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 underperforming 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 \( \theta \) 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 \( f(x; \theta) \) is the pre-softmax prediction for class \( c \). We are also given a test dataset with \( N \) examples \( \mathbf{z} := [z_1, \ldots, z_N] \), 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 \( k \neq k' \), \( \bigcup_{k=1}^K \Phi_k = [N] \). As we will make precise shortly, each slice \( \Phi_k \) should be coherent, and some slices should be under-performing slices. The model is assumed to be trained on a training dataset \( \cup X \cup Z \) where \( X \) is the training dataset and \( Z \) is the test dataset partitioned into slices. Formally, we seek:

\[
\text{return a partition of the test dataset into slices, } \{\Phi_k\}_{k=1}^K, \text{ that minimizes the total Euclidean distance between influence explanations of examples in the same slice.}
\]

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

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

\[
I(z', z) := \nabla_{\theta} L(z'; \theta)^\top \tilde{H}_\theta^{-1} \nabla_{\theta} L(z; \theta),
\]

where \( \tilde{H}_\theta \) is the loss Hessian for the pre-trained model:

\[
\tilde{H}_\theta := \frac{1}{n} \sum_{i=1}^n \nabla_2^2 L(z_i; \theta).
\]

The main challenge in computing influence is that it is impractical to explicitly form \( \tilde{H}_\theta \) 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 \( \tilde{H}_\theta \) via a procedure that does not explicitly form \( \tilde{H}_\theta \). In brief, they run the Arnoldi iteration Trefethen & Bau III (1997) for \( P \) iterations to get a \( P \)-dimensional Krylov subspace for \( \tilde{H}_\theta \), which requires only \( P \) Hessian-vector products. Then, they find a rank-\( D \) approximation of the restriction of \( \tilde{H}_\theta \) 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 \( \tilde{H}_\theta^{-1} \)

\[
M, \lambda = \text{FactorHessian}(\mathbf{z}', \Theta, P, D) \text{ where } \tilde{H}_\theta^{-1} := M \lambda^{-1} M^\top \approx H_\theta^{-1},
\]

\( M \in \mathbb{R}^{\theta \times D} \), \( \lambda \in \mathbb{R}^{D \times D} \) is a diagonal matrix, \( \theta \) is the parameter count, \( \tilde{H}_\theta^{-1} \) approximates \( H_\theta^{-1} \), \( D \) 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. \( \tilde{H}_\theta^{-1} \) is then used to define the practical influence of training example \( z' \) on test example \( z \):

\[
\hat{I}(z', z) := \nabla_{\theta} L(z'; \theta)^\top \tilde{H}_\theta^{-1} \nabla_{\theta} L(z; \theta).
\]

3 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 \( E(z) \):

\[
E(z) := \{\hat{I}(z', z)\}_{1}^{N'},
\]

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:

\[
\text{argmin}_{\{\Phi_k\}_{k=1}^K} \sum_k \sum_{i, i' \in \Phi_k} ||E(z_i) - E(z_{i'})||^2.
\]
Algorithm 2 Our SDM, InfEmbed, applies K-Means to influence embeddings of test examples.

1: procedure GETEMBEDDINGS(Z, Z', Θ, P, D)
2: Outputs: D-dimensional embeddings for test data Z
3: M, λ ← FactorHessian(Z', Θ, P, D)  \(\triangleright\) see Equation 1
4: \(\mu_i \leftarrow \lambda^{-1/2}M^T \nabla_{\theta} L(z_i; \theta)\) for \(i = 1, \ldots, N\) \(\triangleright\) see Equation 5
5: \(\mu \leftarrow [\mu_1, \ldots, \mu_N]\)
6: Return: \(\mu\)
7: end procedure
8: procedure INFEMBED(K, Z, Z', Θ, P, D)
9: Inputs: number of slices \(K\), test dataset \(Z\), training dataset \(Z'\), configuration \(\Theta\), Arnoldi dimension \(P\), influence embedding dimension \(D\)
10: Outputs: partition of test dataset into slices, \(\Phi\)
11: \(\mu \leftarrow \text{GetEmbeddings}(Z, Z', Θ, P, D)\)
12: \([r_1, \ldots, r_N] \leftarrow \text{K-Means}(\mu, K)\) \(\triangleright\) compute cluster assignments for all \(N\) examples
13: Return: \([\{i \in [N] : r_i = k\} \text{ for } k \in [K]\}] \(\triangleright\) 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

