2019 IEEE 35th International Conference on Data Engineering (ICDE)

Slice Finder: Automated Data Slicing for Model Validation

Yeounoh Chung*
Brown University
yeounoh_chung@brown.edu

Tim Kraska
MIT
kraska@mit.edu

Neoklis Polyzotis
Google Research
npolyzotis@google.com

Ki Hyun Tae, Steven Euijong Whang†
KAIST
{kihyun.tae, swhang}@kaist.ac.kr

***Abstract*—As machine learning (ML) systems become democratized, it becomes increasingly important to help users easily debug their models. However, current data tools are still primitive when it comes to helping users trace model performance problems all the way to the data. We focus on the particular problem of slicing data to identify subsets of the validation data where the model performs poorly. This is an important problem in model validation because the overall model performance can fail to reflect that of the smaller subsets, and slicing allows users to analyze the model performance on a more granular-level. Unlike general techniques (e.g., clustering) that can find arbitrary slices, our goal is to find interpretable slices (which are easier to take action compared to arbitrary subsets) that are large and problematic. We propose Slice Finder, which is an interactive framework for identifying such slices using statistical techniques. Applications include diagnosing model fairness and fraud detection, where identifying slices that are interpretable to humans is crucial.**

***Index Terms*—data slicing, model validation**

I. Introduction

Machine learning (ML) systems [1] are becoming more prevalent thanks to a vast number of success stories. However, the data tools for interpreting and debugging models have not caught up yet, and many important challenges exist to improve our model understanding after training [2]. One such key problem is to understand if a model performs poorly on certain parts of the data, hereafter referred to as a *slice*.

The problem is that the overall model performance can fail to reflect that of smaller data slices. Thus, it is important that the performance of a model is analyzed on a more granular level. While a well-known problem [3], current techniques to determine under-performing slices largely rely on domain experts to define important sub-populations (or at least specify a feature dimension to slice by) [4], [5]. Unfortunately, ML practitioners do not necessary have the domain expertise to know all important under-performing slices in advance, even after spending a significant amount of time exploring the data. An underlying assumption here is that the dataset is large to the extent that enumerating all possible data slices and validating model performance for each is not practical due to the sheer number of possible slices. Worse yet, simply searching for the most under-performing slices can be misleading because the model performance on smaller slices can be noisy, and without any safeguard, this leads to slices that are too small for meaningful impact on the model quality or that are false discoveries (i.e., non-problematic slices appearing as problematic). Ideally, we want to identify the largest and true problematic slices from the smaller slices that are not fully reflected on by the overall model performance metric.

In this paper, we propose Slice Finder, which efficiently discovers large possibly-overlapping slices that are both interpretable and problematic. A slice is defined as a conjunction of feature-value pairs where having fewer features is considered more interpretable. A problematic slice is identified based on testing of a significant difference of model performance metrics (e.g., loss function) of the slice and its counterpart. That is, we treat each problematic slice as a hypothesis and check that the difference is statistically significant, and the magnitude of the difference is large enough according to the effect size. In comparison, clustering similar examples has the drawback of not being interpretable.

In addition to testing, the slices found by Slice Finder can be used to evaluate model fairness or in applications such as fraud detection, business analytics, and anomaly detection.

II. Data Slicing Problem

A. Preliminaries

We assume a dataset D with n examples and a model h that needs to be tested. Following common practice, we assume that each example $x_F^{(i)}$ contains features $F = \{F_1, F_2, ..., F_m\}$ where each feature F_j (e.g., country) has a list of values (e.g., {US, DE}) or discretized numeric value ranges (e.g., {[0, 50), [50, 100)}). We also have a ground truth label $y^{(i)}$ for each example, such that $D = \{(x_F^{(1)}, y^{(1)}), (x_F^{(2)}, y^{(2)}), ..., (x_F^{(n)}, y^{(n)})\}$. The test model h is an arbitrary function that maps an input example to a prediction using F, and the goal is to validate if h is working properly for different subsets of the data.

