Slice Tuner: A Selective Data Collection Framework for Accurate and Fair Machine Learning Models

Ki Hyun Tae
KAIST
kihyun.tae@kaist.ac.kr

Steven Euijong Whang
KAIST
swhang@kaist.ac.kr

ABSTRACT

As machine learning becomes democratized in the era of Software 2.0, one of the most serious bottlenecks is collecting enough labeled data to ensure accurate and fair models. Recent techniques including crowdsourcing provide cost-effective ways to gather such data. However, simply collecting data as much as possible is not necessarily an effective strategy for optimizing accuracy and fairness. For example, if an online app store has enough training data for certain slices of data (say American customers), but not for others, collecting more American customer data will only bias the model training. Instead, we contend that one needs to *selectively collect data* and propose Slice Tuner, which collects possibly-different amounts of data per slice such that the model accuracy and fairness on all slices are optimized. At its core, Slice Tuner maintains *learning curves* of slices that estimate the model accuracies given more data and uses convex optimization to find the best data collection strategy. The key challenges of estimating learning curves are that they may be inaccurate if there is not enough data, and there may be dependencies among slices where collecting data for one slice influences the learning curves of others. We solve these issues by iteratively and efficiently updating the learning curves as more data is collected. We evaluate Slice Tuner on real datasets using crowdsourcing for data collection and show that Slice Tuner significantly outperforms baselines in terms of model accuracy and fairness, even for initially small slices. We believe Slice Tuner is a practical tool for suggesting concrete action items based on model analysis.

1. INTRODUCTION

In the era of Software 2.0 [4], machine learning is becoming democratized where successful applications range from recommendation systems to self-driving cars. Software engineering itself is going through a fundamental shift where trained models are the new software, and data becomes a first-class citizen on par with code [25]. Training a model to use in production requires multiple steps including data collection, data analysis and validation, model training, model evaluation, and model serving, which can be a complicated process for machine learning developers. In response, end-to-end machine learning platforms [8, 39] that perform all of these steps have been proposed.

As machine learning applications become more diverse, obtaining enough training data is becoming one of the most critical bottlenecks [30]. Unlike well-known problems like machine translation where there is decade's worth of parallel corpora to train models, most new applications have little or no training data to start with. For example, a smart factory application for quality control may need labeled images of its own specific products. In response, there has been significant progress in data collection research [30] including crowdsourcing methods [1, 2], weak supervision methods [28], and simulator-based data collection [22]. As a result, it is reasonable to assume that labeled data can now be collected at will given enough budget.

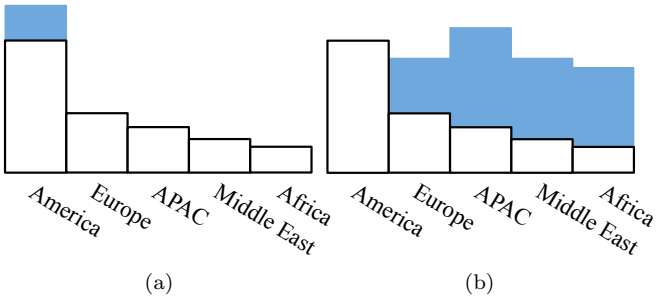


Figure 1: A set of slices for customers in different regions. The naïve approach of blindly collecting any data is not optimal. (a) In this example, there may be enough American customer data, so collecting more data in this region may worsen the bias. (b) Instead, we would like to collect possibly-different amounts of data for slices to optimize both the model accuracy and fairness on all slices.

When collecting data for machine learning, a common misconception is that more data leads to better models. However, this claim does not always hold when the goal is to improve both the model accuracy and fairness, which are not always aligned. For example, suppose a company sells apps online using a model trained on customer purchase information in different regions and needs to ensure that its recommendations are not only accurate overall, but for customers in different regions (see Figure 1). The latter notion of fairness called equalized error rates [35, 38]. If the company collects training data of customers in multiple regions, but already has enough data for say American customers, then collecting more American customer data is not only unnecessary, but may bias the training data and influence the model's behavior on other regions. Worse model accuracy for certain regions can be viewed as unfair discrimination. In general, training data can be divided into any subsets, which we refer to as *slices*.

