ERROR DISCOVERY BY CLUSTERING INFLUENCE EMBEDDINGS

Fulton Wang[†,*], **Julius Adebayo**[‡,*] **Sarah Tan**[♯], **Diego Garcia-Olano**[†], **& Narine Kokhlikyan**[†]
[†]Meta Inc., [‡]Prescient Design / Genentech, [♯]Cornell University.

ABSTRACT

We present a method for identifying groups of test examples—slices—on which a pre-trained model under-performs, a task now known as *slice discovery*. We formalize *coherence*, a requirement that erroneous predictions within returned slices should be *wrong for the same reason*, as a key property that a slice discovery method should satisfy. We then leverage influence functions (Koh & Liang, 2017) to derive a new slice discovery method, `InfEmbed`, which satisfies coherence by returning slices whose examples are influenced similarly by the training data. `InfEmbed` is computationally simple, consisting of applying K-Means clustering to a novel representation we deem *influence embeddings*. Empirically, we show `InfEmbed` outperforms current state-of-the-art methods on a slice discovery benchmark, and is effective for model debugging across several case studies.

1 INTRODUCTION

Error discovery is a longstanding challenge in machine learning (Pope, 1976; Amershi et al., 2015; Sculley et al., 2015; Chung et al., 2019; Chakarov et al., 2016; Cadamuro et al., 2016; Kearns et al., 2018; Kim et al., 2019; Zinkevich Martin, 2020). Recently, Eyuboglu et al. (2022b) formalized a type of error analysis termed the *slice discovery problem*. In the *slice discovery problem*, given a pre-trained multi-class classification model and test data, the goal is to partition the *test* data into a set of *slices*—groups of test examples—on the basis of both their features and labels. An effective slice discovery method (SDM) should satisfy two desiderata:

1. *Error surfacing*: Identify *under-performing slices*, i.e. slices with low accuracy, if they exist.
2. *Coherence*: Return slices that are *coherent* in the following sense: Erroneous predictions in a given slice should have the same *root cause*, i.e. be "wrong for the same reason".

In developing a SDM, there are two key challenges: 1) *Formalizing the coherence property*: defining the *root cause* of an error, identifying whether two errors have similar root causes, and ultimately, defining whether a slice is *coherent*; and 2) *Identifying the representation to use*, i.e., given two samples, for which a model gives the wrong prediction, the basis of comparison to assess the similarity of the cause of the error for both samples.

In this paper, we tackle those challenges and contribute the following:

1. **A new slice discovery method**: We propose `InfEmbed`, a SDM to identify under-performing slices that applies K-Means to a new representation: *influence embeddings*.
2. **Coherence & Theoretical Justification.** We formalize the *coherence property*, which requires that erroneous predictions within a slice should be "wrong for the same reason". We then define influence embeddings, and theoretically show that a procedure which clusters them achieves coherence in a sense made precise by leveraging influence functions: it returns slices whose examples are *influenced similarly by the training data.*
3. **Empirical Performance.** We show that `InfEmbed` outperforms previous state-of-the-art SDMs on the DCBench benchmark (Eyuboglu et al., 2022a). `InfEmbed` is scalable, able to recover known errors in variety of settings, and identifies coherent slices on realistic models.

[*]The two authors contributed equally. Correspondence to: fultonwang@meta.com

2 Background

Slice Discovery Problem: In the slice discovery problem, we have a pre-trained *multi-class* classification model $f(\cdot;\theta)$, where θ is the model's real-valued parameters. Given example $x \in \mathcal{X}$, some input space, $f(x;\theta) \in \mathbb{R}^C$ is the prediction for the example, where C is the number of classes, and $f(x;\theta)_c$ is the pre-softmax prediction for class c. We are also given a test dataset with N examples $\mathbf{Z} := [z_1, \dots, z_N]$, where $z_i := (x_i, y_i)$ with $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y} := [0,1]^C$. Given the model $f(\cdot;\theta)$ and test dataset $\mathbf{Z}$, the goal of the slice discovery problem is to partition the test dataset into K slices: $\{\Phi_k\}_{k=1}^K$, where $\Phi_k \subseteq [N]$, $\Phi_k \cap \Phi_{k'} = \emptyset$ for $k \neq k'$, $\cup_{k=1}^K \Phi_k = [N]$. As we will make precise shortly, each slice Φ_k should be coherent, and some slices should be under-performing slices. The model is assumed to be trained on a training dataset with N' examples, $\mathbf{Z}' := [z'_1, \dots, z'_{N'}]$ where $z'_i := (x'_i, y'_i)$ with $x'_i \in \mathcal{X}, y'_i \in \mathcal{Y}$, so that $\theta = \text{argmin}_{\theta'} \frac{1}{N'} \sum_{i=1}^{N'} L(z'_i;\theta')$. We assume loss function L is cross-entropy loss. Thus, given $z = (x,y)$ with $x \in \mathcal{X}, y = [y_1, \dots, y_C] \in \mathcal{Y}$, $L(z;\theta) = -\sum_c y_c \log p_c$, with $p = [p_1, \dots, p_C]$ and $p_c := \frac{\exp(f(x;\theta)_c)}{\sum_{c'} \exp(f(x;\theta)_{c'}))}$, i.e. $p = \text{softmax}(f(x;\theta))$.

Influence Functions: Influence functions calculate the *influence* of a given training example z' on a given test example z for a given pre-trained model, which approximates the change in a loss for a given test example z when a given training example z' is removed from the training data and the model retrained. Koh & Liang (2017) derive the aforementioned influence to be $I(z', z) := \nabla_\theta L(z';\theta)^\intercal H_\theta^{-1} \nabla_\theta L(z;\theta)$, where H_θ is the loss Hessian for the pre-trained model: $H_\theta := 1/n \sum_{i=1}^n \nabla_\theta^2 L(z;\theta)$. The main challenge in computing influence is that it is impractical to explicitly form H_θ unless the model is small, or if one only considers parameters in a few layers. Schioppa et al. (2022) address this problem by forming a low-rank approximation of H_θ^{-1} via a procedure that does not explicitly form H_θ. In brief, they run the Arnoldi iteration Trefethen & Bau III (1997) for P iterations to get a P-dimensional Krylov subspace for H_θ, which requires only P Hessian-vector products. Then, they find a rank-D approximation of the restriction of H_θ to the Krylov subspace, a small $P \times P$ matrix, via eigendecomposition. Their procedure is summarized by an algorithm FactorHessian (Appendix Section C.1 has details), which returns factors of a low-rank approximation of H_θ^{-1}

$$M, \lambda = \text{FactorHessian}(\mathbf{Z}', \Theta, P, D) \text{ where } \hat{H}_\theta^{-1} := M\lambda^{-1}M^\intercal \approx H_\theta^{-1}, \tag{1}$$

$M \in \mathbb{R}^{|\theta| \times D}$, $\lambda \in \mathbb{R}^{D \times D}$ is an *diagonal* matrix, $|\theta|$ is the parameter count, $\hat{H}_\theta^{-1}$ approximates H_θ^{-1}, D is the rank of the approximation, P is the Arnoldi dimension, i.e. number of Arnoldi iterations, and here and everywhere, *configuration* $\Theta := (L, f, \theta)$ denotes the loss function, model, parameters. $\hat{H}_\theta^{-1}$ is then used to define the *practical influence* of training example z' on test example z:

$$\hat{I}(z', z) := \nabla_\theta L(z';\theta)^\intercal \hat{H}_\theta^{-1} \nabla_\theta L(z;\theta). \tag{2}$$

3 Error Discovery By Clustering Influence Embeddings

3.1 Problem Formulation

We would like to partition the test data into slices such that predictions for examples in the same slice have similar root causes as quantified by influence functions. Therefore, we propose to represent the root cause of the prediction for a test example z with its *influence explanation* $E(z)$*:*

$$E(z) := \{\hat{I}(z'_j, z)\}_{j=1}^{N'}, \tag{3}$$

which is the vector containing the practical influence of every training example on the test example. Thus, test examples with similar influence explanations are "influenced similarly by the training data". We then want each slice be *coherent* in the following sense: the Euclidean distance between influence explanations of examples in the same slice tends to be low.

Therefore, we solve the slice discovery problem described in Section 2 via the following formulation—return a partition of the test dataset into slices, $\{\Phi_k\}_{k=1}^K$, that minimizes the total Euclidean distance between influence explanations of examples in the same slice. Formally, we seek:

$$\text{argmin}_{\{\Phi_k\}_{k=1}^K} \sum_k \sum_{i,i' \in \Phi_k} ||E(z_i) - E(z_{i'})||^2. \tag{4}$$

Algorithm 2 Our SDM, `InfEmbed`, applies K-Means to influence embeddings of test examples.

```
1: procedure GetEmbeddings($\mathbf{Z}, \mathbf{Z}', \Theta, P, D$)
2:     Outputs: $D$-dimensional embeddings for test data $\mathbf{Z}$
3:     $M, \lambda \leftarrow \text{FactorHessian}(\mathbf{Z}', \Theta, P, D)$                ▷ see Equation 1
4:     $\mu_i \leftarrow \lambda^{-1/2} M^\intercal \nabla_\theta L(z_i;\theta)$ for $i = 1, \ldots, N$            ▷ see Equation 5
5:     $\boldsymbol{\mu} \leftarrow [\mu_1, \ldots, \mu_N]$
6:     Return: $\boldsymbol{\mu}$
7: end procedure
8: procedure InfEmbed(K, $\mathbf{Z}, \mathbf{Z}', \Theta, P, D$)
9:     Inputs: number of slices $K$, training dataset $\mathbf{Z}'$, test dataset $\mathbf{Z}$, configuration $\Theta$, Arnoldi
       dimension $P$, influence embedding dimension $D$
10:    Outputs: partition of test dataset into slices, $\Phi$
11:    $\boldsymbol{\mu} \leftarrow \text{GetEmbeddings}(\mathbf{Z}, \mathbf{Z}', \Theta, P, D)$
12:    $[r_1, \ldots, r_N] \leftarrow \text{K-Means}(\boldsymbol{\mu}, K)$     ▷ compute cluster assignments for all $N$ examples
13:    Return: $\{\{i \in [N] : r_i = k\}$ for $k \in [K]\}$      ▷ convert cluster assignments to a partition
14: end procedure
```