A *slice* S is a subset of examples in D with common features and can be described as a predicate that is a conjunction of literals $\bigwedge_j F_j \ op \ v_j$ where the F_j's are distinct (e.g., country = DE $\wedge$ gender = Male), and op can be one of $=, <, \leq, \geq$, or $>$. For numeric features, we can discretize their values (e.g., quantiles or equi-height bins) and generate ranges so that they are effectively categorical features (e.g., age = [20,30)). Numeric features with large domains tend

*First author, work done at Google Research.
†Corresponding author, work done at Google Research and KAIST.

2375-026X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDE.2019.00139

to have fewer examples per value, and hence do not appear as significant. By discretizing numeric features into a set of continuous ranges, we can effectively avoid searching through tiny slices of minimal impact on model quality and group them to more sizable and meaningful slices.

We also assume a classification loss function $\psi(S, h)$ that returns a performance score for a set of examples by comparing h's prediction $h(x_F^{(i)})$ with the true label $y^{(i)}$. A common classification loss function is logarithmic loss (*log loss*).

B. Model Validation

We consider the model validation scenario of pointing the user to "problematic" slices where a single model performs relatively poorly on. That is, we would like to find slices where the loss function returns a significantly higher loss than the rest of the examples in D. At the same time, we prefer these slices to be large as well. For example, the slice country = DE may be too large for a model to perform significantly worse than other countries. On the other hand, the slice country = DE $\wedge$ gender = Male $\wedge$ age = 30 may have a high loss, but may also be too specific and thus small to have much impact on the overall performance of the model. Finally, we would like the slices to be interpretable in the sense that they can be expressed with a few literals. For example, country = DE is more interpretable than country = DE $\wedge$ age = 20-40 $\wedge$ zip = 12345.

Finding the most problematic slices is challenging because it requires a balance between how significant the difference in loss is and how large the slice is. Simply finding a slice with many classification errors will not work because there may also be many correct classifications within the same slice (recall that a slice is always of the form $\bigwedge_j F_j \; op \; v_j$). Another solution would be to score each slice based on some weighted sum of its size and difference in average losses. However, this weighting function is hard to tune by the user because it is not clear how size relates to loss. Instead, we envision the user to either fix the significance or size.

C. Problematic Slice as Hypothesis

We now discuss what we mean by significance in more detail. For each slice S, we define its counterpart S' as $D - S$, which is the rest of the examples. We then compute the relative loss as the difference $\psi(S, h) - \psi(S', h)$. Without loss of generality, we only look for positive differences where the loss of S is higher than that of S'.

A key question is how to determine if a slice S has a significantly higher loss than S'. Our solution is to treat each slice as a hypothesis and perform two tests: determine if the difference in loss is *statistically significant* and if the *effect size* of the difference is large enough. Using both tests is a common practice [6] and necessary because statistical significance measures the existence of an effect (i.e., the slice indeed has a higher loss than its counterpart) while the effect size complements statistical significance by measuring the magnitude of the effect (i.e., how large the difference is).

To measure the statistical significance, we use the hypothesis testing with the following null (H_o) and alternative (H_a) hypotheses:

$$H_o : \psi(S, h) \leq \psi(S', h)$$
$$H_a : \psi(S, h) > \psi(S', h)$$

Here both S and S' should be viewed as samples of all the possible examples in the world, including the training data and even the examples that the model might serve in the future. We then use Welch's t-test, which is used to test the hypothesis that two population means are equal.

To measure the magnitude of the difference between the distributions of losses of S and S', we compute the effect size ϕ, which is defined as follows:

$$\phi = \sqrt{2} \times \frac{\psi(S, h) - \psi(S', h)}{\sqrt{\sigma_S^2 + \sigma_{S'}^2}}$$

Intuitively, if the effect size is 1.0, we know that the two distributions differ by one standard deviation. According to Cohen's rule of thumb [7], an effect size of 0.2 is considered small, 0.5 is medium, 0.8 is large, and 1.3 is very large.

D. Problem Definition

For two slices S and S', we say that $S \prec S'$ if S precedes S' when ordering the slices by increasing number of literals, decreasing slice size, and decreasing effect size. Then the goal of Slice Finder is to identify *problematic slices* as follows:

Definition 1. *Given a positive integer k, an effect size threshold T, and a significance level α, find the top-k slices sorted by the ordering $\prec$ such that:*

(a) Each slice S has an effect size at least T,

(b) The slice is statistically significant,

(c) No slice can be replaced with one that has a strict subset of literals and satisfies the above two conditions.

