Slicing Data Repositories

B-Tech Project 2021

Harin Jani
201701421
DA-IICT
Gandhinagar, India
201701421@daiict.ac.in

Raj Patel
201701422
DA-IICT
Gandhinagar, India
201701422@daiict.ac.in

Shubh Desai
201701466
DA-IICT
Gandhinagar, India
201701466@daiict.ac.in

***Abstract*—Machine Learning Systems, Big Data Analytics and Artificial Intelligence are employed everywhere. However, the model validation and debugging technologies become necessary for improving models especially for machine learning experts less equipped with knowledge of the specific domain. Here, we try to identify the interpreted sets of data where a given model fails to perform accurately. We have tested the developed tool on datasets from multiple different domains.**

***Index Terms*—data slicing, unsupervised machine learning**

I. Introduction

A. Motivation

Machine Learning has become ubiquitous with time. It has been employed to address a large number of problems. However, effective development of machine learning models also calls for effective debugging and evaluation. One such scenario is formed by a microscopic understanding of the subsets of the data where the model fails to perform accurately. The overall model performance can fail to reflect that of smaller data subsets. Thus, it is important to analyze the performance of a model on a more granular level.

Supervised Machine Learning Algorithms are the class of problem solutions that deal with trying to model on the basis of resemblance to ground truth data. These algorithms deal with an eclectic group of domains and many a times, machine learning experts do not have specialised knowledge of these domains which inhibits their ability to develop solutions effectively. Identifying problematic subsets of data will definitely help in such a scenario.

Let us think of a subset of a data with specific constraints enforced on parameters as a *slice*. If we can predict these slices, it can definitely aid in improving the model.

B. Objective

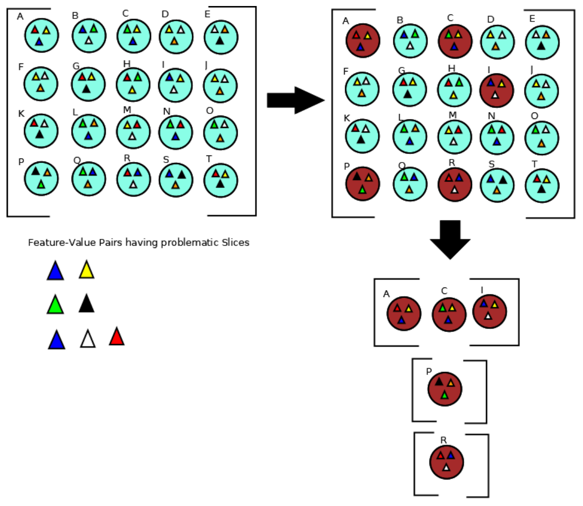
This project aims at identifying the problematic *slices*, i.e., subsets of data that have less accuracy for the underlying model. However, certain aspects of the slices must be considered.

First, the slices must not be selected randomly. Merely taking chunks of data where the model performs poorly will not be helpful. There must be a level of *interpretability* in the slices. Thus, we must be able to attach a definite set of feature-value pairs to each slice, which can help a non-domain expert "interpret" the shortcomings of the model.

Second, the slice must not be too small to negligibly affect the overall model. Thus, we need to have a measure that considers the "effective size" of the *slice*. Thus, the slice must have a minimum threshold effect size.

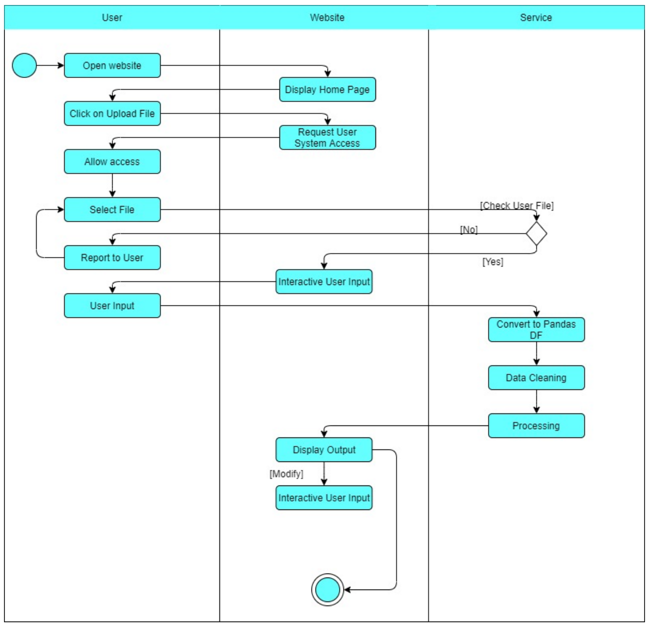
Finally, the tool must be made accessible for easy usage through an ergonomic interface. Thus, we aim to create a website which displays the results of the model validation and also provides a list of the most problematic slices. Moreover, we have tested the tool for a variety of datasets from different domains to check its integrity over different domains.