We thus propose a *selective data collection* framework called Slice Tuner, which determines how much data to col-

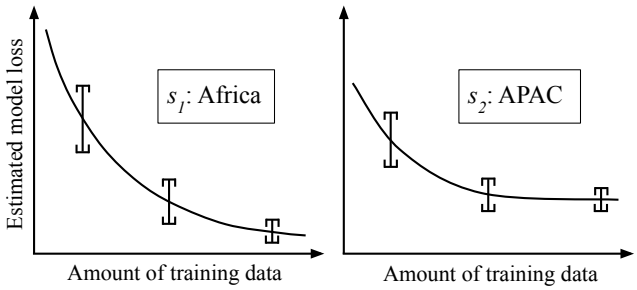


Figure 2: Hypothetical learning curves of two slices.

lect for each slice. Not all slices have the same data collection cost-benefits, which means that collecting the same amounts of data may result in different improvements in model loss for different slices. Hence, the naïve strategy of collecting equal amounts of data per slice is not necessarily optimal. Alternatively, collecting data such that all slices end up having equal amounts of training data is not always optimal either. (This approach is similar to the Water filling algorithm [26]; details in Section 2.2.) Instead, we would like a data collection strategy such that the models are accurate and similarly-accurate for different slices. To measure model accuracy, we use loss functions like logistic loss. For model fairness, there are many definitions, but we extend equal error rates [35, 38] to measure unfairness where we take the average difference between an "underperforming" slice versus the rest of the data (see the exact definition in Section 2.1). For both measures, lower values are better. The key challenge is to figure out the different cost benefits of the slices and estimate how much data to collect for each slice given a budget. In Figure 1b, we may want to collect more non-American customer data. Among the other regions, APAC customers may require more data to obtain a certain model loss than customers in other slices.

At the core of Slice Tuner is the ability to estimate the *learning curves* of slices, which reflect the cost benefits of data collection. It is well known that the impact of data collection on model loss is initially large, but eventually plateaus [32]. That is, lowering the loss becomes difficult to the point where it is not worth the data collection effort. Figure 2 shows hypothetical learning curves of two slices. Recently, there have been multiple studies [19, 14] showing that these curves usually follow a power law according to empirical results in machine translation, language modeling, image classification, and speech recognition. Given the learning curves, Slice Tuner "tunes" the slices by determining how much data to collect for each slice such that the model accuracy and fairness on all slices are optimized while using a limited budget for data collection.

As a simple example, suppose there are slices s_1 and s_2 among others. Say the current model losses of the two slices are 4 and 3, respectively. Also, say that for s_1 or s_2, the rest of the slices have a loss of 1. Hence, the unfairness can be computed as $\text{avg}\{|4-1|, |3-1|\} = 2.5$. Now suppose we estimate their two learning curves as shown in Figure 2. Notice that the curve of s_2 is rather flat, so collecting more data does not have as much benefit as collecting for s_1. Suppose we have a budget of collecting examples using crowdsourcing. Since s_1 has a better cost-benefit, we may decide to only collect examples for s_1. The actual numbers would depend on the result of the optimization problem. As a result, say the model losses of s_1 and s_2 are now 2 and 3, respectively, and that for s_1 or s_2, the rest of the slices still have a loss of 1. In that case, the unfairness improves by decreasing to $\text{avg}\{|2-1|, |3-1|\} = 1.5$.

Two key challenges are that the learning curves may not be reliable initially due to insufficient data, and the slices may influence each other where data collected for one slice may result in the model performing better or worse on other slices. Slice Tuner solves these problems by iteratively updating the learning curves as more data is collected.

In addition, there is the question on how to define the slices themselves. Data slicing can either be done manually or automatically [13]. While Slice Tuner can run on any set of slices, a desirable property of a slice is to be unbiased such that collecting any example that belongs to it has a similar effect on model loss as any other possible example. Although not the main focus on this paper, we discuss possible solutions in Section 6.

In our experiments, we demonstrate on real datasets that Slice Tuner outperforms other baselines in terms of model loss and unfairness. In particular, for a face image dataset called UTKFace [40], we consider the real scenario of collecting new images using crowdsourcing via Amazon Mechanical Turk [1] and show that Slice Tuner is effective even if the data is collected from a completely different data source.