3.2 Influence Embeddings

To form coherent slices, a naive approach would be to form the influence explanation of each test example, and apply K-Means clustering with Euclidean distance to them. However, influence explanations are high-dimensional, with dimensionality equal to the size of the training data, N'. Instead, we will leverage *influence embeddings*: vectors such that the practical influence of a training example on a test example is the dot-product of their respective influence embeddings. Looking at Equations 1 and 2, we see the influence embedding $\mu(z)$ of an example z must be defined as

$$\mu(z) := \lambda^{-1/2} M^\intercal \nabla_\theta L(z;\theta) \tag{5}$$

and M, λ are as defined in Section 2, because for any training example z' and test example z, $\hat{I}(z', z) = \mu(z')^\intercal \mu(z)$. Procedure `GetEmbeddings` of Algorithm 1 finds D-dimensional influence embeddings for the test dataset $\mathbf{Z}$, which has runtime *linear* in the Arnoldi dimension P. The rank D of the factors from `FactorHessian` determines the dimension of the influence embeddings.

Influence embeddings satisfy a critical property - that if two examples have similar influence embeddings, they also tend to have similar influence explanations. This is formalized by the following lemma, whose proof follows from the Cauchy-Schwartz inequality, and that influence is the dot-product of influence embeddings (see Appendix Section C.2)

Lemma 1: There is a constant $C > 0$ such that for any test examples z_i, z_j, $||E(z_i) - E(z_j)||^2 \leq C||\mu(z_i) - \mu(z_j))||^2$.

3.3 Discovering Problematic Slices with InfEmbed

We will solve the formulation of Equation 4 via a simple procedure which applies K-Means with Euclidean distance to the influence *embeddings* of the test dataset, $\boldsymbol{\mu} := [\mu(z_1), \ldots, \mu(z_N)]$. This procedure is justified as follows: Using Lemma 1, given any partition $\{\Phi_k\}$, we know

$$\textstyle\sum_k \sum_{i,i' \in \Phi_k} ||E(z_i) - E(z_{i'})||^2 \leq C \sum_k \sum_{i,i' \in \Phi_k} ||\mu(z_i) - \mu(z_{i'})||^2. \tag{6}$$

However, the quantity in the right of Equation 6 is the *surrogate* objective minimized by this procedure—it is the K-Means objective applied to influence embeddings, scaled by $C > 0$ which does not matter. The quantity in the left of Equation 6 is the *actual* objective which the formulation of Equation 4 minimizes. Therefore, the procedure is minimizing a surrogate objective which upper bounds the actual objective we care about. Put another way, by applying K-Means to influence embeddings, examples within the same slice will not only have similar influence embeddings, but also similar influence explanations as desired, by Lemma 1. Algorithm 1 describes `InfEmbed`.

Please see the Appendix for more details of `InfEmbed`: Section C.4 describes `InfEmbed-Rule`, an extension that instead of finding K slices, finds slices that have accuracy below a given threshold, while having size above a given threshold. Section C.5 derives why slices from `InfEmbed` (and

SDM Approach	Rare	Correlation	Noisy Label
Natural Images			
Domino	0.4	0.45	0.6
Inf-Embed	0.65	0.55	0.67
Medical Images			
Domino	0.39	0.6	0.58
Inf-Embed	0.57	0.62	0.73
Medical Time Series			
Domino	0.6	0.55	0.9
Inf-Embed	0.64	0.65	0.81

Table 1: `InfEmbed` generally outperforms Domino across 3 input types and 3 tasks on the `dcbench` benchmark in terms of precision (higher is better).

`InfEmbed-Rule`) tend to have *label homogeniety* - homogeneous in terms of the label and predictions, a desirable property of slices. Section C.6 describes how to explain slices via the opponents of a slice - the training examples whose presence increases / hurts the total loss over a slice the most.

4 Experiments

Here, we perform a quantitative evaluation of `InfEmbed` on the `dcbench` benchmark, and also perform several case studies showing `InfEmbed`'s usefulness. Please see Appendix Section H on a case study applying `InfEmbed` to a small dataset where additional metadata can help explain slices. For scalability, to compute influence embeddings, at times we will only consider gradients in some layers of the model when calculating the gradient $\nabla_\theta L(z;\theta)$ and inverse-Hessian factors $M, \lambda^{1/2}$ in Equation 5. Furthermore, following Schioppa et al. (2022), we do not compute those factors using the entire training data, i.e. we pass in a subset of the training data to `FactorHessian`. For all experiments, we use Arnoldi dimention $P = 500$, and influence embedding dimension $D = 100$, unless noted otherwise.

4.1 InfEmbed on dcbench

Overview & Experimental Procedure: `dcbench` (Eyuboglu et al., 2022a) provides 1235 pre-trained models that are derived from real-world data, where the training data is manipulated in 3 ways to lead to test-time errors. This gives 3 tasks - discovering slices whose errors are due to the following manipulations: 1) "rare" (examples in a given class are down-sampled), 2) "correlation" (a spurious correlation is introduced), and 3) "noisy label" (examples in a given class have noisy labels). Since the error causing slice is known a priori, `dcbench` can serve as a way to assess a new SDM. The underlying datasets for those tasks include 3 input types: natural images, medical images, and medical time-series. Eyuboglu et al. (2022a) also introduced Domino, the SDM that is currently the best-performing SDM on `dcbench`, which uses multi-modal CLIP embeddings as input to an error-aware mixture model. Following Eyuboglu et al. (2022a), we compare `InfEmbed` to Domino using the precision-at-k measure, which measures the proportion of the top k ($k = 10$) elements in the discovered slice that are in the ground truth slice. We evaluate in the setting where errors are due to a trained model. Appendix Section D contains additional setup details.

Results: Table 1 compares `InfEmbed` to Domino across the 3 tasks and 3 input types. `InfEmbed` always beats Domino except on the "noisy" task for the EEG input type.

4.2 Imagenet & High Dimensional Settings

Overview & Experimental Procedure: Here, we identify problematic slices in a "natural" dataset - the test split of Imagenet [1] (Deng et al., 2009), containing 5000 examples. This image-classification problem is a 1000-class problem, and the analyzed model is a pre-trained Resnet-18 model achieving 69% test accuracy. To compute influence embeddings, we consider gradients in the "fc" and "layer4" layers, which contain 8.9M parameters. We use `ImfEmbed-Rule` to find slices with at most 40% accuracy, and at least 25 examples. To diagnose the root-cause of errors for a slice, we examine the slice's opponents, following Section C.6.

[1] `https://www.image-net.org/`

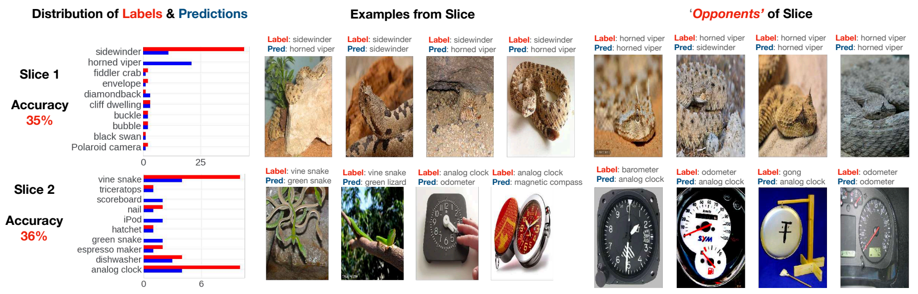

Figure 1: For 2 slices from applying `InfEmbed-Rule` to Imagenet, we show the distributions of labels (red) and predictions (blue) (left), examples from it (middle), and opponents of the slice (right).

Results: We find 25 slices, comprising a total of 954 examples, whose overall accuracy is 34.9%. Figure G displays 2 of these slices (See Section E for additional examples). For each slice, we show the distribution of predicted and true labels, the 4 examples nearest the slice center in influence embedding space, and the 4 strongest opponents of the slice. First, we see that the slices have the label homogeneity property as explained in Section C.5—the predicted and true labels typically come from a small number of classes. In the first slice, the true label is mostly "sidewinder", and often predicted to be "horned viper". The slice's strongest opponents are horned vipers, hard examples of another class, which the model considers to be similar to sidewinders. In the second slice, we observe *two* dominant labels. Here, vine snakes and analog clocks are often mis-predicted. A priori, one would not have guess the predictions, for these seemingly unrelated classes would be wrong for the same reasons. However, looking at the opponents, we see all of them have a clock hand, which turns out to be similar to a snake. Also, because the opponents are of a variety of classes, the mis-predictions are also of a variety of classes.

4.3 AGNews & Mis-labelled Data

Overview & Experimental Procedure: We identify problematic slices in another "natural" dataset - the test split of AGNews (Zhang et al., 2015), comprising 7600 examples. This text-classification problem is a 4-class problem, and the pre-trained model is a BERT base model fine-tuned on the training dataset [2], achieving 93% test accuracy, making 475 errors. We use `ImfEmbed` with $K = 25$, and examine slices with at most 10% accuracy, and at least 10 examples.