C. Generic Data Slicing



The above schematic diagram gives a rough idea about how the problematic slices are found by the data slicer. First we find the key value pairs which contribute to the problematic slices. From the data set, the data values which match the key value pairs are selected. From all the problematic slices the slices that match the effective size criteria are obtained in descending order.

D. Activity Diagram



This is the activity diagram of the entire process starting from uploading the data file to displaying problematic data slices to the user.

II. Related Work

In the whole process there has been extensive importance put on interpretability of the slices found as problematic. The interpretability of slices is kind of self explanatory but when the tool is applied for critical operations such as fraud detection and non-discrimination, the interpretability of the slices and why they are termed as interpretable becomes way more important. This whole thing is described and discussed in the paper [2].

We have successfully identified problematic slices but slice-based learning, a programming model for improving performance on application-critical slices is also necessary. It has been incorporated in the paper [3]. Specifically, it focuses on 1) an intuitive interface for identifying such slices and 2) Describing a modeling approach for improving slice performance that is agnostic to the underlying architecture. This work is useful in the context of an emerging class of programming models slice-based learning is a paradigm that sits on top of traditional modeling approaches in machine learning systems.

The writers of the paper [9] introduced the challenge of improving slice-specific performance without damaging the overall model quality, and proposed the first programming abstraction and machine learning model to support these actions. They demonstrated that the model could be used to push the state-of-the-art quality. In their analysis, they can explain consistent gains in the Slice-based Learning paradigm, as their attention mechanism has access to a rich set of deep features, whereas existing weak supervision paradigms have no way to access this information. They view this work in the context of programming models that sit on top of traditional modeling approaches in machine learning systems.

III. User Stories

A. User Story 1

As a user, I want to upload a file, so that it can be processed for finding problematic slices.

B. User Story 2

As a user, I would like to set the number of problematic slices to be found, to customize output as per requirement.

C. User Story 3

As a user, I would like to set the threshold effect size, to customize output as per requirement.

D. User Story 4

As a user, I would like to download the slices obtained by the Slice finder tool.

IV. Approach and Algorithms

We define a slice in machine learning jargon as a conjunction of feature-value pairs. Fewer feature-value pairs implies that the particular slice is more interpretable. To identify whether a particular slice is problematic or not, it needs to be tested on certain metrics, one of them is log-loss function. We have to calculate the metric for the slice and its counterpart as well and then decide based on some certain predefined criteria to whether consider that slice as problematic or not by comparing both the results of slice and its counterpart.

To identify the most problematic slices, simple searching is not optimal and in many cases it may be possible that we might get a wrong interpretation of the overall model and predictions. Primary reason for this is that model performance on smaller slices may be noisy. Thus by simply searching for problematic slices, we may consider that these smaller slices are actually the answer, which is not correct, since those slices are too small for creating an impact on overall model performance. Thus we are getting non-problematic slices as problematic slices. So our problem boils down to identifying the large and truly problematic slices rather than the smaller slices with negligible impact on overall model performance.

For doing this, the procedure is to consider each slice as the hypothesis and then check that the result of that particular slice is significantly worse than the overall data or not, and whether the slice is large enough for creating an impact on overall model quality or not. It basically means, we are looking for the considerably large slices where the loss function returns a much larger value for the slice than its counterpart.

To put the understanding in concrete terms, for each slice S, its counterpart S' id defined as D - S, which is the rest of the data. We then compute the difference between the loss function of both, the slice and its counterpart,meaning ψ(S, h) - ψ(S', h). What we want are the slices where the difference between the loss function is positive and that value is also statistically large. Here, D is the actual data, ψ is the loss-function and h is the hypothesis.

Our goal is to find top K slices from the data provided by the user following necessary criteria such as slices should be interpretable and large enough as mentioned before. For that, first of all we define an ordering $\prec$. For two slices S and S'(here S' is another slice, not the counterpart of S as mentioned before), we say that S $\prec$ S' according to this ordering, if S precedes S' when ordering the slices by increasing number of literals, decreasing slice size, and decreasing effect size. If observed carefully, this is what our task is, to obtain the most interpretable slices(taken care by ordering in increasing number of literals, since less literals imply more interpretable slices), and ordering by decreasing size implies the largest slice will be the first in this type of ordering. Given a positive integer K, an effect size threshold T, and a significance level α, our goal is to 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 slice finder tool is designed to validate and process large data sets for machine learning models. The tool loads the validation data set into a pandas data frame. Pandas library also provides a number of options to deal with dirty data and missing values and hence data cleaning is more efficient using pandas data frame. The clean data is then sent to find the problematic slices according to the parameters specified by the user.