The top-k slices do not have to be distinct, e.g., country = DE and education = Bachelors overlap in the demographic of Germany with a Bachelors degree.

III. System Architecture

Underlying the Slice Finder system is an extensible architecture (Figure 1) that combines automated data slicing and interactive visualization tools. Slice Finder loads the validation data set into a Pandas DataFrame. The DataFrame supports indexing individual examples, and each data slice keeps a subset of indices instead of a copy of the actual data examples. Slice Finder provides basic slice operators (e.g., intersect and union) based on the indices; only when evaluating the ML model on a given slice does Slice Finder access the actual data by the indices to test the model.

Once the data is loaded into a DataFrame, Slice Finder processes it to identify the problematic slices and allows the user to explore them. This process comprises three major components, as summarized below.

Slice Finder searches for problematic slices either by training a CART decision tree around misclassified examples

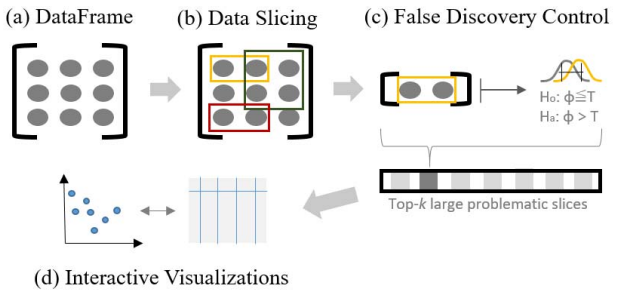


Fig. 1: The Slice Finder architecture.

or by performing a more exhaustive search on a lattice of slices. Both search strategies progress in a top-down manner until they find the top-k problematic slices. The decision tree approach materializes the tree model and traverses the tree to find problematic slices. In lattice searching, Slice Finder traverses a lattice of slices to find the slices. This top-down approach allows Slice Finder to quickly respond to new request queries that use different k values.

As Slice Finder searches through a large number of slices, some slices might appear problematic by chance (i.e., multiple comparisons problem [8]). Slice Finder controls such a risk by applying a marginal false discovery rate (mFDR) controlling procedure called α-investing [8], [9] in order to find statistically significant slices among a stream of slices.

Lastly, even a handful of problematic slices can be overwhelming to the user, since she may need to take action (e.g., deeper analyses or model debugging) on each slice. Hence, it is important to enable the user to quickly browse through the slices by their impacts (size) and scores (effect size). To this end, Slice Finder provides interactive visualization tools for the user to explore the recommended slices.

We now describe the two automated data slicing approaches used in Slice Finder: decision tree training and lattice searching. While lattice searching returns slices that satisfy Definition 1, decision tree training does not necessarily return all such slices as we explain below.

Decision Tree Training: To identify more interpretable problematic slices, we train a decision tree that can classify which slices are problematic. The output is a partitioning of the examples into the slices defined by the tree. For example, a decision tree could produce the slices $\{A > v,\ A \leq v \wedge B > w,\ A \leq v \wedge B \leq w\}$. For numeric features, this kind of partitioning is natural. For categorical features, a common approach is to use one-hot encoding where all possible values are mapped to columns, and the selected value results in the corresponding column to have a value 1.

To use a decision tree, we first identify the bottom-most problematic slices (leaves) with the highest effect size (i.e., highest error concentration). Then we can go up the decision tree to find larger (and more interpretable) slices that generalize the problematic slices and still have effect sizes larger than the threshold T.

The advantage of a decision tree is that it has a natural interpretation, since the leaves directly correspond to slices. The downside is that a decision tree only finds non-overlapping slices that are problematic. In addition, if a decision tree gets too deep with many levels, then it starts to become uninterpretable as well. The decision tree approach can be viewed as "greedy" because it optimizes on the classification results and is thus not designed to exhaustively find all problematic slices according to Definition 1.

