Explanatory Reinforcement Learning from Human Feedback (RLHF) for Concept-based Models

Martijn Elands
martijn.elands@studenti.unitn.it
martijn.elands@student.maastrichtuniversity.nl

Professor Stefano Teso (supervisor)
stefano.teso@unitn.it

Advanced Topics in Machine Learning & Optimization [145871]
Department of Information Engineering and Computer Science
University of Trento
Trento, Italy

Abstract

This project revolves around debugging machine learning models. The idea is to implement a form of explanatory interactive learning – which normally relies on expert annotations on a model's explanations – using ideas from reinforcement-learning from human feedback. Specifically, instead of using (many) explanation-level annotations to debug the model directly, we use them to train a *ranker* for distinguishing between good and bad explanations, and then fine-tune the model to make the ranker happy. At a high-level, the key question is whether this setup can help reducing annotation costs.

1 Introduction

Recent advancements in Deep Learning have led to big improvements in task-based metrics. However, they come with a trade-off which is explainability and interpretability. This boosted the Explainable AI (xAI) field in Machine Learning. Some methodologies try to explain these black boxes by, for example, post-hoc explanations like LIME [1], SHAP [2], and gradients-based methods like in [3]. Unfortunately, as they are post-hoc, there is no way to interact with the found issues directly.

Concept-Bottleneck Models (CBMs) [4] are designed in a way that you can interact with them through the bottleneck. Instead of directly predicting the label from the input features, CBMs learn an intermediate representation of "concepts" that need to be human-interpretable by design and to be labeled with input by humans. The concepts serve as explanations and they can expose potential issues with the model, allowing engineers to tweak it. Research on interactive machine learning has shown that explanations are a good way to refine a machine learning model [5] [6]. A similar work has been carried on by [7], but here a user was required to provide concepts during prediction time.

However, these methods require a human-in-the-loop. Recently a new framework, called Learning by Self-Explaining (LSX) [8], was developed that sits in the crossover between explanatory interactive learning (XIL) and self-refinement machine learning. In this work, the human is replaced by a "critic" which gets the original input and explanations from the original model. It outputs feedback, which is used to revise the original model (also known as "learning"). This is a dual-architecture, so the model and the critic are jointly trained. This means that the critic is also optimized for the labels indirectly. It could be that the model is buggy (e.g. trained on a confounded dataset), and thus the explanations do not make sense, but the critic cannot distinguish this.

This project focuses on using a combination of a critic and a human. We use a ranker model as the critic and detach its learning from the learner. The ranker's goal is to rank good and bad explanations coming from an oracle and a buggy machine learning model respectively. The goal is to use the ranker with human feedback to fine-tune the model. In this way, human decisions are amplified and the idea is to reduce labeling costs. The code is publicly available on (GitHub).

2 Dataset

The CUB200-2011 [9] dataset was used. This is a dataset with 200 types of birds, and 312 concepts of birds. The concepts were filtered in the same way as previous approaches (CBM [4] and CEM [10]) because they were noise due to the non-expert human annotations. The total number of concepts used is 112.

2.1 Confounded version

In order to test the approach, the machine learning model needed to be buggy. Hence, a special version of the train set was made that confounded the labels with the images. This was done by placing blue non-overlapping blocks in the image. As the model (see section 3.1 for more details) took a 299x299 input and there are 200 classes, the biggest possible block was 19x19. The blocks were placed from the left top of the image moving to the right, and then down by a block until the number of classes were reached. The area of the block is only 0.4% of the area of the image, but clearly noticeable.

3 Methods

3.1 Buggy model & Oracle

This project uses a custom version based on the implementation of CBMs provided by the authors of CEM [10] because this version is compatible with a newer version of PyTorch [11]. Concepts Bottleneck Models have two tasks to learn: going from the input to the concepts, and from the concepts to the labels. To potentially compare results to the ones obtained in CEM, we used a pre-trained InceptionV3 [12] model to predict the concepts from the input and fine-tuned on our dataset. This model expects an image of dimensions $(299, 299, 3)$, so the images were center cropped to this size. A simple linear layer was used to learn the labels from the concepts.