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

and \(M, \lambda\) are as defined in Section 2, because for any training example \(z'\) and test example \(z\), \(\check{I}(z', z) = \mu(z')^T \mu(z)\). Procedure GetEmbeddings of Algorithm 1 finds \(D\)-dimensional influence embeddings for the test dataset \(Z\), 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, \(\mu := [\mu(z_1), \ldots, \mu(z_N)]\). This procedure is justified as follows: Using Lemma 1, given any partition \(\{\Phi_k\}\), we know

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

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
Workshop on the pitfalls of limited data and computation for Trustworthy ML, ICLR 2023

<table>
<thead>
<tr>
<th>SDM Approach</th>
<th>Rare</th>
<th>Correlation</th>
<th>Noisy Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domino</td>
<td>0.4</td>
<td>0.45</td>
<td>0.6</td>
</tr>
<tr>
<td>Inf-Embed</td>
<td>0.65</td>
<td>0.55</td>
<td>0.67</td>
</tr>
<tr>
<td>Medical Images</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domino</td>
<td>0.39</td>
<td>0.6</td>
<td>0.58</td>
</tr>
<tr>
<td>Inf-Embed</td>
<td>0.57</td>
<td>0.62</td>
<td>0.73</td>
</tr>
<tr>
<td>Medical Time Series</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domino</td>
<td>0.06</td>
<td>0.55</td>
<td>0.9</td>
</tr>
<tr>
<td>Inf-Embed</td>
<td>0.64</td>
<td>0.65</td>
<td>0.81</td>
</tr>
</tbody>
</table>

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 tends 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 dimension \( P = 500 \), and influence embedding dimension \( D = 100 \), unless noted otherwise.

4.1 INF Embed 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 InfEmbed-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.

1https://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 \(^3\), achieving 93% test accuracy, making 475 errors. We use InfEmbed 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 homogeniety. 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 BONE-AGE 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

\(^{1}\)https://huggingface.co/fabriceyhc/bert-base-uncased-ag_news
\(^{2}\)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


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


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 Iain 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 INFELEMENT

C.1 PRACTICAL LOW-RANK INFLUENCE FUNCTION

The main challenge in computing influence is that it is impractical to explicitly form the $H_\theta$ 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 $H_\theta^{-1}$ without explicitly forming $H_\theta$. Their low-rank implementation first applied the Arnoldi iteration Trefethen & Bau III (1997) to compute an orthonormal basis $(q_1, \ldots, q_P)$ for the $P$-dimensional Arnoldi subspace $(b, H_\theta b, \ldots, H_\theta^{P-1} b)$, as well as the restriction $R$ of $H_\theta$ to the subspace, so that $H_\theta = QRQ^T$, where $Q := [q_1, \ldots, q_P] \in \mathbb{R}^{D \times P}$ 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 $H_\theta$ 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 $R \approx V \lambda V^T$ and $R^{-1} \approx V \lambda^{-1} V^T$, where $V = [v_1, \ldots, v_D] \in \mathbb{R}^{P \times D}$,
We will solve the formulation of Equation 4 via a simple procedure which applies K-Means with Workshop on the pitfalls of limited data and computation for Trustworthy ML, ICLR 2023

Finding low-rank factors of Hessian

Algorithm C.1 outlines finding the factors needed for $\hat{\Theta} := (L, f, \theta)$ denote the loss function, model, and parameters.

