples are described using extracted meta-features that characterize properties of the problems. Finally, we run a rule learner that uses these meta-features to characterize the model’s mistakes.

2.1 Data Preprocessing
Meta-feature Extraction Meta-feature engineering is a domain-specific process requiring understanding of the problem as well as the properties that affect model behavior. For our two QA tasks, language tokens in the questions and the gold answers are the most important meta-features. For images in VQA, we added meta-features representing objects in images detected by YOLOv3 (Redmon and Farhadi, 2018) as well as generated image captions (Luo et al., 2018), which may encode extra properties and relations for the objects in the image. All the text tokens are then
lemmatized and part-of-speech tagged, providing additional meta-features. We lookup all nouns and verbs in WordNet (Miller, 1995) and add their hypernyms up to 4 levels as additional meta-features. Hypernyms help group similar features across examples and allow more abstract characterizations of error cases. Finally, stopwords are removed and all meta-features are represented by a sparse vector, where each entry represents the occurrence frequency of one meta-feature in an example. Figure 2 gives an overview of this process for VQA.

Pre-clustering Characterizing a large dataset is difficult for existing rule learners due to issues scaling to large numbers of examples and features. Hence, motivated by discriminative clustering (Bansal, Farhadi, and Parikh, 2014), we first cluster the dataset into several smaller sub-datasets and run the rule learner on each cluster separately. This pre-clustering step speeds up the rule extraction process and helps arrange the resulting rules in a semantically meaningful way. We ran k-means on the error cases (i.e. positive examples) and build 2 clusters for CSQA. Both of them are combined with all negative examples, creating 2 smaller overlapping sub-datasets for rule learning. Similarly, we ran k-means on VQA error cases and found that words that typically represent question types (e.g. “why”) are all close to cluster centers. To obtain finer-grained clusters that better divide the data, we proceeded with this observation and categorized the dataset into 65 clusters by question types which are given in the annotations for this data.

2.2 Rule Learning

We automatically induce a model that predicts success or failure on an example given its meta-features. For this task we used SkopeRules (Gardin et al., 2017), a rule learner which provides us interpretable insight into the errors made by the target model.

Feature Selection In practice, the raw extracted meta-feature dataset is high dimensional, sparsely populated, and noisy. This prevents the rule learner from generating effective rules, justifying the necessity of dimensionality reduction. Following Yang and Pedersens (1997), we used a chi-square feature selection method as it is an efficient way to reduce the number of features. We performed grid search on the possible number of dimensions and selected 100 meta-features as the best value.

Iterative Learning In order to make the rule learner more efficient and effective, we run it multiple times with a high precision threshold (60%) for the rules it produces, redoing feature selection at each iteration to allow new rule sets to potentially focus on a new set of important features. At the beginning of each iteration, chi-square is used to select features and SkopeRules is run on the current training set to generate a set of high-precision rules for identifying error cases. We then remove the cases covered by these rules and repeat this process until no more error cases can be characterized with high precision, or until a maximum number of iterations is reached, which is 50 in our configuration.

Rule Evaluation and Filtering To ensure learned rules accurately predict errors and do not just overfit the development set, we created a 90/10 training/test split from the original development set for the rule learning process. Rules...
are then learned from the training subset and evaluated on the test subset. Any rule that is not at least 60% accurate on the test split is removed from the final rule set.

3 Experimental Results

We ran our pipeline on ViLBERT pretrained on VQA, and RoBERTa finetuned on CSQA. We collected 404 and 81 rules covering 14.1% and 60.8% of error examples for two tasks respectively. Examining the rules learned from our approach, we were able to gain deeper insights into both tasks. Below we highlight and discuss some of the error patterns that we have detected.

VQA The major category of errors that we identified involves reading text from images, therefore requiring Optical Character Recognition (OCR). Sample rules together with a covered problem instance are shown in Figure 3. Since the lack of OCR ability is the most significant problem uncovered by our system, we propose a mixed architecture to alleviate it in Section 4. Besides, several rules indicate that examples that require describing the position of an object are likely to cause failure, even if the answer is reasonable. This illustrates a problem with the scoring method rather than with the model. In general, this requires determining whether two referring expressions denote the same object in an image, a difficult language-vision problem.

CSQA We obtained several rules for RoBERTa on CSQA, that highlighted specific types of common sense knowledge that the model is lacking. For instance, the rules learn that RoBERTa is unable to answer questions related to “American states”, suggesting that more prior knowledge about this topic would be helpful. More examples of concepts that seem to be the source of errors are shown in Figure 3.

4 Improving Model Accuracy

The ultimate goal of error analysis is use the insight gained about a model’s weaknesses to “close the loop” and develop an improved model. Here we present our results on using the understanding gained from examining the rules learned by our approach to increase model accuracy.