For creating the buggy model, the train set was confounded as described in section 2.1. The validation and test set were untouched. The oracle is trained on the complete dataset, so it includes the test, but not the validation set. There are two different experiments with what training data, which are later described in the experiments (Section 5.1).

3.2 Ranker

Assuming the previous models are sufficiently trained, the ranker can be trained. The ranker's task is to give a lower score to the buggy model (that is trained on the confounded dataset) than the oracle. When checking the ranker, we can see what importance it gives to the features of the two models. There are experiments with the configuration of the ranker. In general, the input of the ranker can be summarized as follows: $r(\mathbf{x}, \mathbf{m})$ where $\mathbf{x}$ is the input image, and $\mathbf{m}$ is the model. Therefore, the following should hold:

$$r(\mathbf{x}, \mathbf{f}) < r(\mathbf{x}, \mathbf{o})$$

4 Experiments

First, the oracle was trained, and then the buggy model. This was to ensure that there was enough performance to back up the ranker. If the oracle performed badly, then learning the ranker would have been hard as there is a low knowledge gap between the two models. Although initial results showed that the oracle and buggy model performed well with only 40 epochs, further investigation showed different results. Therefore, the models were updated accordingly. The oracle was trained to 100 epochs to show satisfactory results, and the confounded models up to 50 epochs.

Table 1: Performance of the oracle model on the train, validation, and test split for task accuracy and concept area-under-curve.

	Task accuracy			Concept AUC		
Epochs	Train	Val	Test	Train	Val	Test
10	0.4410	0.5501	0.5364	0.9533	0.9748	0.9739
20	0.6687	0.7930	0.7969	0.9798	0.9936	0.9931
30	0.7258	0.8648	0.8695	0.9852	0.9974	0.9971
40	0.8217	0.9207	0.9280	0.9943	0.9993	0.9993
50	0.8374	0.9316	0.9375	0.9949	0.9995	0.9995
60	0.8397	0.9316	0.9375	0.9954	0.9995	0.9995
70	0.8378	0.9307	0.9375	0.9951	0.9995	0.9995
80	0.8397	0.9316	0.9379	0.9953	0.9995	0.9995
90	0.8480	0.9299	0.9380	0.9955	0.9995	0.9995
100	**0.8784**	**0.9683**	**0.9651**	**0.9971**	**0.9999**	**0.9999**

The oracle has been trained with two different sets of data, see Sections 5.1.1 and 5.1.2.

Afterward, the ranker was trained. A few configurations have been tried with the inputs ($\mathbf{c}_m(\mathbf{x})$ and $\mathbf{w}_m$ are the concepts for input $\mathbf{x}$ and the weight of the last linear layer from the concepts to label prediction):

- Config 1, all four: (x, c, w, y). Ranker $r(\mathbf{x}, \mathbf{c}_m(\mathbf{x}), \mathbf{w}_m, m(\mathbf{x}))$.
- Config 2, without input image: (c, w, y). Ranker $r(\mathbf{c}_m(\mathbf{x}), \mathbf{w}_m, m(\mathbf{x}))$.
- Config 3, without predicted label: (x, c, w). Ranker $r(\mathbf{x}, \mathbf{c}_m(\mathbf{x}), \mathbf{w}_m)$.
- Config 4, only concepts and weights: (c, w). Ranker $r(\mathbf{c}_m(\mathbf{x}), \mathbf{w}_m)$.