After loading the data into the pandas data frame, slice finder processes it to find the most problematic slices as per user's direction. There are many possible algorithms for finding the most problematic slices, i.e. Clustering Algorithm, Decision Tree Algorithm, Lattice Search Algorithm, etc. Out of which clustering is the most basic of them. Decision Tree Algorithm and Lattice Search Algorithm, both find the slices in breadth first traversal in a top down manner, though differently which is explained further. All the three algorithms have advantages of their own as well as disadvantages over others.

Clustering Algorithm: As the name suggests, in this algorithm, similar examples are clustered together and considered as separate slices. Then on each of the slices in the form of clustered data, test model is executed. If the test model does not perform with desired accuracy required by the user, deeper analysis of that particular slice is performed. Clustering is one of the most fundamental algorithms for slice finding. One of the major drawback of this algorithm is low interpret-ability of slices. This is not desired, as our goal is to find large and interpretable problematic slices. The other drawback is that in this algorithm, the number of clusters needs to be defined at the beginning which is hard to tune and it has an impact on the overall result.

Decision Tree Algorithm: In decision algorithm, the found slices are much more interpretable than in the case of Clustering Algorithm. The problematic slices are the leaf nodes of the constructed decision tree from the actual data. The tree keeps getting divided until no more division is possible or the required slices have already been found. The slices on each level are sorted by the $\prec$ ordering and then filtered based on whether they are large enough and statistically significant or not. The advantage over clustering algorithm is obvious which is that the slices are more interpretable. If the k slices are found without the whole tree being generated, the algorithm can be terminated midway to save computational power. Thus it is more efficient. Another drawback of this algorithm is that if a node gets divided into two children, and there exists a slice which is a combination of both, then it is certain that the algorithm will not be able to classify it because of the very structure of the algorithm.

Lattice Searching Algorithm: To avoid the drawbacks of Decision Tree Algorithm, Lattice Search Algorithm considers a larger searching space. This algorithm can deal with the scenario where the slices may be overlapping, which was not the case with Decision Tree Algorithm. But the trade off is that in contrast to the decision tree training approach, lattice searching can be more expensive because it searches overlapping slices. Lattice searching performs a breadth-first search and efficiently identifies problematic slices. The drawback of this algorithm is that since it also tries to find overlapping slices, it is computationally very expensive. It is almost impossible to implement the algorithm under normal day to day computer. This algorithm needs to be implemented on a machine with better specifications using a parallel algorithm strategy.

V. Implementation

The implementation has been approached in two specific sections which have been integrated as one. The first section corresponds to the Unsupervised Machine Learning Algorithm which serves the main purpose of the tool. The second section corresponds to the Website Interface which helps provide an ergonomic access to the Data Slicer. As mentioned before, Clustering and Lattice Search Algorithm have different issues with their implementation, therefore **Decision Tree Algorithm** has been used for our purpose and has been explained later.

A. *Algorithm*

The algorithm for the problem is as follows:

Selecting the Root Node: The entire dataset *D* is considered. The statistical t - testing [8] is used to identify the correlation between a column (i.e. a feature) to the model utilized. Thus, for our purpose, we need to consider a slice *S* and its counterpart *S' = D - S*. We assume one to be the null hypothesis (assumed as contributing accurately to the model and verified later) and conduct hypothesis testing to identify whether the other is problematic.

An important thing to note is that the root node parameter identification is used for each sub-tree. Thus we take a node, split it into left child and right child followed by identifying root node parameter for both and continuing the process recursively for the left and right sub-tree.

Attribute Selection Measure: The root node of any sub-tree of the decision tree is selected using an Attribute Selection Measure(ASM). Various measures for the same can be employed. Popular ones are Gini impurity, Information gain and Gain Ratio. We have utilized the Entropy and Information Gain Measure.

$$Entropy = -\Sigma_i p_i log_2(p_i)$$

$$\begin{aligned} InformationGain = {} & Entropy_{Parent} \\ & - \frac{Size_{Left}}{Size_{Parent}} * Entropy_{Left} \\ & - \frac{Size_{Right}}{Size_{Parent}} * Entropy_{Right} \end{aligned}$$

Understanding one Node splitting at a tree level: We find out a root node and then split it on the basis of a feature putting a constraint on some value. Let us consider a parameter "Age". If that is identified using hypothesis testing, we loop over various values of "Age". For each value, we calculate the Information Gain. For whatever value the gain maximises, we undertake splitting.