To democratize machine learning, it is important to help users to not only analyze their models [5, 13], but also fix any problems easily. To our knowledge, Slice Tuner is the first system to provide concrete action items to make models both accurate and fair. We also release our code and crowdsourced dataset as a community resource [3].

The rest of the paper is organized as follows:

- We cover preliminaries and formulate the problem of selective data collection (Section 2).
- We describe Slice Tuner's architecture (Section 3).
- We propose accurate and efficient methods for estimating the learning curves of slices (Section 4).
- We propose the following selective data collection algorithms (Section 5):
  - *One-shot*: Assumes that slices are independent and suggests how much data to collect in one step.
  - *Iterative*: Repeatedly updates the learning curves as more data is collected and invokes *One-shot*.
- We discuss methods for finding slices for data collection (Section 6).
- We evaluate Slice Tuner on real datasets and show that it outperforms baselines by obtaining better model accuracy and fairness results using the same data collection budget, even if the slices are initially small (Section 7).

2. PROBLEM DEFINITION

2.1 Preliminaries

Data Slicing. We denote by D the training data set with examples where each example $e \in D$ has features and a label. D can be divided into slices $S = \{s_i\}_{i=1}^n$ where each $s_i \subseteq D$. We assume that the slices partition D, i.e., $\bigcup_i s_i = D$. A typical way to define a slice is to use conjunctions

of feature-value pairs. For customers, slices can be defined based on features including region and gender, e.g., $region = Europe \wedge gender = Female$. As another example, for self-driving car applications, the slices can be defined based on weather conditions for driving, i.e., $weather = Sunny$. We can also use the label feature for the slicing. For example, the MNIST dataset [24] contains images that represent the digits 0 to 9. Here, we can define 10 slices for the 10 digits. The slices can be found automatically or set manually by a domain expert. We also define the *complement* of a slice s to be the other examples in the entire dataset, i.e., $D - s$. Although there is no restriction in defining slices, a desirable property of a slice is to be unbiased such that adding a new example to it has a similar effect on the model accuracy as any other example. We briefly discuss how to find such slices in Section 6.

Model Accuracy and Fairness Measures. We assume a model M is trained on D or its subsets. We also assume a classification loss function $\psi(s, M)$ that returns a performance score on how well M predicts the labels of the dataset s. A common loss function for binary classification is *log loss*, which is defined using cross entropy:

$$LogLoss = \frac{1}{m}\sum_{i=1}^{m} -y^{(i)} \log \hat{y}^{(i)} - (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}).$$

A perfect classifier M has a log loss of zero, and a random classifier $-ln(0.5) = 0.693$. Our setting can be generalized to other machine learning problems (multi-class classification and regression) by using the appropriate loss functions. Throughout the paper, we will use loss to measure accuracy.

We propose a model fairness measure based on equalized error rates [35], which has the following definition:

$$Pr(\hat{y} \neq y | z = 0) \approx Pr(\hat{y} \neq y | z = 1)$$

where $\hat{y}$ is the model prediction, and z is a sensitive attribute like gender or age. While there are other fairness measures [35], we believe equalized error rates is practical and can be used by any machine learning product that needs to provide similar qualities of service to customers of any region or slice.

While most existing fairness research typically assumes two sensitive groups based on a sensitive attribute (e.g., $gender = male$, $gender = female$), we extend the notion to any number of slices. Suppose we have a model M. We say a slice s is *underperforming* if it has a higher loss than its complement, i.e., $\psi(s, M) > \psi(D - s, M)$. Let us denote the set of underperforming slices as $S_u \subseteq S$.

DEFINITION 1. *The* unfairness *of the slices is defined as the average difference between the loss of an underperforming slice and its complement:*

$$\text{avg}_{s \in S_u} \psi(s, M) - \psi(D - s, M).$$

Notice that the unfairness measure does not require sensitive attributes and can thus be used on any dataset. We can also think of other variations such as computing the maximum difference instead of the average.