The main parameters of the ranker, together with the above configurations are:

- Margin ranking loss with a margin of 5. This ensures that the learned representations are further apart.
- Linear rate of 1^{-5} and a scheduler to reduce the learning rate on plateau.
- one linear layer to go from the inputs (depending on the configuration) to the ranking score.
- Number of epochs: only 2

5 Results & Discussion

5.1 Training of the oracle

The oracle has been trained with two different strategies. One strategy (see Section 5.1.1) was to include the test data with the train data so the oracle had an all-knowing view of the dataset. The other strategy was to train the oracle on the confounded train set from Section 2.1 and on the normal, non-confounded test set. This last strategy is presented in Section 5.1.2.

5.1.1 Oracle on train and test

Table 1 shows the performance of the oracle $\mathbf{o}$. The final version of the oracle was picked to be the one trained on 100 epochs as this version had the best-performing metrics. However, there is still room for improvement as the train task accuracy is not yet on the same level as the validation and test task accuracy (which might indicate underfitting). Initial results showed that 40 epochs were already enough, however, this needed to be adjusted and the table needed to be recalculated.

5.1.2 Oracle on CF train and NCF test

Table 2 shows the performance of the oracle $\mathbf{o}$ on the confounded train set, and non-confounded test set. The final version of the oracle was picked to be the one trained on 100 epochs as this version had the best-performing metrics for all data splits. The learned model is good at predicting both confounded and non-confounded data.

Table 2: Performance of the oracle model (trained on confounded train and unconfounded test data) on its training data, the training, validation, and test split for task accuracy and concept area-under-curve. The combination of confounded training and normal test data is called PCFT (partial confounded training.

	Task accuracy				Concept AUC			
Epochs	PCFT	Train (CF)	Val	Test	PCFT	Train (CF)	Val	Test
10	0.5035	0.5048	0.5075	0.6703	0.9592	0.9606	0.9687	0.9857
20	0.8173	0.8196	0.7137	0.9066	0.9931	0.9930	0.9861	0.9987
30	0.9030	0.9066	0.7287	0.9537	0.9975	0.9978	0.9847	0.9995
50	0.9898	0.9898	0.8080	0.9896	1.0000	1.0000	0.9925	1.0000
100	**1.0000**	**1.0000**	**0.8122**	**1.0000**	**1.0000**	**1.0000**	**0.9933**	**1.0000**

Table 3: Performance of the normal model (“buggy") on the confounded train, and normal validation and test split. Metrics are task accuracy and concept area-under-curve.

	Task accuracy				Concept AUC			
Epochs	Train	Train CF	Val	Test	Train	Train CF	Val	Test
10	0.1856	0.197	0.1219	0.1847	0.9292	0.9354	0.9331	0.9342
20	0.3624	0.7998	0.3230	0.3230	0.9389	0.9942	0.9452	0.9456
30	0.3782	0.9412	0.3698	0.3916	0.9246	0.9995	0.9367	0.9365
40	0.2950	0.9946	0.2980	0.3091	0.8976	1.0000	0.9182	0.9174
50	**0.1797**	**0.9954**	**0.1928**	**0.2113**	**0.7987**	**1.0000**	**0.8512**	**0.8541**

5.2 Training of the buggy model

Table 3 shows the performance of the “buggy" machine learning model $\mathbf{f}$. This can be seen by the high performance on the confounded train set, and low performance on the non-confounded train, test, and validation set. The final version of $\mathbf{f}$ was picked to be the one trained on 50 epochs as this already showed that it had learned the shortcut by focusing on the square box that was given in the image. Also here, the model could be more confounded when trained longer, as the performance of the normal splits goes down from the model trained on 40 to 50 epochs.

5.3 Inspection of oracle and shortcut model

Before proceeding, manual inspection took place to make sure the model was actually buggy. Appendix A has an example of the prediction of the trained oracle from Section 5.1.2 and the buggy model.

Furthermore, after this investigation, there was a strong suspicion that the buggy model learned to overclassify one class when it was presented with the test set. After further investigation, presented in Appendix B, this was confirmed.

5.4 Training of the ranker

For the ranker, training on only 2 epochs was already enough to let the loss go from the margin to 0 for both versions of the oracle. Furthermore, a simple inspection was done on the learned weights due to time reasons. The top 2500 most influential weights were filtered and assigned to the input features. The results are presented in Tables 4 and 5 depending on the used oracle.