Results: We find 9 such slices, comprising a total of 452 examples, whose overall accuracy is only 3%. Table 2 shows 2 slices, both of which have label homogeneity. In the first, the model achieves 0% accuracy, systematically mis-predicting "business" examples to be "world", perhaps predicting any text with a country name in it to be "world". In the second slice, most labels are "science/technology", and most predictions are "business". However, they may in fact be mis-labeled, as they are about businesses and their technology. This shows that `InfEmbed` can also identify *mis-labeled data*.

4.4 Detecting Spurious Signals in BoneAge Classification

Overview & Experimental Procedure: We now artificially inject a signal that spuriously correlates with the label in training data, and see if `InfEmbed-rule` can detect the resulting test errors as well as the spurious correlation. Here, the classification problem is bone-age classification [3] - to predict which of 5 age groups a person is in given their x-ray. Following the experimental protocol of Zhou et al. (2022) and Adebayo et al. (2021), we consider 3 possible signals that can be added to an x-ray, shown in Figure 4 of the Appendix - adding a tag, stripe, or blur. For a given signal, we manipulate the bone-age training dataset by injecting the spurious signal into all *training* examples from the "mid-pub" class (short for mid-puberty). Importantly, we leave the test dataset untouched, so that we expect a model trained on the manipulated training data to err on "mid-pub" test examples. This setup mimics a realistic scenario where x-ray training data comes from a hospital where one class is over-represented, and also contains a spurious signal (i.e. x-rays from the hospital might have the hospital's tag). We then train a resnet50 on the manipulated training data, and apply

[2] https://huggingface.co/fabriceyhc/bert-base-uncased-ag_news
[3] https://www.kaggle.com/datasets/kmader/rsna-bone-age

Slice with 45 examples, 0% accuracy. Predictions: 100% "world", labels: 100% "business"
1. *"British grocer Tesco sees group sales rise 12.0-percent. Britain's biggest supermarket chain ..."*
2. *"Nigerian Senate approves $1.5 bln claim on Shell LAGOS - Nigeria's Senate has passed ... "*
3. *"Cocoa farmers issue strike threat. Unions are threatening a general strike in the Ivory Coast ..."*

Slice with 139 examples, 3% accuracy. Predictions: 93% "business", labels: 94% "sci/tech"
1.*"Google Unveils Desktop Search, Takes on Microsoft. Google Inc. rolled out a version of its ..."*
2.*"PalmOne to play with Windows Mobile? Rumors of Treo's using a Microsoft operating system ..."*
3.*"Intel Posts Higher Profit, Sales. Computer-chip maker Intel Corp. said yesterday that earnings ..."*

Table 2: Representative examples for 2 slices obtained using `InfEmbed` on AGnews. In the 1st slice, the model mis-predicts business text containing country names to be "world". The 2nd slice appears to be mis-labelled, showing `InfEmbed` can detect mis-labeled data.

`InfEmbed-Rule` on the untouched test data to discovered slices with at least 25 examples and at most 40% accuracy (same as for the Imagenet analysis). We repeat this manipulate-train-slice discovery procedure for 3 runs: once for each possible signal.

Results: For all 3 runs / signals, `InfEmbed-Rule` returned 2 slices satisfying the rule. In Figure 5, for each signal, we show the distribution of labels and predictions for the most problematic slice (left column) and the same distribution for that slice's top-50 opponents (right column). We see that for each signal, the labels in the most problematic slice are of the "mid-pub" class, showing that `InfEmbed-Rule` is able to detect test errors due to the injected spurious training signal. Furthermore, for each run, around 80% of the slice's opponents are training examples with the spurious training signal, i.e. examples with the "mid-pub" label. Together, this shows `InfEmbed-Rule` can not only detect slices that the model errs on because of a spurious training signal, but also identify training examples with the spurious training signal by using slice opponents.

5 Related Work

To the best of our knowledge, `InfEmbed` is the first SDM based on influence functions. More broadly, it represents the first use of influence functions for clustering and global explainability. Pruthi et al. (2020) also define influence embeddings, but use a different definition and also use it for a different purpose - accelerating the retrieval of an example's most influential training examples. Crucially, their embeddings were high-dimensional (exceeding the number of model parameters) and ill-suited for clustering, lacking any dimension reduction procedure, aside from random projections.

Previous SDM use a variety of representations as input into a partitioning procedure. Some rely on *external* feature extractors, using PCA applied to pre-trained CLIP embeddings Eyuboglu et al. (2022b), or features from a robustly pre-trained image model Singla et al. (2021). This external reliance limits their generalizability—what if they were applied to a domain very different from the one CLIP was trained on, or wanted to consider a new modality, like audio or graphs? For example, Kim et al. (2023) show that CLIP embeddings cannot generate accurate captions for domains like X-rays and air-view images. Other SDM's use the pre-trained model's last-layer representations d'Eon et al. (2022), or transform them via SCVIS Plumb et al. (2022) or UMAP Sohoni et al. (2020). While they are performant, it is less clear *why* this is. In contrast, the influence embeddings `InfEmbed` uses do not rely on external feature extractors, and are theoretically motivated through the influence function-based definition of coherency. Appendix Section A contains additional related work.

6 Conclusion

We present a method, `InfEmbed`, that addresses the slice discovery problem. Our proposed solution departs from previous work in that it both identifies slices—group of data points— on which a model under-performs, and that satisfy a certain *coherence* property. We formalize coherence—predictions being wrong for the same reasons within a slice—as a key property that all slice discovery methods should satisfy. In particular, we leverage influence functions (Koh & Liang, 2017) to define coherent slices as ones whose examples are influenced similarly by the training data, and then develop a procedure to return coherent slices based on clustering a representation we call influence embeddings. We then demonstrate the effectiveness of `InfEmbed` both quantitatively and through case studies.

REFERENCES

Julius Adebayo, Michael Muelly, Harold Abelson, and Been Kim. Post hoc explanations may be ineffective for detecting unknown spurious correlation. In *International Conference on Learning Representations*, 2021.

Ahmed Alqaraawi, Martin Schuessler, Philipp Weiß, Enrico Costanza, and Nadia Berthouze. Evaluating saliency map explanations for convolutional neural networks: a user study. In *Proceedings of the 25th International Conference on Intelligent User Interfaces*, pp. 275–285, 2020.

Saleema Amershi, Max Chickering, Steven M Drucker, Bongshin Lee, Patrice Simard, and Jina Suh. Modeltracker: Redesigning performance analysis tools for machine learning. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pp. 337–346, 2015.

Aparna Balagopalan, Haoran Zhang, Kimia Hamidieh, Thomas Hartvigsen, Frank Rudzicz, and Marzyeh Ghassemi. The road to explainability is paved with bias: Measuring the fairness of explanations. *arXiv preprint arXiv:2205.03295*, 2022.

Elnaz Barshan, Marc-Etienne Brunet, and Gintare Karolina Dziugaite. Relatif: Identifying explanatory training samples via relative influence. In *International Conference on Artificial Intelligence and Statistics*, pp. 1899–1909. PMLR, 2020.

Tolga Bolukbasi, Adam Pearce, Ann Yuan, Andy Coenen, Emily Reif, Fernanda Viégas, and Martin Wattenberg. An interpretability illusion for bert. *arXiv preprint arXiv:2104.07143*, 2021.

Gabriel Cadamuro, Ran Gilad-Bachrach, and Xiaojin Zhu. Debugging machine learning models. In *ICML Workshop on Reliable Machine Learning in the Wild*, volume 103, 2016.

Aleksandar Chakarov, Aditya Nori, Sriram Rajamani, Shayak Sen, and Deepak Vijaykeerthy. Debugging machine learning tasks. *arXiv preprint arXiv:1603.07292*, 2016.

Valerie Chen, Jeffrey Li, Joon Sik Kim, Gregory Plumb, and Ameet Talwalkar. Towards connecting use cases and methods in interpretable machine learning. *arXiv preprint arXiv:2103.06254*, 2021.

Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, Ki Hyun Tae, and Steven Euijong Whang. Automated data slicing for model validation: A big data-ai integration approach. *IEEE Transactions on Knowledge and Data Engineering*, 32(12):2284–2296, 2019.

Joseph Paul Cohen, Paul Morrison, Lan Dao, Karsten Roth, Tim Q Duong, and Marzyeh Ghassemi. Covid-19 image data collection: Prospective predictions are the future. *Journal of Machine Learning for Biomedical Imaging*, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Greg d'Eon, Jason d'Eon, James R Wright, and Kevin Leyton-Brown. The spotlight: A general method for discovering systematic errors in deep learning models. In *2022 ACM Conference on Fairness, Accountability, and Transparency*, pp. 1962–1981, 2022.

Sabri Eyuboglu, Bojan Karlaš, Christopher Ré, Ce Zhang, and James Zou. dcbench: a benchmark for data-centric ai systems. In *Proceedings of the Sixth Workshop on Data Management for End-To-End Machine Learning*, pp. 1–4, 2022a.

Sabri Eyuboglu, Maya Varma, Khaled Saab, Jean-Benoit Delbrouck, Christopher Lee-Messer, Jared Dunnmon, James Zou, and Christopher Ré. Domino: Discovering systematic errors with cross-modal embeddings. *arXiv preprint arXiv:2203.14960*, 2022b.

Marzyeh Ghassemi, Luke Oakden-Rayner, and Andrew L Beam. The false hope of current approaches to explainable artificial intelligence in health care. *The Lancet Digital Health*, 3(11):e745–e750, 2021.