Thus, for every column at a given node *N* we first identify a column using hypothesis testing and then split it at different values to divide the dataset into two subsets being the left and the right child.

Termination of this recursion: This splitting can keep continuing till we reach individual rows. Hence, we must enforce a stopping condition to avoid splitting the data too much in order to get larger slices and decrease computational and time complexity costs. Thus, we enforce a limit on the depth of the Decision Tree (in our case, we have restricted it to 3) to constrain the algorithm.

The algorithm also constrains on the number of slices (which we have set to 20 to adjust as per computational power available and it must be greater than 10 that we provide as the max output possible in the Web Tool.)

Getting the required output: Creating the decision tree does not still get the required output, since we require the K most problematic slices. Thus we need to take each node and quantitatively evaluate each node, i.e., slice if it can be recommended. For the purpose we use the log - loss metric and effect size(ϕ).

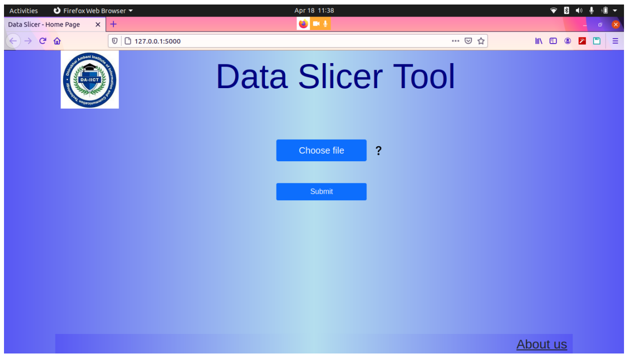
The Tool returns all or K most problematic slices depending on the number of problematic slices we found with effect size greater than the minimum effect size.

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

For each slice, we check whether the effect size is greater than the minimum effect size we provided as an input. From the list of slices we get, we sort the slices on the basis of size(number of rows) and return the top K slices.

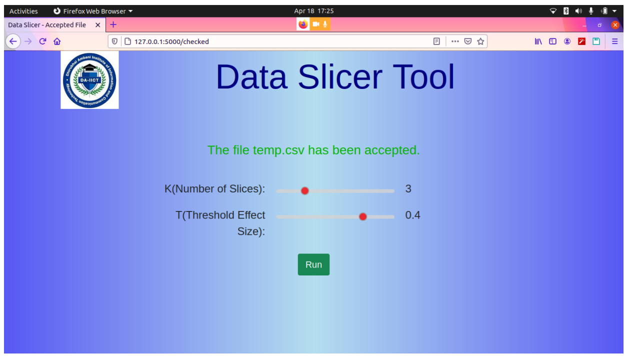
B. *Web Application*

To provide ease of use for the user, we have created a website which will accept the CSV file as an input and provide the problematic slices according to the parameters set by the user.

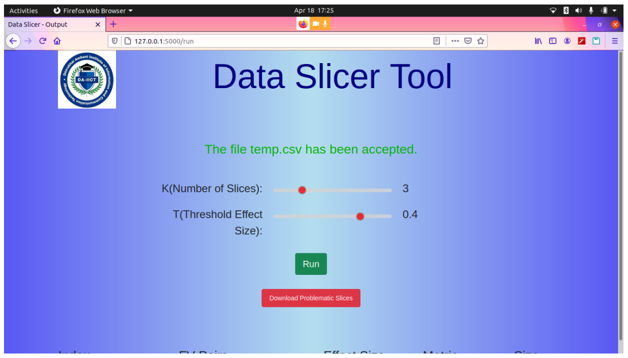


This is the homepage of the website. Here the user need to upload a CSV file containing the data for which the slices need to be found. There are some restrictions on the format of the file. First of all there must be a header row describing the data, since without that slices can not be categorised. There must be at least three columns with one column labelled as "Predicted" and one column labelled as "Target". "Target" is the column containing the correct classification data with which "Predicted" data needs to be compared.

If the file format is not meeting with any of these conditions, an appropriate message will be displayed indicating the same.