Table 4: Ranker performance. X denotes that the feature was not available for the ranker. The ranker was trained using the confounded training data.

	Loss			Important weights			
Type	Train	Val	Test	$\mathbf{x}$	$\mathbf{c}_m(\mathbf{x})$	$\mathbf{w}_m$	$\mathbf{m}(\mathbf{x})$
Config 1	0.0000	0.0018	0	0	9	2461	30
Config 2	0.0000	0.0007	0	X	6	2472	22
Config 3	0.0000	0.0000	0	0	13	2487	X
Config 4	0.0000	0.0000	0	X	14	2486	X

Table 5: Ranker performance. X denotes that the feature was not available for the ranker. The ranker was trained using the confounded training and normal testing data.

	Loss			Important weights			
Type	Train	Val	Test	$\mathbf{x}$	$\mathbf{c}_m(\mathbf{x})$	$\mathbf{w}_m$	$\mathbf{m}(\mathbf{x})$
Config 1	0.0061	0.0248	0.0000	0	12	2448	40
Config 2	0.0093	0.0165	0.0158	X	8	2475	17
Config 3	0.0001	0.0001	0.0000	0	9	2491	X
Config 4	0.0114	0.0107	0.0000	X	13	2487	X

In general, the ranker puts the most importance on the weights of the linear layer between the concepts and the final prediction layer. However, these measurements are not a robust approach and should be handled with care as the scale of the input of the weights is completely different compared to the scale of the other features (predicted concepts, etc).

6 Conclusion

This project was designed to dive into a possible pathway of combining a ranker to integrate Reinforcement Learning from Human Feedback with Concept Bottleneck Models to improve machine learning model debugging while reducing annotation costs. Unfortunately, the implementation of the ranker with RLHF was not within the current time allowance, but we showed that we can leverage a ranking model to distinguish between good and bad explanations of models. Therefore, this will probably have great opportunities to fine-tune a model, fix bugs in the data, and prevent learning shortcuts.

An advantage of this technique is that the model and the ranker are not jointly optimized for a certain objective. Hence, the ranker could lead us to see that there are some shortcuts that the model learned. Challenges remain in optimizing the training strategy for the ranker and connecting it to an RLHF pipeline. This will include fine-tuning the model and testing it with the ranker to see if its score increases. A drawback of this approach is that there needs to be access to the unconfounded test set. If this is not the case, then the ranker cannot be constructed. Furthermore, this time the CUB-200-2011 dataset was manually confounded with a bright blue block. Different approaches should be investigated in confounding this dataset and using other datasets.

References

[1] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939778. URL https://doi.org/10.1145/2939672.2939778.

[2] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[3] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. *J. Mach. Learn. Res.*, 11:1803–1831, August 2010. ISSN 1532-4435.

[4] Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5338–5348. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/koh20a.html.

[5] Yuyang Gao, Siyi Gu, Junji Jiang, Sungsoo Ray Hong, Dazhou Yu, and Liang Zhao. Going beyond xai: A systematic survey for explanation-guided learning. *ACM Comput. Surv.*, 56(7),

April 2024. ISSN 0360-0300. doi: 10.1145/3644073. URL https://doi.org/10.1145/3644073.

[6] Stefano Teso, Öznur Alkan, Wolfgang Stammer, and Elizabeth Daly. Leveraging explanations in interactive machine learning: An overview. *Frontiers in Artificial Intelligence*, 6:1066049, 2023.

[7] Kushal Chauhan, Rishabh Tiwari, Jan Freyberg, Pradeep Shenoy, and Krishnamurthy Dvijotham. Interactive concept bottleneck models. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023. ISBN 978-1-57735-880-0. doi: 10.1609/aaai.v37i5.25736. URL https://doi.org/10.1609/aaai.v37i5.25736.