Safwan S Halabi, Luciano M Prevedello, Jayashree Kalpathy-Cramer, Artem B Mamonov, Alexander Bilbily, Mark Cicero, Ian Pan, Lucas Araújo Pereira, Rafael Teixeira Sousa, Nitamar Abdala, et al. The rsna pediatric bone age machine learning challenge. *Radiology*, 290(2):498–503, 2019.

Wenyue Hua, Lifeng Jin, Linfeng Song, Haitao Mi, Yongfeng Zhang, and Dong Yu. Discover, explanation, improvement: Automatic slice detection framework for natural language processing. *arXiv preprint arXiv:2211.04476*, 2022.

Saachi Jain, Hannah Lawrence, Ankur Moitra, and Aleksander Madry. Distilling model failures as directions in latent space. *arXiv preprint arXiv:2206.14754*, 2022.

Michael Kearns, Seth Neel, Aaron Roth, and Zhiwei Steven Wu. Preventing fairness gerrymandering: Auditing and learning for subgroup fairness. In *International Conference on Machine Learning*, pp. 2564–2572. PMLR, 2018.

Michael P Kim, Amirata Ghorbani, and James Zou. Multiaccuracy: Black-box post-processing for fairness in classification. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 247–254, 2019.

Younghyun Kim, Sangwoo Mo, Minkyu Kim, Kyungmin Lee, Jaeho Lee, and Jinwoo Shin. Explaining visual biases as words by generating captions. *arXiv preprint arXiv:2301.11104*, 2023.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pp. 1885–1894. PMLR, 2017.

Zhifeng Kong and Kamalika Chaudhuri. Understanding instance-based interpretability of variational auto-encoders. *Advances in Neural Information Processing Systems*, 34:2400–2412, 2021.

Himabindu Lakkaraju, Ece Kamar, Rich Caruana, and Eric Horvitz. Identifying unknown unknowns in the open world: Representations and policies for guided exploration. In *Thirty-first aaai conference on artificial intelligence*, 2017.

Sebastian Lapuschkin, Stephan Wäldchen, Alexander Binder, Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Unmasking clever hans predictors and assessing what machines really learn. *Nature communications*, 10(1):1–8, 2019.

Donghoon Lee, Hyunsin Park, Trung Pham, and Chang D Yoo. Learning augmentation network via influence functions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10961–10970, 2020.

Weitang Liu, Xiaoyun Wang, John Owens, and Yixuan Li. Energy-based out-of-distribution detection. *Advances in Neural Information Processing Systems*, 33:21464–21475, 2020.

Zhuoming Liu, Hao Ding, Huaping Zhong, Weijia Li, Jifeng Dai, and Conghui He. Influence selection for active learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9274–9283, 2021.

Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2(11):e7, 2017.

Gregory Plumb, Nari Johnson, Ángel Alexander Cabrera, Marco Tulio Ribeiro, and Ameet Talwalkar. Evaluating systemic error detection methods using synthetic images. *arXiv preprint arXiv:2207.04104*, 2022.

Allen J Pope. *The statistics of residuals and the detection of outliers*. US Department of Commerce, National Oceanic and Atmospheric Administration . . . , 1976.

Forough Poursabzi-Sangdeh, Daniel G Goldstein, Jake M Hofman, Jennifer Wortman Vaughan, and Hanna Wallach. Manipulating and measuring model interpretability. *arXiv preprint arXiv:1802.07810*, 2018.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33: 19920–19930, 2020.

Nazneen Rajani, Weixin Liang, Lingjiao Chen, Meg Mitchell, and James Zou. Seal: Interactive tool for systematic error analysis and labeling. *arXiv preprint arXiv:2210.05839*, 2022.

Laura Rieger, Chandan Singh, W James Murdoch, and Bin Yu. Interpretations are useful: penalizing explanations to align neural networks with prior knowledge. *International Conference on Machine Learning*, 2020.

Andrea Schioppa, Polina Zablotskaia, David Vilar, and Artem Sokolov. Scaling up influence functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 8179–8186, 2022.

Peter Schulam and Suchi Saria. Can you trust this prediction? auditing pointwise reliability after learning. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1022–1031. PMLR, 2019.

David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In *Advances in neural information processing systems*, pp. 2503–2511, 2015.

Berfin Simsek, Melissa Hall, and Levent Sagun. Understanding out-of-distribution accuracies through quantifying difficulty of test samples. *arXiv preprint arXiv:2203.15100*, 2022.

Sahil Singla, Besmira Nushi, Shital Shah, Ece Kamar, and Eric Horvitz. Understanding failures of deep networks via robust feature extraction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12853–12862, 2021.

Nimit Sohoni, Jared Dunnmon, Geoffrey Angus, Albert Gu, and Christopher Ré. No subclass left behind: Fine-grained robustness in coarse-grained classification problems. *Advances in Neural Information Processing Systems*, 33:19339–19352, 2020.

Stefano Teso, Andrea Bontempelli, Fausto Giunchiglia, and Andrea Passerini. Interactive label cleaning with example-based explanations. *Advances in Neural Information Processing Systems*, 34:12966–12977, 2021.

Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

Olivia Wiles, Isabela Albuquerque, and Sven Gowal. Discovering bugs in vision models using off-the-shelf image generation and captioning. *arXiv preprint arXiv:2208.08831*, 2022.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/250cf8b51c773f3f8dc8b4be867a9a02-Paper.pdf.

Yilun Zhou, Serena Booth, Marco Tulio Ribeiro, and Julie Shah. Do feature attribution methods correctly attribute features? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 9623–9633, 2022.

Roland S Zimmermann, Judy Borowski, Robert Geirhos, Matthias Bethge, Thomas Wallis, and Wieland Brendel. How well do feature visualizations support causal understanding of cnn activations? *Advances in Neural Information Processing Systems*, 34:11730–11744, 2021.

Zinkevich Martin. Rules of Machine Learning: Best Practices for ML Engineering. http://martin.zinkevich.org/rules_of_ml/rules_of_ml.pdf, 2020. Online; accessed 10 January 2020.

A Additional Related Work

B Related Work

Understanding the errors of trained model, typically via diagnosis tools, is a long-standing problem in statistics and machine learning. The issue has taken on renewed importance in machine learning especially given the dominance of overparametrized deep nets across a variety domains and data modalities. While error analysis, more generally, has been studied extensively even within the deep learning literature, we focus our discussions on the recently formalized *slice discovery problem*—as termed by Eyuboglu et al. (2022b).

Other methods solve problems related to, but not the same as slice discovery. Rajani et al. (2022); Wiles et al. (2022); Hua et al. (2022) leverages the aforementioned SDM's to create *interactive* systems for discovering slices, focusing on improving the user interface and how to explain slices. In contrast, we focus on solving the core slice discovery problem. All these methods use last-layer representations along with K-Means or an "error-aware" mixture model Hua et al. (2022). One past work Jain et al. (2022) globally *ranks* examples to identify examples in a *single* failure mode, but being unable to output multiple slices, is not a SDM. Numerous works also rank examples using scores capturing out-of-distribution Liu et al. (2020), unreliable Schulam & Saria (2019), or confusing Simsek et al. (2022) examples. Complementary to SDM's are methods for identifying low-performance subgroups when tabular metadata is available Lakkaraju et al. (2017).

Influence Functions: Influence functions Koh & Liang (2017) have been used to calculate influences, which can be used for ranking training examples for local explainability Barshan et al. (2020) for both supervised and unsupervised models Kong & Chaudhuri (2021), identifying mis-labeled data interactively Teso et al. (2021), data augmentation Lee et al. (2020), active learning Liu et al. (2021), quantifying reliability Schulam & Saria (2019), and identifying mis-labeled data Pruthi et al. (2020). However, influence functions have not been used for slice discovery or global explainability.

Interpretability and Post-hoc Explanations: Despite initial evidence that explanations might be useful for detecting that a model is reliant on spurious signals (Lapuschkin et al., 2019; Rieger et al., 2020), a different line of work directly counters this evidence. Zimmermann et al. (2021) showed that feature visualizations (Olah et al., 2017) are not more effective than dataset examples at improving a human's understanding of the features that highly activate a DNN's intermediate neuron. Increasing evidence demonstrates that current post hoc explanation approaches might be ineffective for model debugging in practice (Chen et al., 2021; Alqaraawi et al., 2020; Ghassemi et al., 2021; Balagopalan et al., 2022; Poursabzi-Sangdeh et al., 2018; Bolukbasi et al., 2021). In a promising demonstration, Lapuschkin et al. (2019) apply a clustering procedure to the LRP saliency masks derived from a trained model. In the application, the clusters that emerge are able to separate groups of inputs where, presumably, the model relies on different features for its output decision. This work differs from that in a key way: Lapuschkin et al. (2019) demonstration is to seek understanding of the model behavior and not to perform slice discovery. There is no reason why a low performing cluster should emerge from such clustering procedure.