Data Slicer Tool
The file temp.csv has been accepted.
K(Number of Slices): 3
T(Threshold Effect Size): 0.4
Run

If the file format is supported with the conditions mentioned above, user will be redirected to the above page, where two parameters need to be selected from the provided slider. K(Number of slices) and T(Threshold Effect Size). After setting the parameters, user can click on "Run" button to see the result.

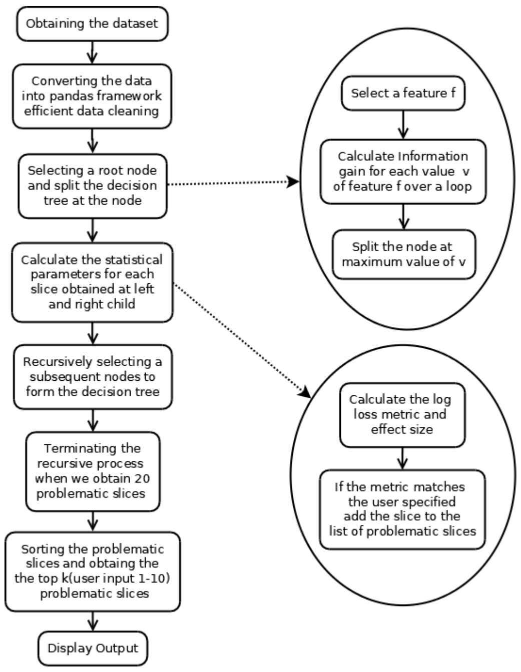

Data Slicer Tool
The file temp.csv has been accepted.
K(Number of Slices): 3
T(Threshold Effect Size): 0.4
Run
Download Problematic Slices

On the next page, all the options will be available along with the "Download Slices" option, from where we can download the slices as a CSV file. If the slices for the same dataset are needed but with different "K" and "T" parameters, it can be set from here rather than repeating the whole process.

C. Psuedocode

```
1) Request the input .csv file F from the user U.
2) Check if F has columns "Target" and "Prediction".
3) If not, report to user and move back to 1.
4) Request U to provide Min Effect Size $\phi_{min}$ and Number of Slices $K$.
5) Extract the dataset D from F.
6) Refine D by removing rows with NaN values.
7) Consider D as root node.
8) Select a feature F for splitting using the statistical testing.
9) We loop over various values v of F.
10) For each value, calculate information gain IG for each v of F.
11) Find value $v_{max}$ for which IG maximises.
12) Conduct splitting on $value == v$ and $value != v$
13) We get the left child L and the right child R. For each of them, repeat steps 7 to 12 by considering each as root node.
14) If after splitting, we have reached the limiting depth of the decision tree, we stop splitting.
15) Compute the size, log loss metric and effect size for each slice.
16) Filter the list to obtain slices with effect size greater than $\phi_{min}$.
17) Sort the filtered list in decreasing order on the basis of size.
18) Obtain the list of top K slices.
19) Compile the list into a .csv file F'.
20) Allow U to download F'.
```

D. Schematic Diagram of Implementation


Obtaining the dataset
Converting the data into pandas framework efficient data cleaning
Selecting a root node and split the decision tree at the node
Calculate the statistical parameters for each slice obtained at left and right child
Recursively selecting a subsequent nodes to form the decision tree
Terminating the recursive process when we obtain 20 problematic slices
Sorting the problematic slices and obtaing the the top k(user input 1-10) problematic slices
Display Output
Select a feature f
Calculate Information gain for each value v of feature f over a loop
Split the node at maximum value of v
Calculate the log loss metric and effect size
If the metric matches the user specified add the slice to the list of problematic slices

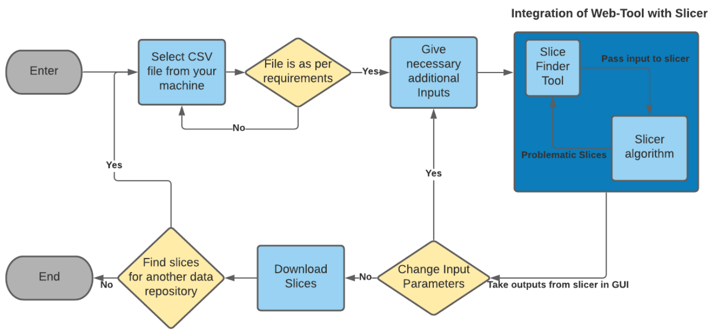
E. Technology

We used Visual Studio Code as the IDE, Git for Version Control and GitHub to manage the same.

For the algorithm, we have employed Python 3 as the language of implementation along with various libraries such as numpy, pandas, sklearn, matplotlib, and pickle. We have used Jupyter Notebook for development.