Data Collection Cost. We assume that any data collection technique can be used to collect data by paying a certain cost. The most expensive, but accurate method is crowdsourcing performed by crowd workers [1, 2]. More recently, weak supervision [28] is on the rise where data programming systems [29] like Snorkel [27, 7] and Snuba [34] are used to collect weak labels that are not as accurate as manual labels, but can be collected at scale to still train accurate models. Yet another approach is to use simulators to generate as much data as one needs. For example, low level functions of an open-world video game have been manipulated to simulate accident and non-accident scenes of self-driving cars [22]. The caveat here is that there must be a simulator that can truly represent events in the world.

We abstract the data collection methods and define a cost function $C(s)$ that returns the cost to collect an example in slice s. Even for the same data source, the data collection cost may vary by slice. For example, collecting face images of large groups (e.g., $race = White$) may be easier than images of smaller groups (e.g., $race = Native\ Hawaiian$). We assume that within the same slice, the cost to collect an example is the same. As more examples are collected for s, $C(s)$ may increase possibly because data becomes scarcer. However, we assume that, during each iteration of selective data collection (see Section 5.2), $C(s)$ is a constant.

Slice Dependencies. Slices may have *dependencies* where collecting data for one slice may influence the learning curves of other slices. For example, if there are two independent slices s_1 and s_2, and we collect too much data for s_1, the model may overfit and have a worse accuracy on s_2. If s_2 is similar to s_1 content-wise, then the accuracy on s_2 can actually increase as well. If there are no dependencies, the slices are considered *independent* of each other. We discuss how to handle slice dependencies in Section 5.2.

2.2 Selective Data Collection

We now define the problem of selective data collection.

DEFINITION 2. *Given a set of examples D, its slices $S = \{s_i\}_{i=1}^n$, and the set of underperforming slices $S_u \subseteq S$, a trained model M, a data collection cost function C, and a data collection budget B, the selective data collection problem is to collect d_i examples for each slice $s_i \in S$ such that the following are all satisfied:*

- *The average loss $\frac{1}{|D|}\sum_{s_i \in S} \psi(s_i, M)$ is minimized,*
- *The unfairness $\text{avg}_{s_i \in S_u} \psi(s_i, M) - \psi(D - s_i, M)$ is minimized, and*
- *The total data collection cost $\sum_i C(s_i) \times d_i = B$.*

We note that minimizing loss and unfairness are correlated, but not necessarily the same and thus need to be balanced (Section 7.3.2 studies the tradeoffs). In some cases, making sure slices have similar losses may also result in the lowest loss. For example, if there are two independent slices with identical learning curves, and one of them has less data, then simply making the two slices have the same amount of data results in the optimal solution. However, there are cases where the two objectives are not aligned. Continuing our example, suppose that the two slices now have different learning curves where the slice with less data has a curve that is lower than the other curve and also decreases more rapidly. In this case, collecting data for the smaller slice would lower loss, but increase unfairness. Instead, the optimal solution could be to also collect some data for the larger slice to lower the loss without sacrificing too much fairness.

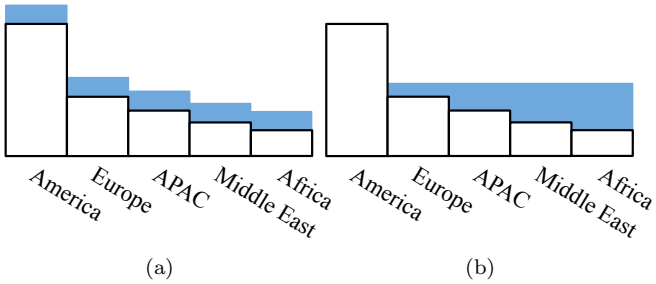


Figure 3: The two baselines for data collection: (a) collect similar amounts of data per slice, (b) collect data such that the slices have similar amounts of data in the end (Water filling algorithm).