4.1 ViLBERT for VQA

Not only do several learned rules indicate that ViLBERT has problems with reading text, we can use these rules to automatically identify cases where applying OCR might prevent the model from making a mistake. Therefore, we tried to utilize a state-of-the-art OCR-VQA system, M4C (Hu et al.), to improve ViLBERT. Specifically designed for OCR-VQA, M4C generates answers conditioning on additional text extracted by an OCR system. We used Mask TextSpotter v3 (Liao et al., 2020) for OCR. To ensure M4C fits the right distribution, we trained M4C on the official training set of VQA, then finetuned M4C on training examples whose images have text. During testing, we picked the 40 most accurate OCR rules (by F-score) to identify OCR questions, and redirect these questions to M4C. The final prediction of our system is a mixture of ViLBERT and M4C, depending on whether the rules indicate the question requires OCR.

4.2 RoBERTa for CSQA

The error-analysis rules learned for RoBERTa on CSQA point to a number of general conceptual areas where the model exhibits weaknesses. One approach to improving the model based on this insight is to train the model on additional relevant data to correct for these deficiencies. We observed that while Wikipedia is a part of the pretraining dataset, RoBERTa can still “forget” critical knowledge during large scale pretraining. Hence, we use a dataset refinement approach to ensure that RoBERTa uses the pre-training data effectively for optimizing CSQA performance. We ranked discovered rules by their F-scores and picked 11 most interpretable rules for demonstration, which themselves consist of a keyword list as a summary of RoBERTa’s “knowledge gap.” Next, we filtered Wikipedia data by retaining sentences containing these keywords or one of their hyponyms, as well as their neighboring sentences. Afterward, RoBERTa was first finetuned on this filtered Wikipedia data and then finetuned again on Commonsense QA, obtaining 78.7% accuracy on the development set.

As shown on Table 2, our “refocused” RoBERTa obtained a 0.35 percentage point improvement on the CSQA test set, which is nearly the best among RoBERTa’s single model variants. We also show the best performance that the single RoBERTa model could achieve, which is trained with G-DAUG-Combo data augmentation technique. (Yang et al., 2020) Compared to this method, we did not augment the dataset with complex synthetic data but only refocused training on critical aspects of the original training data and still obtained 70% of the additional improvement gained by the best-performing model. The refocused model corrects several of the original errors shown in Figure 3, illustrating the effectiveness of our approach.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoBERTa + G-DAUG-Combo</td>
<td>72.6</td>
</tr>
<tr>
<td>Ours</td>
<td>72.45</td>
</tr>
<tr>
<td>RoBERTa (Baseline)</td>
<td>72.1</td>
</tr>
</tbody>
</table>

Table 2: Single RoBERTa’s performance on CSQA test split, collected from the public leaderboard compared to our “refocused” model.

The results are in Table 1 and show this ensemble is better than either of the individual systems. The improvement is modest due to the difficulty of OCR problems and relatively low fraction of them in the data (only 3.86% of test questions are identified as OCR problems). However it illustrates how the learned rules can help direct system improvement.

<table>
<thead>
<tr>
<th>Data Split</th>
<th>dev</th>
<th>test-dev</th>
<th>test-std</th>
</tr>
</thead>
<tbody>
<tr>
<td>M4C</td>
<td>53.17</td>
<td>54.98</td>
<td>55.28</td>
</tr>
<tr>
<td>ViLBERT</td>
<td>68.75</td>
<td>69.47</td>
<td>69.64</td>
</tr>
<tr>
<td>Mix</td>
<td>69.44</td>
<td>69.64</td>
<td>69.82</td>
</tr>
</tbody>
</table>

Table 1: Performance on VQA v2.0
Figure 3: Rules the system had discovered. On the left hand side we show examples of rules for VQA. On the right hand side we show CSQA rules.

5 Conclusion

In this paper, we presented a novel pipeline that helps automate the error analysis process by learning interpretable rules that characterize the type of mistakes that a system makes. We then demonstrated its effectiveness by applying this pipeline to two different tasks. Presented in the form of well-organized rules, deficiencies and weaknesses in the model, datasets, and even the evaluation metrics were disclosed to researchers, shedding light on potential directions for improvement. We demonstrated the ability to “close the loop” and use the insight gained from some of the induced rules to make modest improvements to these systems. These simple but effective approaches have the potential to be applied in production environment shortening the iterative update cycle of models.

Acknowledgments

This research was supported by the DARPA XAI program under a grant from AFRL. We would like to thank Bill Ferguson and his colleagues at Raytheon for inspiring our work.

References


Hu, R.; Singh, A.; Darrell, T.; and Rohrbach, M. Iterative answer prediction with pointer-augmented multi-modal transformers for TextVQA. In CVPR.