For the website, we have used HTML5, CSS3, JavaScript and Bootstrap5 for the implementation of the website front-end. We have employed Flask for the website back-end.

F. Abstract Diagram of overall Tool


Integration of Web-Tool with Slicer
Enter
Select CSV file from your machine
File is as per requirements
Yes
No
Give necessary additional Inputs
Slice Finder Tool
Pass input to slicer
Slicer algorithm
Problematic Slices
Yes
Take outputs from slicer in GUI
Change Input Parameters
No
Download Slices
Find slices for another data repository
Yes
No
End

VI. Execution and Testing

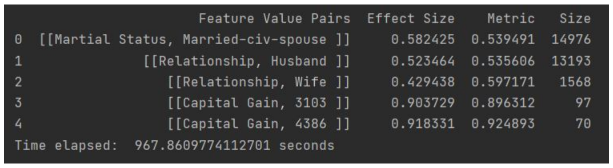
A. Primary Testing

Testing of such a tool is not a trivial task. There is no available platform from where we can directly confirm the results. So for testing purpose, we have taken datasets of different sizes and corrupted the datasets on purpose. To elaborate why we need to do this, the whole procedure of slice finding must be understood. We are essentially finding the feature-value pairs for which the model performs below a specific benchmark. What we can do is we can select some feature value pair beforehand which are performing well. If we change the predicted value of these feature-value pairs, what we are doing essentially is converting these non-problematic slices to problematic. And after the datasets has been corrupted and passed through the tool, we must get those slices as answer.

There are two testing procedures followed to verify the correctness of the tool:

1) In the main referred research paper [1], the Census Income Dataset from the UCI Machine Learning Repository has been utilized. The Dataset lists a group of people (48842 instances) with 14 features and their income has been classified as either ”$>= 50k$” or ”$< 50k$” [4]. For the classification problem, the Random Forest Classifier approach has been employed.

   This provided with a set of predictions, over which the tool was employed using the Decision Tree algorithm in [1]. We aim to do the same and replicate the results as shown in the same. Our results are shown as below:

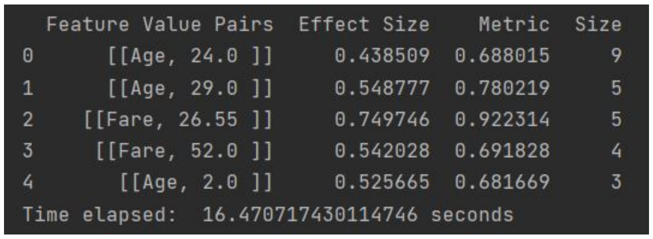


```
                        Feature Value Pairs  Effect Size    Metric   Size
0  [[Martial Status, Married-civ-spouse ]]     0.582425  0.539491  14976
1             [[Relationship, Husband ]]     0.523464  0.535606  13193
2                [[Relationship, Wife ]]     0.429438  0.597171   1568
3                [[Capital Gain, 3103 ]]     0.903729  0.896312     97
4                [[Capital Gain, 4386 ]]     0.918331  0.924893     70
Time elapsed:  967.8609774112701 seconds
```

2) The second method that we have employed for testing the tool is by intentionally adding noise to the dataset and verifying if the same is reflected in the list of slices obtained as a result. Now, it becomes quite important to employ a dataset and a predicted model alongside. This is because, in its actual usage, the tool will not have access to the model, but only the ground truth data and the predicted data.

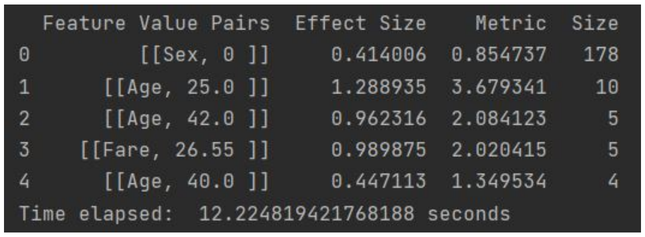
   For the same purpose, the Titanic Dataset for accident prediction has been employed [5]. The dataset represents people dying or surviving the titanic disaster on the basis of various parameters. It has a list of people (1309 instances) with 10 columns of feature-value pairs. The referred link also provides a python notebook which uses a Random Forest Classifier algorithm solution for the problem.