C Additional Background and Details on InfEmbed

C.1 Practical Low-Rank Influence Function

The main challenge in computing influence is that it is impractical to explicitly form the H_θ needed to compute influence in Equation **??** unless the model is small, or if one only considers parameters in a few layers. Schioppa et al. (2022) address this problem by forming a low-rank approximation of H_θ^{-1} without explicitly forming H_θ. Their low-rank implementation first applied the Arnoldi iteration Trefethen & Bau III (1997) to compute an orthonormal basis $(q_1, \ldots, q_D)$ for the P-dimensional Arnoldi subspace $(b, H_\theta b, \ldots, H_\theta^{P-1})$, as well as the restriction R of H_θ to the subspace, so that $H_\theta = QRQ^\intercal$, where $Q := [q_1, \ldots, q_P] \in \mathbb{R}^{|\theta| \times P}$ and $R \in \mathbb{R}^{P \times P}$. They choose P to be 200 in their experiments. Crucially, it is computationally feasible to run the Arnoldi iteration even on large models, because it only requires access to H_θ through Hessian-vector products, not through its explicit formation. Then, leveraging the fact that the Arnoldi subspace of a matrix tends to contain its top eigenvectors, they approximate $R \approx V \lambda V^\intercal$ and $R^{-1} \approx V \lambda^{-1} V^\intercal$, where $V = [v_1, \ldots, v_D] \in \mathbb{R}^{P \times D}$,

$v_1, \ldots, v_D$ are the top D eigenvectors of R, and $\lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_D)$, where $\lambda_1, \ldots, \lambda_D$ are the corresponding eigenvalues. They choose D to be around 50 in their experiments. Crucially, it is computationally feasible to find the top eigenvectors and eigenvalues via eigen-decomposition on an explicitly-formed R, because R is of size $P \times P$ with $P = 200$. Finally, they form a low-rank approximation of H_θ^{-1},

$$\hat{H}_\theta^{-1} := M\lambda^{-1}M^\intercal, \text{ where } M = QV \tag{7}$$

and use it to define the *practical influence* of training example z' on test example z:

$$\hat{I}(z', z) := \nabla_\theta L(z'; \theta)^\intercal \hat{H}_\theta^{-1} \nabla_\theta L(z; \theta). \tag{8}$$

Algorithm C.1 outlines finding the factors needed for $\hat{H}_\theta^{-1}$. Note that for brevity of notation, throughout we will let *configuration* $\Theta := (L, f, \theta)$ denote the loss function, model, and parameters.

Algorithm 3 Finding low-rank factors of Hessian

```
procedure FACTORHESSIAN($\mathbf{Z}'$, $\Theta$, $P$, $D$)
    Inputs: training data $\mathbf{Z}'$, configuration $\Theta$, Arnoldi dimension $P$, rank $D$
    $H_\theta \leftarrow \sum_{i=1}^{N'} \nabla_\theta^2 L({z'}_i; \theta)$        ▷ implicitly define HVP
    Run Arnoldi iteration on $H_\theta$ for $P$ iterations to get $Q \in \mathbb{R}^{|\theta| \times P}$, $R \in \mathbb{R}^{P \times P}$
    $V, \lambda \leftarrow$ top-$D$ eigenvectors / values of $R$ via SVD
    $M \leftarrow QV$
    Return: $M, \lambda$
end procedure
```

C.2 Proof of Lemma 1

Influence embeddings satisfy a critical property - that if two examples have similar influence embeddings, they also tend to have similar influence explanations. This is formalized by the following lemma:

lemmahere exists a constant C such that for any test examples z_i, z_j, $||E(z_i) - E(z_j)||^2 \leq C||\mu(z_i) - \mu(z_j))||^2$.

Proof. The proof follows from the Cauchy-Schwartz inequality, and the fact that influence is the dot-product of influence embeddings.

$$\begin{aligned}
||E(z_i) - E(z_j)||^2 &= \textstyle\sum_{n=1}^{N'} (\hat{I}({z'}_n, z_i) - \hat{I}({z'}_n, z_j))^2 \\
&= \textstyle\sum_{n=1}^{N'} (\mu({z'}_n)^\intercal \mu(z_i) - \mu({z'}_n)^\intercal \mu(z_j))^2 \\
&= \textstyle\sum_{n=1}^{N'} (\mu({z'}_n)^\intercal (\mu(z_i) - \mu(z_j)))^2 \\
&\leq \textstyle\sum_{n=1}^{N'} ||\mu({z'}_n)||^2 ||\mu(z_i) - \mu(z_j)||^2 \\
&\leq C||\mu(z_i) - \mu(z_j)||^2, \text{ where}
\end{aligned}$$

$C := \sum_{n=1}^{N'} ||\mu({z'}_n)||^2$ does not depend on i nor j. □

C.3 Description of InfEmbed

We will solve the formulation of Equation 4 via a simple procedure which applies K-Means with Euclidean distance to the influence *embeddings* of the test dataset, $\boldsymbol{\mu} := [\mu(z_1), \ldots, \mu(z_N)]$. This procedure is justified as follows: Using Lemma 1, given any partition $\{\Phi_k\}$, we know

$$\textstyle\sum_k \sum_{i,i' \in \Phi_k} ||E(z_i) - E(z_{i'})||^2 \leq \tag{9}$$

$$C \textstyle\sum_k \sum_{i,i' \in \Phi_k} ||\mu(z_i) - \mu(z_{i'})||^2. \tag{10}$$

However, the quantity in Equation 10 is the *surrogate* objective minimized by this procedure - it is the K-Means objective applied to influence embeddings, scaled by $C > 0$ which does not matter. The quantity in Equation 9 is the *actual* objective which the formulation of Equation 4 minimizes. Therefore, the procedure is minimizing a surrogate objective which upper bounds the actual objective we care about. Put another way, by applying K-Means to influence embeddings, examples within the same slice will not only have similar influence embeddings, but also similar influence explanations as originally desired, by Lemma **??**. Algorithm 4 is the proposed InfEmbed method.

Algorithm 5 Our SDM, `InfEmbed`, consists of K-Means clustering applied to influence embeddings of test examples.

```
 1: procedure GetEmbeddings($\mathbf{Z}, \mathbf{Z}', \Theta, P, D$)
 2:     Outputs: $D$-dimensional embeddings for test data $\mathbf{Z}$
 3:     $M, \lambda \leftarrow \text{FactorHessian}(\mathbf{Z}', \Theta, P, D)$
 4:     $\mu_i \leftarrow \lambda^{-1/2} M^\intercal \nabla_\theta L(z_i; \theta)$ for $i = 1, \dots, N$
 5:     $\boldsymbol{\mu} \leftarrow [\mu_1, \dots, \mu_N]$
 6:     Return: $\boldsymbol{\mu}$
 7: end procedure
 8: procedure K-Means($\boldsymbol{\mu}, K$)
 9:     randomly initialize cluster centers $c_1, \dots, c_K$
10:     while not converged do
11:         $r_i = \text{argmin}_k \, ||\mu_i - c_k||^2$ for $\mu_i \in \boldsymbol{\mu}$
12:         $c_k = \frac{\sum_i 1[r_i = k]\mu_i}{\sum_i 1[r_i = k]}$ for $k \in [K]$
13:     end while
14:     Return: $[r_1, \dots, r_N]$
15: end procedure
16: procedure InfEmbed(K, $\mathbf{Z}, \mathbf{Z}', \Theta, P, D$)
17:     Inputs: number of slices $K$, training dataset $\mathbf{Z}'$, test dataset $\mathbf{Z}$, configuration $\Theta$, Arnoldi
    dimension $P$, influence embedding dimension $D$
18:     Outputs: partition of test dataset into slices, $\Phi$
19:     $\boldsymbol{\mu} \leftarrow \text{GetEmbeddings}(\mathbf{Z}, \mathbf{Z}', \Theta, P, D)$
20:     $[r_1, \dots, r_N] \leftarrow \text{K-Means}(\boldsymbol{\mu}, K)$
21:     Return: $\{\{i \in [N] : r_i = k\}$ for $k \in [K]\}$
22: end procedure
```

C.4 Description of InfEmbed-Rule

Note that the key hyperparameter of the InfEmbed method is K, the number of slices to return. In practice, it may not be intuitive for a user to choose K. Instead, the user may want to know if there exists any coherent slices that are problematic, as defined by satisfying a rule: has accuracy less than some threshold, and number of examples above some threshold. Therefore, we also propose a procedure that recursively clusters influence embeddings until slices satisfying the rule are found, or until the slices are too small. The approach is analogous to building a tree to identify slices satisfying the rule, where the splits are determined by K-Means clustering of influence embeddings. In addition to letting the user specify more intuitive hyperparameters, this procedure also has the advantage that if a large slice with sufficiently low accuracy is found, it will not be clustered further. Algorithm 1 outlines this `InfEmbed-Rule` method. Its inputs are the same as `InfEmbed`, except instead of specifying K, the number of slices, one specifies accuracy threshold A, size threshold S, and branching factor B, which specifies how many clusters the K-means call in each step of the recursion should return. Its outputs is a set of slices, each having accuracy less than A and size greater than S. In practice, the recursion proceeds to a maximum depth.

C.5 Properties of Discovered Slices

Existing SDMs Eyuboglu et al. (2022b); Hua et al. (2022) explicitly encourage slices to have a *label homogeneity* property—that a given slice is homogeneous in terms of both the true and predicted

Algorithm 6 Procedure finding slices with low accuracy and large size

```
procedure RULEFIND($\boldsymbol{\mu}, A, S, B$)
    Inputs: $\boldsymbol{\mu}$ a list of influence embeddings, accuracy threshold $A$, size threshold $S$, branching factor $B$
    Outputs: a set of lists of influence embeddings. Each set of influence embeddings corresponds to a slice with accuracy < $A$ and size > $S$
    $acc \leftarrow$ accuracy of examples represented in $\{\mu_i\}$
    $size \leftarrow$ number of examples represented in $\{\mu_i\}$
    if $acc \leq A$ and $size \geq S$ then
        Return: $\{\boldsymbol{\mu}\}$
    end if
    if $size < S$ then
        Return: $\{\}$
    end if
    $\boldsymbol{r} \leftarrow$ K-Means$(\boldsymbol{\mu}, B)$                    ▷ get cluster assignments
    $\boldsymbol{r}_k \leftarrow \{\mu_i\}_{i:r_i=k}$ for $k \in [B]$               ▷ embeddings for each cluster
    $F = \{\}$
    for $k \in [B]$ do                                   ▷ search within each cluster
        $F \leftarrow F \cup \{\text{RuleFind}(\{\mu_i\}_{i:r_i=b}, A, S, B)\}$
    end for
    Return: $F$
end procedure
procedure INFEMBED-RULE($A, S, B, \mathbf{Z}, \mathbf{Z}', \Theta, P, D$)
    $\boldsymbol{\mu} \leftarrow$ GetEmbeddings$(\mathbf{Z}, \mathbf{Z}', \Theta, P, D)$
    $F \leftarrow \texttt{RuleFind}(\boldsymbol{\mu}, A, S, B)$      ▷ a set of lists of embeddings, with each embedding corresponding to a test example
    Return: the partition of test dataset induced by $F$
end procedure
```

label, relying on "error-aware" mixture models of examples' features, predicted label, and true label. This requires tuning a hyperparameter trading off homogeneity in labels versus that in features.