Lattice Searching: The lattice searching approach considers a larger search space where the slices form a lattice, and problematic slices can overlap with one another. We assume that slices only have equality literals, i.e., $\bigwedge_i F_i = v_i$. In contrast to the decision tree training approach, lattice searching can be more expensive because it searches overlapping slices.

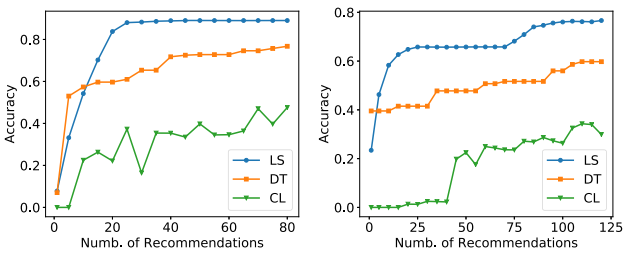
The input is the training data, a model, an effect size threshold T, and a significance level α. As a pre-processing step, Slice Finder takes the training data and discretizes numeric features. For categorical features that contain too many values (e.g., IDs are unique for each example), Slice Finder uses a heuristic where it considers up to the N most frequent values and places the rest into an "other values" bucket. The possible slices of these features form a lattice where a slice S is a parent of every S with one more literal.

Slice Finder finds the top-k interpretable and large problematic slices by traversing the slice lattice in a breadth-first manner using a priority queue. The priority queue contains the current slices being considered sorted by the ordering $\prec$. For each slice $\bigwedge_{i \in I} F_i = v_i$ that is popped, Slice Finder checks if it has an effect size at least T and is statistically significant using α-investing. If so, the slice is added to the top-k list. Otherwise, the slice is expanded where we generate each new slice by adding a literal, only if the resulting slice is not subsumed by a previously-identified problematic slice. The intuition is that any subsumed (expanded) slice contains a subset of the examples of its parent and is smaller with more filter predicates (less interpretable); thus, we do not expand larger and already problematic slices. By starting from the base slices whose predicates are single literals and expanding only non-problematic slices with one additional literal at a time (i.e., top-down search from lower order slices to higher order slices), we can generate a superset of all candidate slices. This process repeats until the top-k slices have been found or there are no more slices to explore.

IV. Experiments

In this section, we evaluate the two Slice Finder approaches – decision tree (DT) and lattice search (LS) – with a baseline that clusters points using the k-means algorithm. We trained a random forest classifier to predict whether the income exceeds \$50K/yr based on UCI census data [10]. We also assume that all slices are statistically significant. More details and experiments can be found in our technical report [11].

Accuracy: An important question to answer is whether Slice Finder can indeed find the largest most problematic slices, in a user's point of view. Unfortunately for the real datasets, we do not know what are the true problematic slices, which makes our evaluation challenging. Instead, we *add new*



(a) Synthetic data (b) UCI census data

Fig. 2: Accuracy comparison of finding problematic slices using (a) synthetic data and (b) real data.

Slice	Size	Effect Size
Census Income Classification		
Marital Status = Married-civ-spouse	14065	0.58
Relationship = Husband	12463	0.52
Relationship = Wife	1406	0.46
Capital Gain = 3103	94	0.87
Capital Gain = 4386	67	0.94
Capital Gain = 8614	52	0.42
Capital Loss = 1485	45	0.77
Capital Gain = 10520	43	0.4
Capital Gain = 14344	26	0.45
Capital Loss = 2258	25	0.69

TABLE I: Top-10 largest problematic slices.

problematic slices by randomly perturbing labels and focus on finding those slices. While Slice Finder may find both new and existing problematic slices, our evaluation will only be whether Slice Finder finds the new problematic slices.

We first generate a simple synthetic dataset where the generated examples have two discretized features F_1 and F_2 and can be classified into two classes – 0 and 1 – perfectly. We make the model use this decision boundary and do not change it further. We then add some noise and random possibly-overlapping slices of the form $F_1 = A$, $F_2 = B$, or $F_1 = A \wedge F_2 = B$. For each slice, we flip the labels of the examples with 50% probability.