   The tool was first executed to observe results in the pristine dataset. The results observed were as follows:



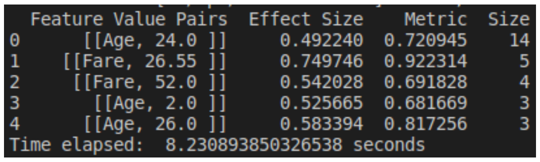
```
  Feature Value Pairs  Effect Size    Metric  Size
0      [[Age, 24.0 ]]     0.438509  0.688015     9
1      [[Age, 29.0 ]]     0.548777  0.780219     5
2    [[Fare, 26.55 ]]     0.749746  0.922314     5
3     [[Fare, 52.0 ]]     0.542028  0.691828     4
4       [[Age, 2.0 ]]     0.525665  0.681669     3
Time elapsed:  16.470717430114746 seconds
```

   For the intentional noise, we inverted the ground truth value of the dataset for 25 rows such that for each row the ”Sex” column value was set to zero. The number 25 was selected as per the size of slices observed in the noise-free slice execution.



```
  Feature Value Pairs  Effect Size    Metric  Size
0         [[Sex, 0 ]]     0.414006  0.854737   178
1      [[Age, 25.0 ]]     1.288935  3.679341    10
2      [[Age, 42.0 ]]     0.962316  2.084123     5
3    [[Fare, 26.55 ]]     0.989875  2.020415     5
4      [[Age, 40.0 ]]     0.447113  1.349534     4
Time elapsed:  12.224819421768188 seconds
```

   For another instance of intentional noise, we changed the value of age from 29 to 24 in 5 rows. This should have eliminated the second slice and should have enlarged the first slice by the size of slice 2 as per the output of the pristine dataset. Satisfactory results as expected were observed:



```
  Feature Value Pairs  Effect Size    Metric  Size
0      [[Age, 24.0 ]]     0.492240  0.720945    14
1    [[Fare, 26.55 ]]     0.749746  0.922314     5
2     [[Fare, 52.0 ]]     0.542028  0.691828     4
3       [[Age, 2.0 ]]     0.525665  0.681669     3
4      [[Age, 26.0 ]]     0.583394  0.817256     3
Time elapsed:  8.230893850326538 seconds
```

Multiple other instances of intentional noise were further added. For the same, satisfactory results were observed.

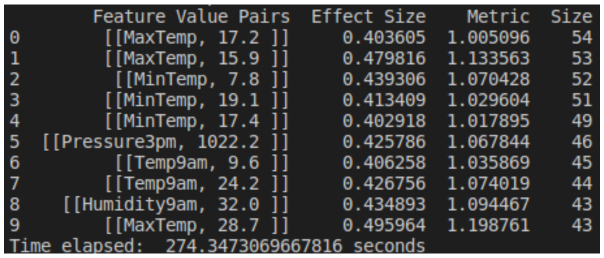
Hence from the above two scenarios, we can safely conclude that the Tool must be producing correct results.

B. Further execution

The tool was then further executed on a variety of datasets to observe results on multi-disciplinary data. The same strategy was employed. A dataset along with an already implemented model are taken to produce predictions. Now, the ”Target” data and the ”Prediction” data are compared to estimate problematic slices. Results obtained were as follows:

1) **Meteorological:** The Rain in Australia dataset [6] has been employed that uses various set of parameters observed over a period of 10 years (145460 instances) in Australia to see if it rains the next day. Parameters such as Humidity, Rain Today, Status of sky and others were considered. The data was modelled as a Classification Problem using the Random Forest Classification Algorithm.

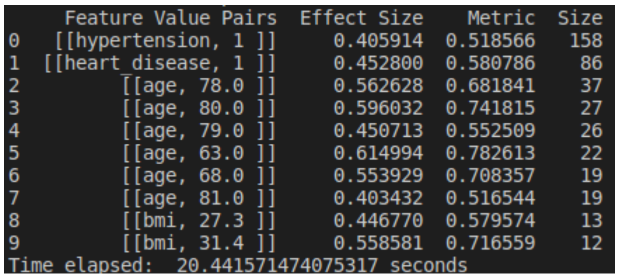
   This can be crucial in identifying what parameters are not modelled correctly or appropriately by the used algorithm.



```
        Feature Value Pairs  Effect Size    Metric  Size
0        [[MaxTemp, 17.2 ]]     0.403605  1.005096    54
1        [[MaxTemp, 15.9 ]]     0.479816  1.133563    53
2         [[MinTemp, 7.8 ]]     0.439306  1.070428    52
3        [[MinTemp, 19.1 ]]     0.413409  1.029604    51
4        [[MinTemp, 17.4 ]]     0.402918  1.017895    49
5  [[Pressure3pm, 1022.2 ]]     0.425786  1.067844    46
6         [[Temp9am, 9.6 ]]     0.406258  1.035869    45
7        [[Temp9am, 24.2 ]]     0.426756  1.074019    44
8    [[Humidity9am, 32.0 ]]     0.434893  1.094467    43
9        [[MaxTemp, 28.7 ]]     0.495964  1.198761    43
Time elapsed:  274.3473069667816 seconds
```

2) **Medical:** This dataset [7] is used to predict whether a patient is likely to get a stroke based on 10 input parameters (5110 instances). Parameters such as gender, age, various diseases and smoking status were considered. The data was modelled as a Classification Problem using the Logistic Regression Algorithm.