Algorithm 3 Finding low-rank factors of Hessian

**Procedure** FACTOR_HESSIAN($Z'$, $\Theta$, $P$, $D$)

**Inputs:** training data $Z'$, configuration $\Theta$, Arnoldi dimension $P$, rank $D$

$H_\Theta \leftarrow \sum_{n=1}^{N'} \nabla^2 L(z'_n; \theta)$ \hspace{1cm} \triangleright$ 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.

$$||E(z_i) - E(z_j)||^2 = \sum_{n=1}^{N'} (\hat{I}(z'_n, z_i) - \hat{I}(z'_n, z_j))^2$$

$$= \sum_{n=1}^{N'} (\mu(z'_n)\top \mu(z_i) - \mu(z'_n)\top \mu(z_j))^2$$

$$= \sum_{n=1}^{N'} (\mu(z'_n)\top (\mu(z_i) - \mu(z_j)))^2$$

$$\leq \sum_{n=1}^{N'} ||\mu(z'_n)||^2 ||\mu(z_i) - \mu(z_j)||^2$$

$$\leq C||\mu(z_i) - \mu(z_j)||^2$$

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

C.3 DESCRIPTION OF INF EMBED

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, $\mu := [\mu(z_1), \ldots, \mu(z_N)]$. This procedure is justified as follows: Using Lemma 1, given any partition $\{\Phi_k\}$, we know

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

$v_1, \ldots, v_D$ are the top $D$ eigenvectors of $R$, and $\lambda = diag(\lambda_1, \ldots, \lambda_D)$, 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 $H_\Theta^{-1}$,

$$\hat{H}_\Theta^{-1} := M \lambda^{-1} M\top,$$  \hspace{1cm} (7)