Figure 2(a) shows the accuracy comparison of the LS, DT, and CL algorithms on synthetic data. As the number of recommendations k increases, LS gradually performs better than DT because LS is able to better pinpoint the problematic slices including overlapping ones while DT tends to find larger slices, which initially have high recall, but low precision. For CL, we only evaluated the clusters with effect sizes at least T. Even so, the accuracy is much lower than LS and DT. Figure 2(b) shows similar comparison results on the UCI census dataset where we generated new problematic slices on top of the existing data. In comparison to the synthetic data, the UCI census data may also have problematic slices, which we do not evaluate. Hence, the accuracies of the three algorithms tend to be lower than the synthetic data results.

Interpretation: Users prefer seeing data slices that are easy to understand with a few common features. In other words, the performance metrics or presenting a cluster of misclassified examples are not sufficient to understand and describe the model behavior. In practice, a user often goes through all the misclassified examples (or clusters of them) manually to describe and understand the problem. To this end, Slice Finder can be used as a pre-processing step to quickly identify data slices where the model might be biased or failing, and the slices are easy to describe with a few common features. Table I shows top-10 problematic slices ordered by $\prec$ in the Census Income Classification; the slices are easy to interpret with a few number of common features. We see that the Marital Status = Married-civ-spouse slice has the largest size as well as a large effect size, which indicates that the model can be improved for this slice. It is also interesting to see that the model fails for the people who are husbands or wives, but not for other relationships: own-child, not-in-family, other-relative, and unmarried. We also see slices with high capital gain or loss tend to be problematic in comparison to the common case where the value is 0.

V. Conclusion

We have proposed Slice Finder for efficiently finding interpretable, large, and significant slices. The techniques are relevant to model validation in general, but also to model fairness and fraud detection where human interpretability is critical to understand model behavior. We have proposed two methods for automated data slicing for model validation: decision tree training, which is efficient and finds slices defined as ranges of values, and slice lattice search, which can find overlapping slices and is more effective for categorical features. We also provide an interactive visualization front-end to help users quickly browse through problematic slices.

Acknowledgments

Steven Euijong Whang and Ki Hyun Tae were supported by a Google AI Focused Research Award and by the Engineering Research Center Program through the National Research Foundation of Korea (NRF) funded by the Korean Government MSIT (NRF-2018R1A5A1059921).

References

[1] D. Baylor *et al.*, "Tfx: A tensorflow-based production-scale machine learning platform," in *KDD*, 2017, pp. 1387–1395.
[2] F. Doshi-Velez and B. Kim, "Towards A Rigorous Science of Interpretable Machine Learning," *ArXiv e-prints*, Feb. 2017.
[3] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin *et al.*, "Ad click prediction: a view from the trenches," in *KDD*, 2013, pp. 1222–1230.
[4] M. Kahng, D. Fang, and D. H. P. Chau, "Visual exploration of machine learning results using data cube analysis," in *HILDA*. ACM, 2016, p. 1.
[5] "Introducing tensorflow model analysis," https://medium.com/tensorflow/introducing-tensorflow-model-analysis-scaleable-sliced-and-full-pass-metrics-5cde7baf0b7b, 2018.
[6] G. M. Sullivan and R. Feinn, "Using effect size-or why the p value is not enough," *Journal of Graduate Medical Education*, vol. 4, no. 3, pp. 279–282, 2012.
[7] J. Cohen, "Statistical power analysis for the behavioral sciences." 1988.
[8] D. Foster and B. Stine, "Alpha-investing: A procedure for sequential control of expected false discoveries," *Journal of the Royal Statistical Society Series B (Methodological)*, vol. 70, no. 2, pp. 429–444, 2008.
[9] Z. Zhao, L. D. Stefani, E. Zgraggen, C. Binnig, E. Upfal, and T. Kraska, "Controlling false discoveries during interactive data exploration," in *SIGMOD*, 2017, pp. 527–540.
[10] R. Kohavi, "Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid." in *KDD*, vol. 96, 1996, pp. 202–207.
[11] Y. Chung, T. Kraska, N. Polyzotis, and S. Euijong Whang, "Slice Finder: Automated Data Slicing for Model Validation," *ArXiv e-prints*, Jul. 2018.