   Given the sensitiveness of the problem, it becomes extremely important to identify where the algorithm fails and what are the parameters where the employed model is susceptible to failure.



```
   Feature Value Pairs  Effect Size    Metric  Size
0   [[hypertension, 1 ]]     0.405914  0.518566   158
1  [[heart_disease, 1 ]]     0.452800  0.580786    86
2         [[age, 78.0 ]]     0.562628  0.681841    37
3         [[age, 80.0 ]]     0.596032  0.741815    27
4         [[age, 79.0 ]]     0.450713  0.552509    26
5         [[age, 63.0 ]]     0.614994  0.782613    22
6         [[age, 68.0 ]]     0.553929  0.708357    19
7         [[age, 81.0 ]]     0.403432  0.516544    19
8         [[bmi, 27.3 ]]     0.446770  0.579574    13
9         [[bmi, 31.4 ]]     0.558581  0.716559    12
Time elapsed:  20.441571474075317 seconds
```

VII. Conclusion

We have proposed Slice Finder as a tool for efficiently and accurately finding large, significant, and interpretable slices. The techniques are relevant to model validation in general, but also to model fairness and fraud detection where human interpret-ability is critical to understand model behavior. We have proposed and implemented Decision Tree Algorithm for slice finding: which finds non-overlapping slices. We have also created a website to provide an end to end interface to the user.

VIII. Future Scope

Till now, the Slice Finder Tool is made to find slices in Classification models. We can modify the functionalities so that it also works on regression models. We are also planning for implementing Lattice Searching Algorithm, for which powerful machines are required through which parallel algorithms can be implemented efficiently. We can also add some more interesting features in the website for providing even more detailed and interpretable results.

Acknowledgment

First of all, we would like to express our humble gratitude towards our BTP(B-Tech Project) mentor and supervisor Professor Jaypraksah Lalchandani sir, without whose guidance nothing of this would have been possible. He guided us throughout the course of this project and made sure we were not stuck at any point in time. Apart from the immense technical knowledge, there were many other things which we could learn during this period. We could follow a definite plan because of his guidance and regular suggestions, without which this could have taken a lot longer.

We would also like to express our gratitude to our university DA-IICT for providing us the opportunity to work on such an interesting and important topic as a part of our BTP(B-Tech Project). We are grateful for this experience which will surely prove to be a major milestone in our respective careers.

Appendix

Execution Environment

The time provided in the above images have been executions on a specific run-time environment and are subjective under the same. They have been executed under the following configuration:

- Processor: Intel® Core™ i5-4200U CPU @ 1.60GHz × 4
- OS: Windows 8.1 Pro
- RAM: 6 GB

References

[1] Automated Data Slicing for Model Validation:A Big data - AI Integration Approach by Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, Ki Hyun Tae, Steven Euijong Whang *IEEE, arXiv:1807.06068*

[2] Towards A Rigorous Science of Interpretable Machine Learning Finale Doshi-Velez and Been Kim *IEEE, arXiv:1702.08608*

[3] Slice-based Learning Vincent S. Chen, Sen Wu, Zhenzhen Weng, Alexander Ratner, Christopher Ré *https://www.snorkel.org/blog/slicing*

[4] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[5] Predictions Titanic *https://www.kaggle.com/ovimaha/predictions-titanic*

[6] Rainfall Prediction with 6 Machine Learn Algo 98 *https://www.kaggle.com/midouazerty/rainfall-prediction-with-6-machine-learn-algo-98*

[7] Best Score with EDA *https://www.kaggle.com/nikitasenpai/best-score-with-eda?scriptVersionId=60353398*

[8] https://en.wikipedia.org/wiki/Student

[9] Slice-based Learning: A Programming Model for Residual Learning in Critical Data Slices by Vincent S. Chen, Sen Wu, Zhenzhen Weng, Alexander Ratner, Christopher Ré of Stanford University *https://arxiv.org/pdf/1909.06349.pdf*