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

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

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

Proof. The proof follows from the Cauchy-Schwartz inequality, and the fact that influence is the dot-product of influence embeddings.
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 5 Our SDM, InfEmbed, consists of K-Means clustering applied to influence embeddings of test examples.

```
1: procedure GetEmbeddings(Z, Z', Θ, P, D)
2: Outputs: D-dimensional embeddings for test data Z
3: $M, λ \leftarrow \text{FactorHessian}(Z', Θ, P, D)$
4: $μ_i \leftarrow λ^{-1/2}M^T\nabla_θ L(z_i; θ)$ for $i = 1, ..., N$
5: $μ ← [μ_1, ..., μ_N]$
6: Return: $μ$
7: end procedure
8: procedure K-Means($μ, K$)
9: randomly initialize cluster centers $c_1, ..., c_K$
10: while not converged do
11: $r_i = \text{argmin}_k \|μ_i - c_k\|^2$ for $μ_i ∈ μ$
12: $c_k = \frac{\sum_{i=1}^{N} [r_i = k] μ_i}{\sum_{i=1}^{N} [r_i = k]}$ for $k ∈ [K]$
13: end while
14: Return: $[r_1, ..., r_N]$
15: end procedure
16: procedure InfEmbed(K, Z, Z', Θ, P, D)
17: Inputs: number of slices $K$, training dataset $Z'$, test dataset $Z$, configuration $Θ$, Arnoldi dimension $P$, influence embedding dimension $D$
18: Outputs: partition of test dataset into slices, $Φ$
19: $μ ← \text{GetEmbeddings}(Z, Z', Θ, P, D)$
20: $[r_1, ..., r_N] ← \text{K-Means}(μ, K)$
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(μ, A, S, B)

Inputs: μ 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 ← accuracy of examples represented in \{μ_i\}
size ← number of examples represented in \{μ_i\}
if acc ≤ A and size ≥ S then
  Return: \{μ\}
else if size < S then
  Return: {} end if
end if

r ← K-Means(μ, B) \[▷ get cluster assignments\]
r_k ← \{μ_i\}_{i:r_i=k} for k ∈ [B] \[▷ embeddings for each cluster\]
F = {} \[▷ search within each cluster\]
for k ∈ [B] do
  F ← F ∪ \{RuleFind(\{μ_i\}_{i:r_i=b}, A, S, B)\}
end for

Return: F end procedure

procedure InfEmbed-Rule(A, S, B, Z, Z', θ, P, D)

μ ← GetEmbeddings(Z, Z', θ, P, D)
F ← RuleFind(μ, 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

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 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; θ) = θ^T x, where θ = [B_1, ..., B_C], linear coefficient vectors for each class.

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

\nabla_θ L(z; θ) \nabla_θ L(z'; θ) = (y − p)^T (y' − p') x^T 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.
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”.

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).
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. 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 closest 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
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 Profits, 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."

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
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\footnote{https://github.com/ieee8023/covid-chestxray-dataset}. 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.
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.

<table>
<thead>
<tr>
<th></th>
<th>True Negatives</th>
<th>False Positives</th>
<th>False Negatives</th>
<th>True Positives</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C1</td>
<td>C2</td>
<td>C3</td>
<td>C1</td>
</tr>
<tr>
<td>Aspergillosis</td>
<td>0.04</td>
<td>0.0</td>
<td>NaN</td>
<td>0.0</td>
</tr>
<tr>
<td>COVID-19</td>
<td>0.00</td>
<td>0.0</td>
<td>NaN</td>
<td>0.0</td>
</tr>
<tr>
<td>Chlamyphila</td>
<td>0.00</td>
<td>0.0</td>
<td>NaN</td>
<td>0.0</td>
</tr>
<tr>
<td>H1N1</td>
<td>0.00</td>
<td>0.0</td>
<td>NaN</td>
<td>0.0</td>
</tr>
<tr>
<td>Herpes</td>
<td>0.00</td>
<td>0.0</td>
<td>NaN</td>
<td>0.2</td>
</tr>
<tr>
<td>Influenza</td>
<td>0.00</td>
<td>0.0</td>
<td>NaN</td>
<td>0.0</td>
</tr>
<tr>
<td>Klebsiella</td>
<td>0.12</td>
<td>0.0</td>
<td>NaN</td>
<td>0.2</td>
</tr>
<tr>
<td>Legionella</td>
<td>0.00</td>
<td>0.5</td>
<td>NaN</td>
<td>0.0</td>
</tr>
<tr>
<td>MERS-CoV</td>
<td>0.00</td>
<td>0.0</td>
<td>NaN</td>
<td>0.2</td>
</tr>
<tr>
<td>MRSA</td>
<td>0.00</td>
<td>0.0</td>
<td>NaN</td>
<td>0.0</td>
</tr>
<tr>
<td>Mycoplasma</td>
<td>0.00</td>
<td>0.0</td>
<td>NaN</td>
<td>0.0</td>
</tr>
<tr>
<td>Nocardia</td>
<td>0.04</td>
<td>0.0</td>
<td>NaN</td>
<td>0.0</td>
</tr>
<tr>
<td>Pneumocystis</td>
<td>0.16</td>
<td>0.0</td>
<td>NaN</td>
<td>0.2</td>
</tr>
<tr>
<td>Pneumonia</td>
<td>0.92</td>
<td>1.0</td>
<td>NaN</td>
<td>1.0</td>
</tr>
<tr>
<td>SARS</td>
<td>0.12</td>
<td>0.0</td>
<td>NaN</td>
<td>0.0</td>
</tr>
<tr>
<td>Staphylococcus</td>
<td>0.00</td>
<td>0.0</td>
<td>NaN</td>
<td>0.0</td>
</tr>
<tr>
<td>Streptococcus</td>
<td>0.04</td>
<td>0.0</td>
<td>NaN</td>
<td>0.0</td>
</tr>
<tr>
<td>Tuberculosis</td>
<td>0.08</td>
<td>0.0</td>
<td>NaN</td>
<td>0.0</td>
</tr>
<tr>
<td>Varicella</td>
<td>0.04</td>
<td>0.0</td>
<td>NaN</td>
<td>0.0</td>
</tr>
</tbody>
</table>

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.