[8] Wolfgang Stammer, Felix Friedrich, David Steinmann, Manuel Brack, Hikaru Shindo, and Kristian Kersting. Learning by self-explaining. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=bpjU7rLjJ7.

[9] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[10] Mateo Espinosa Zarlenga, Pietro Barbiero, Gabriele Ciravegna, Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, Zohreh Shams, Frederic Precioso, Stefano Melacci, Adrian Weller, Pietro Lió, and Mateja Jamnik. Concept embedding models: Beyond the accuracy-explainability trade-off. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 21400–21413. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/867c06823281e506e8059f5c13a57f75-Paper-Conference.pdf.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

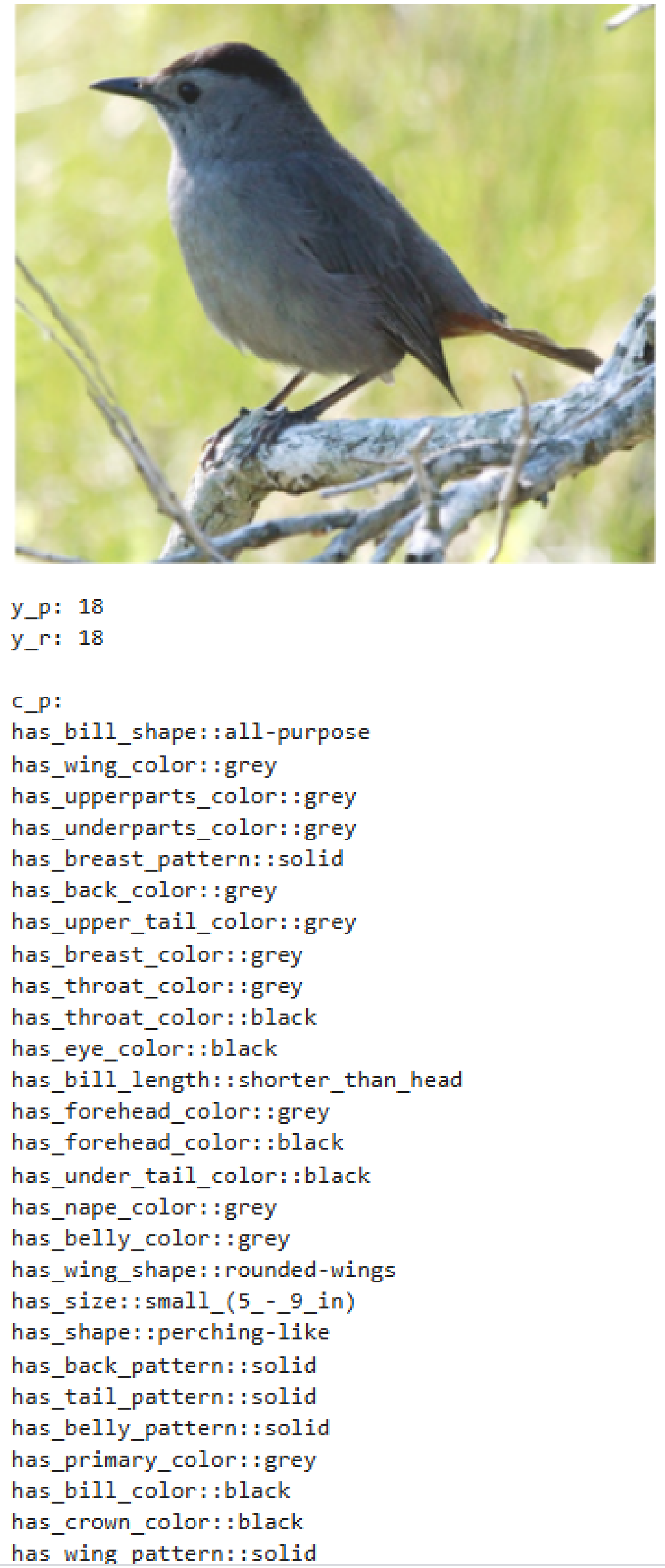
[12] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

A Examples of test set

Below are Figures 1 and 2 which show an instance of the test set predicted by the oracle (that was trained on the confounded train and non-confounded test splits) and the buggy model respectively.