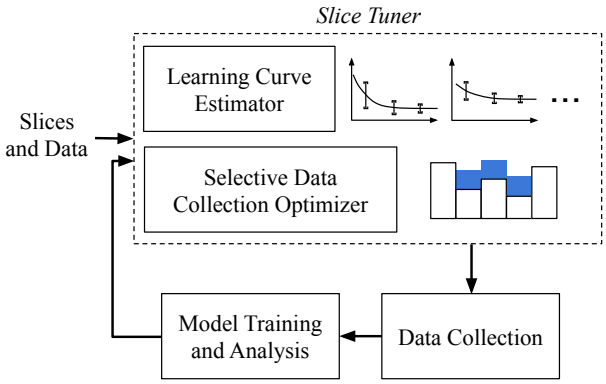


Figure 4: The input of Slice Tuner are the slices with their data. Slice Tuner consists of two main components: the Learning Curve Estimator and Selective Data Collection Optimizer. The learning curves may be iteratively updated as more data is collected by the Data Collection component. Slice Tuner suggests how much data to collect per slice.

Two Baselines. We now explain why the two baselines mentioned in Section 1 do not optimally solve our problem in Definition 2. Suppose there are two independent slices s_1 and s_2. The first baseline is to collect equal amounts of data for all slices (Figure 3a). This approach does not perform well if the two slices have significantly different learning curves. In a worst-case scenario, s_1 may already have a low loss and does not need more data collection whereas s_2 may have a high loss and can benefit from more data. In this case, collecting equal amounts of data will result in both suboptimal loss and unfairness. The second baseline is to collect data such that all slices have similar amounts of data in the end, which can be viewed as a Water filling algorithm (Figure 3b). The implicit assumption is that all slices require similar amounts of data to obtain similar losses. However, this assumption does not hold if the two slices have different losses even if they have the same size. Continuing from the above worst-case scenario, if s_1 is smaller than s_2, then we will end up collecting data for s_1 unnecessarily and again get suboptimal results. What we need instead is a way to utilize the learning curves and solve an optimization problem to determine how much data to collect for each slice.

3. SYSTEM OVERVIEW

We describe the overall workflow of Slice Tuner as shown in Figure 4. Slice Tuner receives as input a set of slices and their data and estimates the learning curves of the slices by training models on samples of data. We explain the details of the estimation methods in Section 4. Next, Slice Tuner performs the selective data collection optimization where it determines how much data should be collected per slice in order to minimize the loss and unfairness. As data is collected, the learning curves can be iteratively updated. We propose selective data collection algorithms in Section 5.

We discuss the runtime requirements of Slice Tuner. Looking at Figure 4, we consider the Data Collection step to be the most expensive process done as a batch process, especially if manual crowdsourcing is used. Even if weak supervision or simulation are used, they may still involve time-consuming manual programming. As a result, it is critical for Slice Tuner to minimize the amount of data collection at the expense of possibly using more computation for estimating learning curves and performing the optimization.

4. LEARNING CURVE

A learning curve is a projection of how a model trained on the entire dataset will perform on a particular slice s as a function of the number of examples in s. Assuming that the examples are helpful to the model training, we expect the loss to decrease as more examples are added. However, this trend may not always hold due to multiple factors: the examples may be noisy and actually harm the model training, the examples may be biased and only represent a small part of the slice, or the model training itself may be unstable, all of which may result in non-monotonic behavior. Despite the complexity, we believe it is reasonable to assume that more training data on an unbiased slice generally leads to lower loss, but that the benefits have diminishing returns. Unlike existing work, a significant challenge for Slice Tuner is to plot the learning curves on slices, which can be arbitrarily smaller than the entire data. We first discuss how to efficiently estimate learning curves in this section and then how to address the small data issue by iteratively updating the learning curves in Section 5.

4.1 Estimation

A key property we exploit is that training data benefits the model accuracy, but more collection has diminishing returns. A recent work from Baidu [19] conducted an analysis on learning curves (see Figure 5). The curve starts with the *small-data region*, where models try to learn from a small number of training data and can only make "best guess" predictions. According to our experience, tens of examples is enough to move beyond this region. For more data, we can see a *power-law region* of the form $y = bx^{-a}$, where new training examples provide useful information to improve the predictions. For real-world applications, a lower bound error may exist due to errors such as mislabeled data that cause incomplete generalization. Hence, we then see a *diminishing-returns region* where there is a minimum loss that cannot be reduced. Another study [14] compares 11 parametric models including variations of exponential models and custom models. Based on these works, a power-law curve fits as well as any other curve.