In contrast, our procedure is able to achieve label homogeneity by simply applying K-Means to influence embeddings without any hyperparameter. This is because K-Means encourages influence embeddings within the same slice to have low Euclidean distance, which is in turn their negative dot-product plus a constant. Therefore, we consider what factors results in two examples $z = (x, y)$, $z' = (x', y')$ having influence embeddings with high dot-product. For simplicity, we consider when the dot-product of their *gradients* would be high; influence embeddings are linearly transformed gradients (see Equation 5), and high dot-product before transformation implies high dot-product after transformation. For further simplicity, we consider the case when the model f is a linear model, so that $f(x;\theta) = \theta^\intercal x$, where $\theta = [B_1, \ldots, B_C]$, linear coefficient vectors for each class.

Given that L is cross-entropy loss, we define $p := [p_1, \ldots, p_C] = \text{softmax}(f(x))$, the predicted probabilities for each class for example z, and define p' analogously for example z'. A simple calculation shows $\nabla_\theta L(z;\theta) = [(1[y_1{=}1]-p_1)x, \ldots, (1[y_C{=}1]-p_C)x]$, where y_c indicates whether the label is class c. Thus,

$$\nabla_\theta L(z;\theta)^\intercal \nabla_\theta L(z';\theta) = (y-p)^\intercal (y'-p') x^\intercal x'. \tag{11}$$

Note that $y - p$ is the *margin* for example z—the vector containing the difference between true label and predicted probability for each class (and similarly for $y' - p'$). Thus, clustering influence embeddings leads to label homogeneity due to the use of gradients. The gradients of two examples are similar by dot-product when their margins are similar by dot-product, which occurs if their true labels and predicted probabilities are similar.

Equation 11 also shows that in the simple case of a linear model, our method considers the dot-product between the last-layer activations, i.e. input features for this case. However, in the more general case, unlike past slice discovery methods relying on last-layer activations, our method considers information in other layers of the model simply by virtue of considering gradients in them.

C.6 Root-Cause Analysis via Slice Opponents

Given an under-performing slice, we are interested in identifying the root-cause of the erroneous predictions in that slice. As the root-cause, we compute the strongest *opponents* of the slice. These opponents are the training examples whose influence on the *slice*, i.e. influence on the *total* loss over all examples in the slice is the most negative, i.e., most harmful. Due to the properties of influence embeddings, the influence of a training example on a slice is simply the dot-product between the influence embedding of the training example, and the *sum* of the influence embeddings in that slice. Inspecting slice opponents can provide insight into the key features responsible for low performance in a slice.

D DcBench

Overview. Eyuboglu et al. (2022b) formalized the slice discovery problem and introduced Dcbench (Eyuboglu et al., 2022a), a benchmark, consisting of pre-trained models across a variety of data sets for testing any new SDM. For each dataset in the benchmark, a manipulation is applied to the dataset in order to induced one of three kinds of errors in a collection of samples—slice—, and a model is trained on the modified dataset to exhibit the injected error. To assess a new SDM, the partitioning returned by the method is then compared to the ground-truth slices used to generate the model. The collection of datasets in dcbench includes natural images (ImageNet, and CelebA), Medical images based on the MIMIC Chest X-Ray, and Medical Time-Series Data of electroencephalography (EEG) signals. A collection of 1235 models were trained for various dataset, model, and error type groups. In addition to the benchmark, Eyuboglu et al. (2022b) introduced Domino, an SDM, that uses a mixture model that models the generative process of a slice. The input to the mixture model is a multi-model embedding (CLIP or ConVIRT) of the test data, and the mixture model assumes that the embedding, label, and predictions are independent conditioned on the slice. Eyuboglu et al. (2022b) demonstrate that Domino outperforms or matches approaches based on last-layer or vision-transformer-based embeddings. Consequently, we compare against Domino on dcbench since it is, as of the writing of this paper, the state-of-the-art.

Experimental Procedure. The dcbench benchmark consists of a collection of slice discovery tasks. Each task includes a series of artifacts: the model, base dataset, model activations, validation and test predictions, clip activations for the test and validation data, and ground-truth slices. Domino was evaluated using two kinds of models: a synthetic and trained model. We restrict our focus to the trained models, only, since our proposed approach only applies to trained models. We compute the precision-at-10 metric, similar to domino, across all data modalities and error type. We refer the reader to Section D of the Appendix for additional details of the setup.

Results. We present a comparison of the Precision-at-10 metric between Domino and the influence embeddings approach in Table 1. The influence embedding approach outperforms domino across all settings except on the Noisy Label task for the EEG data modality.

D.1 Datasets

All of the dcbench dataset, models, and task information is publicly available via the opensource repository: `https://github.com/data-centric-ai/dcbench`

E Imagenet & High Dimensional Settings

Overview & Experimental Procedure: The Imagenet validation data Deng et al. (2009) contains 1000 classes, with 50 examples per class. We use the `InfEmbed-Rule` procedure to find slices with at most 40% accuracy, and at least 25 examples.

We use a Resnet18 trained on Imagenet, provided in the torchvision PyTorch library. The model achieves 69.8% accuracy. To compute influence embeddings, we consider gradients in the "fc" and "layer4" layers, which contain a total of 8.9M parameters. As in all experiments, we compute influence embeddings of $D = 100$ dimensions using an Arnoldi dimension of $P = 500$. To explain the root-cause of predictions in a given slice, we compute the strongest opponents of the slice. These are the training examples whose influence on the *slice*, i.e. influence on the *total* loss over all examples

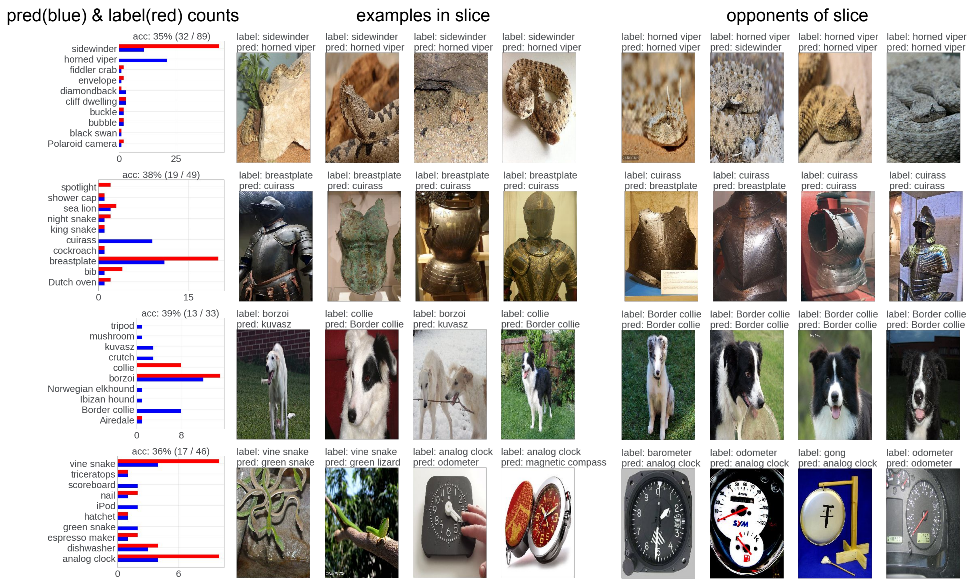

Figure 2: For slices discovered by applying `InfEmbed-Rule` to Imagenet, we show the distributions of labels (red) and predictions (blue) in the slice (left), examples from the slice (middle), and opponents of the slice (right).

in the slice is the most negative, i.e. most harmful. Due to the properties of influence embeddings, the influence of a training example on a slice is simply the dot-product between the influence embedding of the training example, and the *sum* of the influence embeddings in the slice. For simplicity, we search for opponents within the test dataset, i.e. treating the test data as the training data. Since there is no distribution shift in Imagenet, for explanatory purposes this is an acceptable approximation.