B Distributions of test set

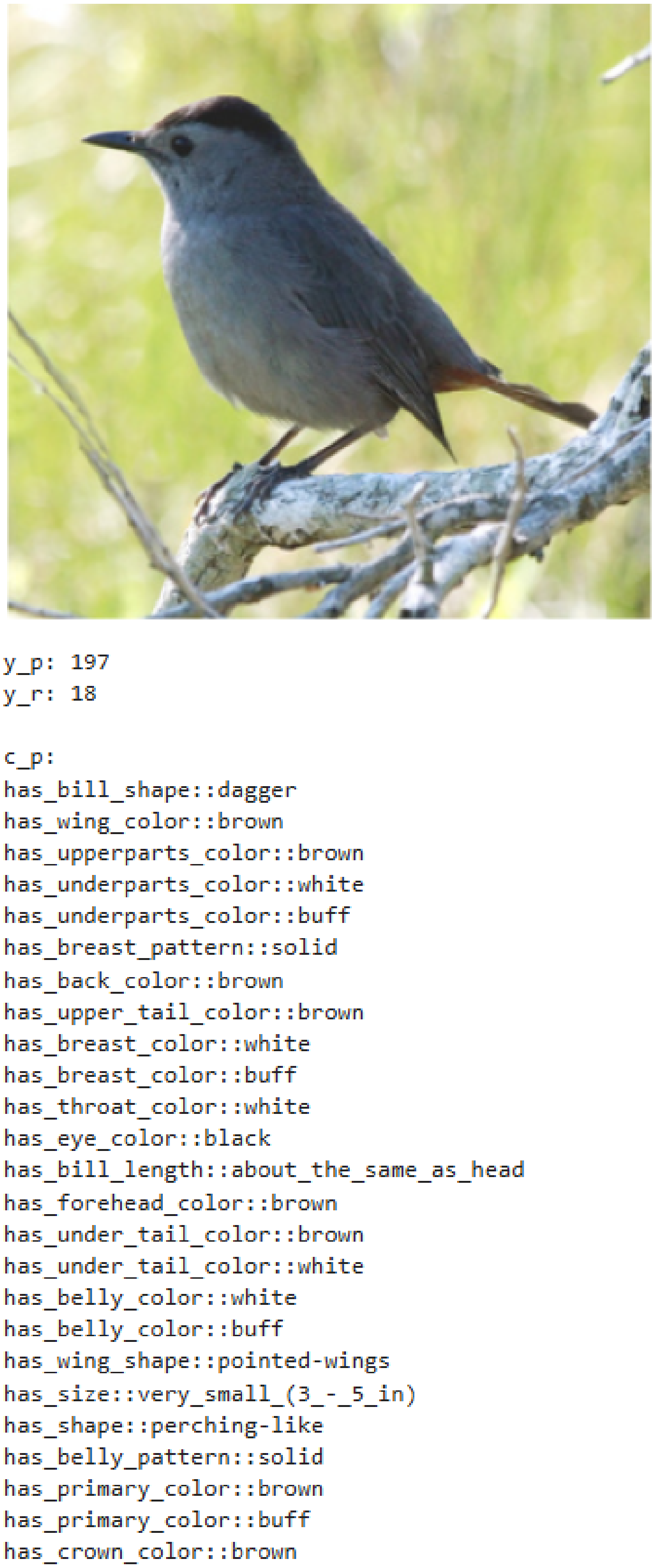
Below are four Figures. Figures 3 and 4 show the output distribution of the oracle on the test set. Figure 5 the distribution of the buggy model. Lastly, Figure 6 gives the real distribution of the test set labels.



```
y_p: 18
y_r: 18

c_p:
has_bill_shape::all-purpose
has_wing_color::grey
has_upperparts_color::grey
has_underparts_color::grey
has_breast_pattern::solid
has_back_color::grey
has_upper_tail_color::grey
has_breast_color::grey
has_throat_color::grey
has_throat_color::black
has_eye_color::black
has_bill_length::shorter_than_head
has_forehead_color::grey
has_forehead_color::black
has_under_tail_color::black
has_nape_color::grey
has_belly_color::grey
has_wing_shape::rounded-wings
has_size::small_(5_-_9_in)
has_shape::perching-like
has_back_pattern::solid
has_tail_pattern::solid
has_belly_pattern::solid
has_primary_color::grey
has_bill_color::black
has_crown_color::black
has_wing_pattern::solid
```