To fit a learning curve of a slice, we first divide its data into train and validation sets. Although the train set may be small, we do assume a validation set that is large enough

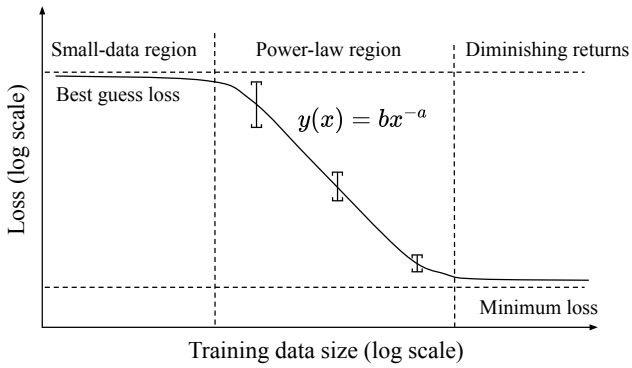


Figure 5: A detailed view of how loss changes against more training data based on work by Baidu [19]. There are three regions: (1) *small-data region*: the slice is too small to the extent that the model predictions are more or less random, (2) *power-law region*: the curve is of the form $y = bx^{-a}$, and (3) *diminishing-returns region*: there is enough data, so collecting more does not help reduce loss.

to evaluate models. This assumption is reasonable in the common setting where we start with some initial data and would like to collect more examples. We then train models on random subsets of the train set with different sizes and generate data points by evaluating the models on the validation set. How many data points to generate depends on how much time we are willing to invest to estimate the learning curve. Each time we train a model on a subset of data, we also combine that data with the rest of the slices. (Section 4.2 presents a more efficient method for generating data points.) We then fit a power-law curve on the points using a non-linear least squares method [14]. Here the error is measured as the squared sum of the loss differences between the curve and the actual losses for the random subsets.

When taking random subsets of the data, some of them may be close to or within the small-data region, which means the model losses are not reliable and have a high variance when repeatedly training models as depicted as the error bars in Figure 5. We thus give more weight to larger subsets when fitting a learning curve. We can further improve the learning curve accuracy by drawing multiple curves and averaging them at the expense of more computation, although we did not need this in our experiments. In the worst case when all subsets are in the small-data region, then Slice Tuner may not benefit from learning curves, but in that case would fall back to performing like baselines.

If there is enough data in a slice to be in the diminishing return region, we do not need to do any special handling because Slice Tuner will simply collect data for other slices that need more data.

4.2 Efficient Implementation

A straightforward approach of fitting learning curves can be an expensive process. Given the slices S, suppose we generate for each slice K random subsets of data to fit a power-law curve. In addition, we may repeatedly update the learning curves U times using our iterative algorithms in Section 5.2. We would thus have to train a model $|S| \times K \times U$ times. A typical setting in our experiments is $|S| = 10$, $K = 20$, and $U = 5$, which means that we may train a model 1K times. Another problem is that, if we train a model on a subset of a slice plus the rest of the slices, then the rest of the slices could be relatively too large for the model to properly train on the subset due to the data bias. Moreover, each model training make take a long time because the rest of the slices are repeatedly used for training.

We thus propose several amortization techniques to drastically reduce the number of model trainings. First, instead of generating K data points for each of the U iterations, we reuse previous data points and incrementally add a few data points per iteration. Second, instead of taking an X% subset of one slice and leaving the rest as is to train each model, we take X% subsets of all slices and train a model. This model is then evaluated on each of the slices to generate different learning curves independently. This approach relies on the independence assumption where taking subsets of other slices does not affect the model accuracy on the current slice. Even if the independence assumption does not hold, updating the learning curves frequently enough will have the effect of enforcing independence. In Section 5.2, we explain how Slice Tuner decides to update the learning curves. Combining the two techniques, the number of model trainings reduces to $O(K + U)$. In our example above, number 1K reduces to about 25. We verify these results in Section 7.4.

5. SELECTIVE DATA COLLECTION

We first tackle the selective data collection problem when slices are independent of each other and then extend our methods to the case where slices may influence each other.