Results: Applying the above rule, we find 25 slices, comprising a total of 954 examples, whose overall accuracy is 34.9%. Figure E displays 4 of the slices, where each row contains the distribution of predicted and true labels, the 4 examples nearest the slice center in influence embedding space, and the 4 strongest opponents of the slice. First, we see that the slices have the label homogeneity property as explained in Section C.5 - the predicted and true labels typically come from a small number of classes. In the first slice (counting from the top), the true label is mostly "sidewinder", and often predicted to be "horned viper". The slice's strongest opponents are horned vipers, hard examples of another class, which the model likely models similarly to sidewinders. Thus the presence of the former drives the prediction of the latter towards horned vipers and away from sidewinders, increasing loss. A similar story holds for the second slice, where breastplates are often predicted to be cuirasses. Interestingly, the first opponent may actually be mis-labeled (cuirasses are breastplates fused with backplates, which it may lack), reflecting the fine line between hard and mis-labeled opponents examples. The third and fourth slices are interesting in that each slice is predominantly of *two* labels. For example, in the third slice, border collies are strong opponents for both borzois and collies (which are a different dog breed than border collies), causing them to be mis-predicted as border collies. This makes sense given that all three dog breeds look similar. In the fourth slice, vine snakes and analog clocks are often mis-predicted. A priori, we would not know that predictions for these seemingly unrelated classes would be wrong for the same reasons. However, looking at the opponents, we see all of them have a clock hand, which turns out to be similar to a snake. Also, because the opponents are of a variety of classes, the mis-predictions are also of a variety of classes.

Finally, we also list a few other discovered slices: 1) a slice where "gown" (wedding dress) is mis-predicted to be "groom", whose opponents are images labeled "groom" containing both a groom and spuriously-correlated gown, 2) a slice where "windsor tie" is mis-predicted to be suit, due to similar spurious correlations, 3) a slice where mis-predicted "sunglasses" are due to examples which contain sunglasses, but are labeled as other present classes, like "lipstick" and "bib", 4) two different classes (i.e. index) actually refer to the same object - "maillot".

F AGNews & Mis-labeled Data

Overview & Experimental Procedure: The AGnews test data Zhang et al. (2015) contains 4 classes (Business, Sci/Tech, Sports, World) with 1900 examples per class. We use a BERT base model fine-tuned on the training set [4], which achieves 93.75% accuracy on the test data which results in 475 test errors. We use the InfEmbed method of Algorithm 1 with $K = 25$ to find slices with at most 10% accuracy, and at least 10 examples since the total number of errors is small.

To compute influence embeddings, we consider gradients in the "bert.pooler.dense" and "classifier" layers, which are the top 2 linear layers of the model and contain a total of 590K parameters parameters. As in all experiments, we compute influence embeddings of $D = 100$ dimensions using an Arnoldi dimension of $P = 500$.

Results: We find 9 slices that account for 92% of errors (438 out of 475). Some interesting patterns we observe:

- Sci/Tech examples predicted as Business (30%) & Business predicted as Sci/Tech (25%) are 55% of errors.
- Business examples predicted as World (10.3%) & World predicted as Business (9.4%) are 19.7% of errors.
- Sci/Tech examples predicted as World (8.5%) & World predicted as Sci/Tech (7.3%) are 15.8% of errors.
- Sports articles are least likely to get predicted for other genres and only a few Business and Sci/Tech examples get predicted as Sports (4.1% and 2.7% respectively of errors.)

Table 3 displays representative examples (ie, those closet to the cluster centers) from 3 of the slices. For Sci/Tech news articles that are predicted as Business articles, we see Google, PalmOne and Intel all referenced, but with references to investment bank reports, Stock symbols and earning reports, the true class is a bit ambiguous and illustrates why the model has the most troubles distinguishing between the two (they account for 55% of errors). In fact, it is possible examples in this slice are actually mis-labeled, showing that `InfEmbed` can be used to detect systematically mis-labeled data. For Business articles that are predicted to be World articles we see references to the grocer Tesco, Shell and cocoa farmers, but with reference to Britain, the Nigerian Senate and protests in the Ivory coast which again makes their class a bit ambiguous; accounting for 20% of model errors. Finally, for Business examples that are predicted as Sports, which is one of the smaller error slices we find, we see less ambiguous error instances which the model has trouble with.

Furthermore, we display in Figure F the distribution of labels and predictions for the 9 slices, as well as their error rate. We see that the slices possess the label homogeniety property.

G Detecting Spurious Signals in Boneage Classification

Bone Age Dataset: We consider the high stakes task of predicting the bone age category from a radiograph to one of five classes based on age: Infancy/Toddler, Pre-Puberty, Early/MiD Puberty, Late Puberty, and Post Puberty. This task is one that is routinely performed by radiologists and as been previously studied with a variety of DNN. The dataset we use is derived from the Pediatric Bone Age Machine learning challenge conducted by the radiological society of North America in 2017 [5] Halabi et al. (2019). The dataset consists of 12282 training, 1425 validation, and 200 test samples. We resize all images to (299 by 299) grayscale images for model training. We note here that the training, validation, and test set splits correspond to similar splits used for the competition, so we retain this split.

Model and Hyperparameters: We consider and a Resnet-50 model. The small DNN consists of: conv-relu-batchnorm-maxpooling operation successively, and two fully connected layers at the end. All convolutional kernels have stride 1, and kernel size 5. We train this model with SGD with momentum (set to 0.9) and an initial learning rate of 0.01. We use a learning rate scheduler that decays the learning rate every 10 epochs by $\gamma = 0.1$.

[4]https://huggingface.co/fabriceyhc/bert-base-uncased-ag_news

[5]https://www.kaggle.com/datasets/kmader/rsna-bone-age

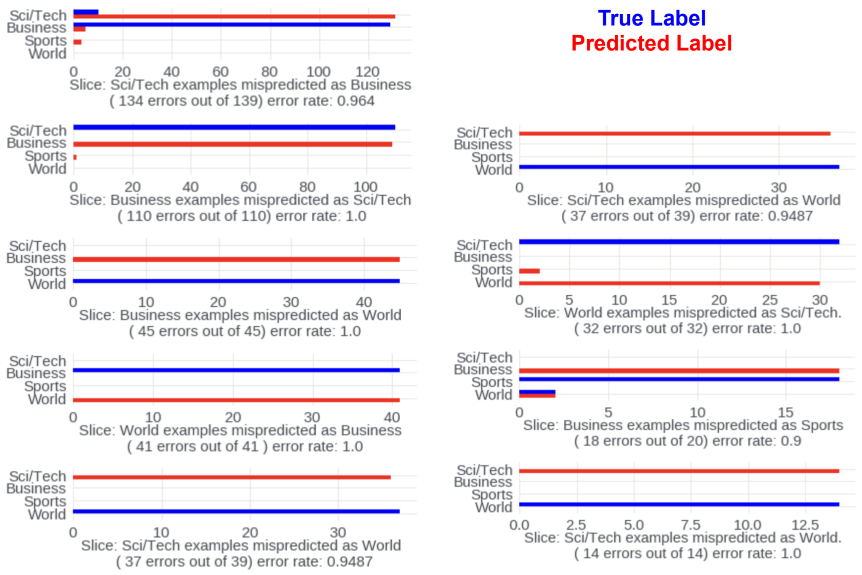

Figure 3: Predicted vs True label distributions for 9 slices found for AGnews data

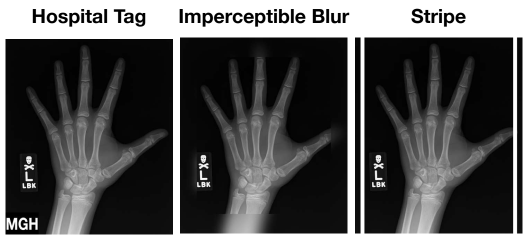

Figure 4: We introduce spurious correlations into the bone-age data of Section 4.4 by injecting 1 of 3 possible signals - a hospital tag, blur, or stripe. Examples with injected signals are shown here.

Overview & Experimental Procedure: We now artificially inject a signal that spuriously correlates with the label in training data, and see if `InfEmbed-rule` can detect the resulting test errors as well as the spurious correlation. Here, the classification problem is bone-age classification - to predict which of 5 age groups a person is in given their x-ray. Following the experimental protocol of Zhou et al. (2022) and Adebayo et al. (2021), we consider 3 possible signals that can be added to an x-ray, shown in Figure 4 - adding a tag, stripe, or blur. Importantly, spurious correlations involving these signal have been difficult to detect in past work. For a given signal, we manipulate the bone-age training dataset by injecting the spurious signal into all *training* examples from the "mid-pub" class (short for mid-puberty, one of the 5 age groups). Importantly, we leave the test dataset untouched, so that we expect a model trained on the manipulated training data to err on "mid-pub" test examples; the spurious signal the model associated with "mid-pub" is missing, so that the model, having not associated other features with "mid-pub", finds no evidence to deem it "mid-pub". This setup mimics a realistic scenario where x-ray training data comes from a hospital where one class is over-represented, and also contains a spurious signal (i.e. x-rays from the hospital might have the hospital's tag).