Figure 1: Prediction on a test instance using oracle of Section 5.1.2).



```
y_p: 197
y_r: 18

c_p:
has_bill_shape::dagger
has_wing_color::brown
has_upperparts_color::brown
has_underparts_color::white
has_underparts_color::buff
has_breast_pattern::solid
has_back_color::brown
has_upper_tail_color::brown
has_breast_color::white
has_breast_color::buff
has_throat_color::white
has_eye_color::black
has_bill_length::about_the_same_as_head
has_forehead_color::brown
has_under_tail_color::brown
has_under_tail_color::white
has_belly_color::white
has_belly_color::buff
has_wing_shape::pointed-wings
has_size::very_small_(3_-_5_in)
has_shape::perching-like
has_belly_pattern::solid
has_primary_color::brown
has_primary_color::buff
has_crown_color::brown
```

Figure 2: Prediction on a test instance using the buggy model.

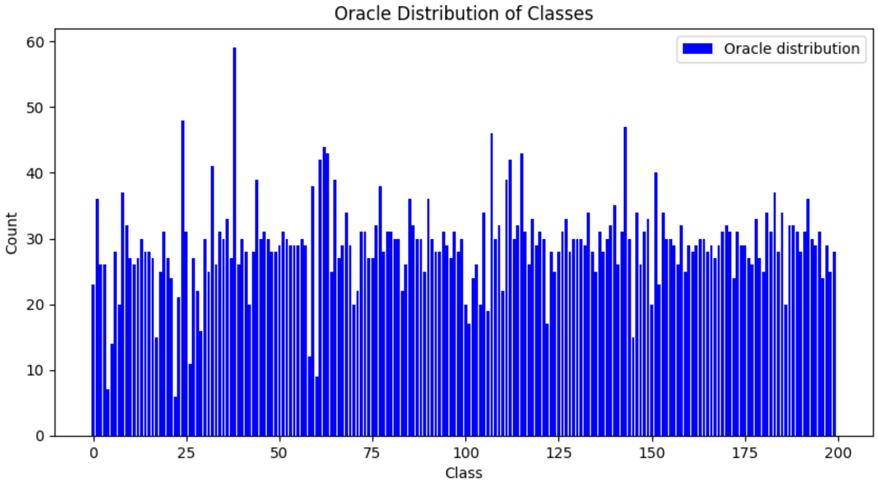


Figure 3: Oracle test data output distribution (see Section 5.1.1).

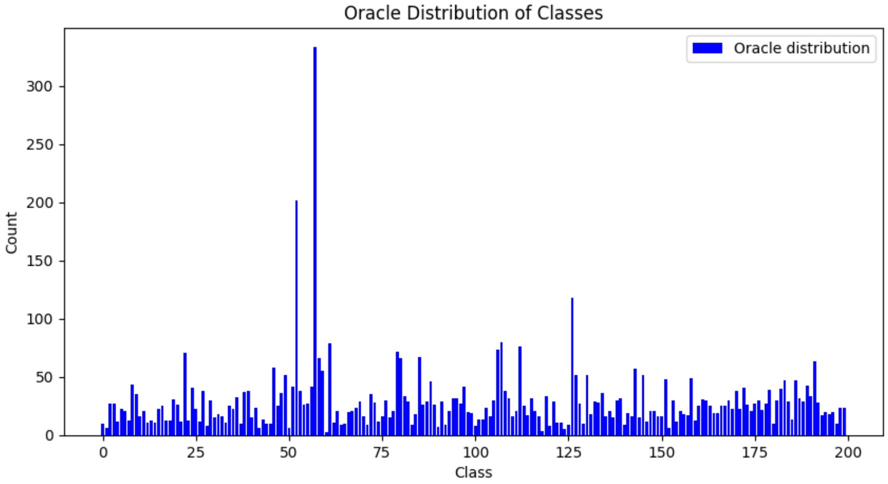


Figure 4: Oracle test data output distribution (see Section 5.1.2).

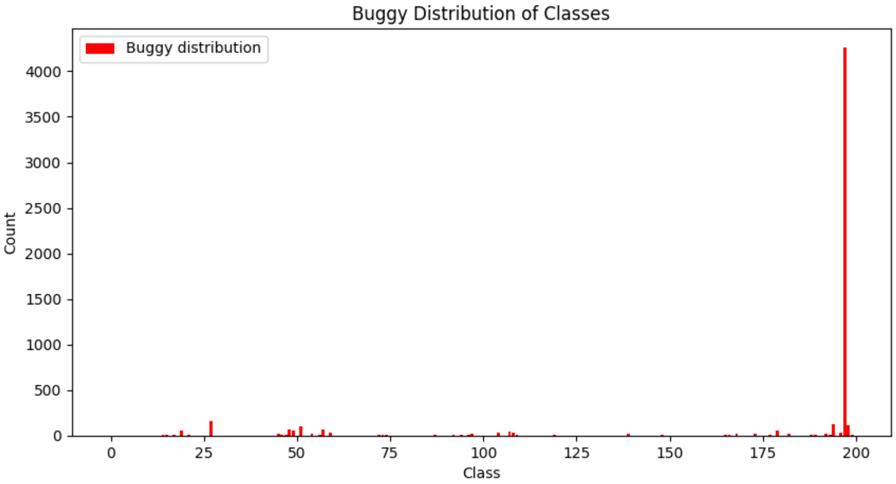


Figure 5: Buggy model test data output distribution (see Section 5.2).

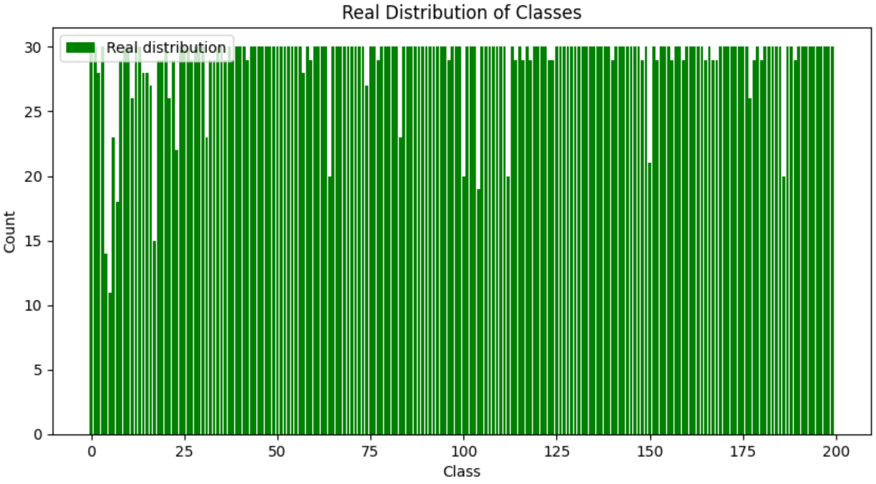


Figure 6: Real test set label distribution (see Section 2).