Examples from slice with 139 examples, 3% accuracy. Predictions are 93% "business", labels are 94% "sci/tech"
1. *"Google Unveils Desktop Search, Takes on Microsoft Google Inc. (GOOG.O: Quote, Profile, Research) on Thursday rolled out a preliminary version of its new desktop search tool, making the first move against ..."*
2. *"PalmOne to play with Windows Mobile? Rumors of Treo's using a Microsoft operating system have been circulating for more than three years. Now an investment bank reports that PalmOne will use a ..."*
3. *"Intel Posts Higher Profit, Sales Computer-chip maker Intel Corp. said yesterday that earnings for its third quarter were $1.9 billion – up 15 percent from the same quarter a year ago ..."*

Examples from slice with 45 examples, 0% accuracy. Predictions are 100% "world", labels are 100% "business"
1. *"British grocer Tesco sees group sales rise 12.0-percent (AFP) AFP - Tesco, Britain's biggest supermarket chain, said that group sales grew by 12.2 percent in the third quarter, driven by strong performances from its stores ..."*
2. *"Nigerian Senate approves $1.5 bln claim on Shell LAGOS - Nigeria's Senate has passed a resolution asking Shell's Nigerian unit to pay $1.5 billion in compensation to oilfield communities for pollution, a Senate spokesman said. "*
3. *"Cocoa farmers issue strike threat. Unions are threatening a general strike in the Ivory Coast in a protest against the prices farmers are paid for their cocoa supplies."*

Examples from slice with 20 examples, 10% accuracy. Predictions are 90% "world", labels are 90% "business"
1. *"Perry OKs money for APS as more accusations arise. The state's Adult Protective Services agency will get an emergency infusion of $10 million to correct the kinds of problems that have arisen in El Paso."*
2. *Sign off, then sign in. G. Michael Caggiano Jr. lies awake at night thinking about bank signs. He ponders them during breakfast, while brushing his teeth, and quot;constantly quot; during the day, he says."*
3. *"Stanley set sights on Elland Road for casino Stanley Leisure plc has announced a Stanley Casinos Limited plan to develop a casino complex on land adjacent to Leeds United's Elland Road stadium."*

Table 3: Representative test examples for 3 high error slices obtained using `InfEmbed` on AGnews

We then train a resnet50 on the manipulated training data, and apply `InfEmbed-Rule` on the untouched test data to discovered slices with at least 25 examples and at most 40% accuracy (same as for the Imagenet analysis). To assess whether `InfEmbed-Rule` succeeded at detecting the model's reliance on the spurious training signal, we 1) take the most problematic slice, i.e. discovered slice with the lowest accuracy, and see if its labels are mostly "mid-pub" - the class we know a priori to be under-performing in the test dataset due to spurious training signal injection, and 2) examine whether the slice opponents are training examples with the spurious training signal, i.e. of the "mid-pub" class. This lets us confirm whether the errors in the slice are due to the spurious training signal (as opposed to other root-causes), and whether using `InfEmbed-Rule` along with slice opponents analysis lets us discover the spurious training signal. We repeat this manipulate-train-slice discovery procedure for 3 runs: once for each possible signal.

Results: For all 3 runs / signals, `InfEmbed-Rule` returned 2 slices satisfying the rule. In Figure 5, for each signal, we show the distribution of labels and predictions for the most problematic slice (left column) and the same distribution for that slice's top-50 opponents (right column). We see that for each signal, the labels in the most problematic slice are of the "mid-pub" class, showing that `InfEmbed-Rule` is able to detect test errors due to the injected spurious training signal. Furthermore, for each run, around 80% of the slice's opponents are training examples with the spurious training signal, i.e. examples with the "mid-pub" label. Together, this shows `InfEmbed-Rule` can not only detect slices that the model errs on because of a spurious training signal, but also identify the

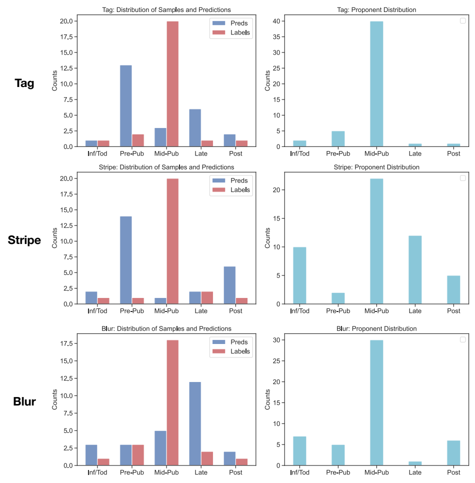

Figure 5: For all 3 spuriously-injected signals for the boneage classification task, we plot the label distribution for the lowest-accuracy slice in test data discovered by `InfEmbed-Rule` in red (left), and the label distribution for the slice's opponents in training data on the right.

root-cause of those errors by using slice opponents to identify training examples with the spurious signal.

H Applying Influence Embeddings to Limited Data Settings Using Attributes

We use the COVID-19 Chest X-Ray Dataset[6]. This dataset, provided in the torchxrayvision PyTorch library, has 535 images of chest X-rays. We use the COVID-19 label, a binary label, where there are 342 cases of COVID. We leverage a pretrained resnet50 model provided in the torchxrayvision library, trained to predict multiple diseases, and adapt it to predict the COVID label by replacing the last fully-connected layer of the model. We then finetune the model, including this last layer, on a training split of the COVID dataset using a binary cross-entropy loss. We follow standard normalization techniques for these datasets, including using a central crop of 224x224 pixels and normalizing the image values in the $[-1024, 1024]$ range. The resulting model achieves an accuracy of 76.6% on the test set of 107 points. We apply InfEmbed, taking gradients with respect to the last fully connected layer of the model with $K = 3$, Arnoldi dimension $P = 250$, and influence embedding dimension $D = 125$, considering gradients in the 'classifier', 'features.denseblock4.denselayer16.conv1', and 'features.denseblock4.denselayer16.conv2' layers.

Results. The first cluster had an accuracy of 85%, capturing the majority of correctly classified samples. Besides correctly classified positive and negative samples, the first cluster also had five false positives and nine false negatives. The second cluster consisted of four samples that were false positives and two samples that were true negatives, while the third cluster had six samples that were false positives, and one sample that was a false negative. To a layperson not trained in interpreting radiology images, it may be hard to pick out differences between images in the three clusters. To study the differences between the three clusters, we inspected attributes and metadata provided in the dataset.

Besides the COVID-19 label, the dataset had 18 additional labels for various diseases and infections, from Aspergillosis to Varicella. The full list can be found in Table 4; see Cohen et al. (2020) for more details. We computed the incidence of these diseases for each of the true negatives, true positives, false negatives, false positives samples by cluster. Table 4 presents the results. We observe that false positive samples in all three clusters tended to have pneumonia and tuberculosis, but not COVID-19. Looking at true positives, all the samples in cluster 1 that had COVID-19 also had pneumonia.

[6] https://github.com/ieee8023/covid-chestxray-dataset

However, samples with pneumonia do not always have COVID-19; rather, sometimes they have other lung diseases such as tuberculosis, SARS, etc.

A differentiator between the three clusters is the incidence of additional diseases. Some false positive samples in cluster 2, in addition to pneumonia and tuberculosis, also had legionella, while those in cluster 3 in addition had SARS or pneumocystis. The false positive samples in cluster 1 had more diseases, such as Herpes, Klebsiella, and several others. Many of these diseases are lung diseases; these findings illustrate that the model may be having a harder time differentiating between different lung diseases and COVID-19.

	True Negatives			**False Positives**			**False Negatives**			**True Positives**		
	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
Aspergillosis	0.04	0.0	NaN	0.0	0.00	0.00	0.0	NaN	0.0	0.0	NaN	NaN
COVID-19	0.00	0.0	NaN	0.0	0.00	0.00	1.0	NaN	1.0	1.0	NaN	NaN
Chlamydophila	0.00	0.0	NaN	0.0	0.00	0.00	0.0	NaN	0.0	0.0	NaN	NaN
H1N1	0.00	0.0	NaN	0.0	0.00	0.00	0.0	NaN	0.0	0.0	NaN	NaN
Herpes	0.00	0.0	NaN	0.2	0.00	0.00	0.0	NaN	0.0	0.0	NaN	NaN
Influenza	0.00	0.0	NaN	0.0	0.00	0.00	0.0	NaN	0.0	0.0	NaN	NaN
Klebsiella	0.12	0.0	NaN	0.2	0.00	0.00	0.0	NaN	0.0	0.0	NaN	NaN
Legionella	0.00	0.5	NaN	0.0	0.25	0.00	0.0	NaN	0.0	0.0	NaN	NaN
MERS-CoV	0.00	0.0	NaN	0.2	0.00	0.00	0.0	NaN	0.0	0.0	NaN	NaN
MRSA	0.00	0.0	NaN	0.0	0.00	0.00	0.0	NaN	0.0	0.0	NaN	NaN
Mycoplasma	0.00	0.0	NaN	0.0	0.00	0.00	0.0	NaN	0.0	0.0	NaN	NaN
Nocardia	0.04	0.0	NaN	0.0	0.00	0.00	0.0	NaN	0.0	0.0	NaN	NaN
Pneumocystis	0.16	0.0	NaN	0.2	0.00	0.33	0.0	NaN	0.0	0.0	NaN	NaN
Pneumonia	0.92	1.0	NaN	1.0	0.25	1.00	1.0	NaN	1.0	1.0	NaN	NaN
SARS	0.12	0.0	NaN	0.0	0.00	0.50	0.0	NaN	0.0	0.0	NaN	NaN
Staphylococcus	0.00	0.0	NaN	0.0	0.00	0.00	0.0	NaN	0.0	0.0	NaN	NaN
Streptococcus	0.04	0.0	NaN	0.0	0.00	0.00	0.0	NaN	0.0	0.0	NaN	NaN
Tuberculosis	0.08	0.0	NaN	0.0	0.25	0.00	0.0	NaN	0.0	0.0	NaN	NaN
Varicella	0.04	0.0	NaN	0.0	0.00	0.00	0.0	NaN	0.0	0.0	NaN	NaN

Table 4: Average characterization of the true negatives (tn), false positives (fp), false negatives (fn), and true positives (tp) of each of the 3 clusters C0, C1, and C2. We compute the characterization by averaging the 20-dimensional ground-truth label of each sample (that describes the pathologies associated with that sample) across all samples belonging in that group. For groups where no sample is assigned, the characterization is NaN.

I Limitations of Influence Embeddings

One major limitation of the proposed approach is that the validation set needs to include the kind of error that one is seeking to detect. In addition, the proposed procedure suggests to inspect the slice opponents as the root cause of the wrong predictions in a slice. However, it is still up to the user of the proposed Algorithm to infer which feature(s) of the slice opponents is responsible for causing the errors.