5.1 Independent Slices

We first assume that slices do not influence each other. Hence, we only need to solve the optimization problem once. The optimization should be done on all slices as our objective for minimizing loss and unfairness is global. We can formulate a convex optimization problem for selective data collection as follows. For the slices $\{s_i\}_{i=1}^n$ with sizes $\{|s_i|\}_{i=1}^n$, we want to find the number of examples to collect for the slices $\{d_i\}_{i=1}^n$ to minimize the objective function using a total budget of B. We assume that the model's loss on a slice follows a power-law curve of the form $b_i(|s_i| + d_i)^{-a_i}$ where b_i and a_i are positive values. We define A_i to be the loss of s_i's complement $D - s_i$ and $C(s_i)$ to be the cost function that captures the effort to collect an example for s_i. The optimization problem is then

$$\min \sum_{i=1}^{n} b_i(|s_i|+d_i)^{-a_i} + \lambda \sum_{i=1}^{n} \max\left\{0, \frac{b_i(|s_i|+d_i)^{-a_i}}{A_i} - 1\right\}$$
$$\text{subject to} \sum_{i=1}^{n} C(s_i) \times d_i = B$$

where the first term minimizes the loss, and the second term minimizes unfairness by giving a penalty to each slice s_i that has a higher loss than A_i. If the loss of s_i is lower than the loss of A_i, then we return 0 to prevent the unfairness term from giving a negative value. Since minimizing loss and unfairness are not always aligned, we introduce the λ term to balance between minimizing each of them.

This problem is convex assuming that the model's loss decreases monotonically as more data is collected. The first loss term $b_i(|s_i| + d_i)^{-a_i}$ is a convex function of d_i because it is a power-law curve. The second unfairness term is also

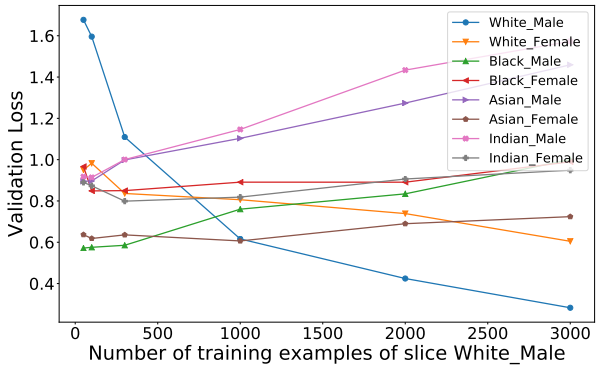


Figure 6: The losses of 8 UTKFace data slices where data is collected for the White_Male slice only. The losses of the other slices are influenced as more examples are collected.

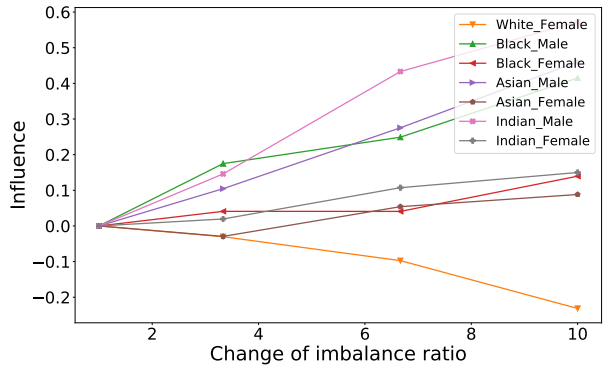


Figure 7: The influences (loss changes) of other UTKFace data slices as more data is collected for White_Male, starting from size 300. As the change of the imbalance ratio increases, so does the magnitude of influences on other slices.

convex because $b_i(|s_i|+d_i)^{-a_i}$ is convex, A_i does not change against d_i, and taking a maximum between two convex functions is convex. Finally, $C(s_i)$ is simply a constant that varies by slice. Hence, we can derive an optimal solution efficiently using any off-the-shelf convex optimization solver.

Algorithm. The *One-shot* algorithm updates the learning curves using the techniques in Section 4.2 and solves the above optimization problem to determine how much data to collect for each slice. Note that *One-shot* always uses the entire budget B, assuming the learning curves are perfect.

5.2 Dependent Slices

We now consider the case where slices can influence each other. Suppose that there are two slices s_1 and s_2. If we collect data for s_1 such that it dominates s_2 in quantity, then the model training may overfit on s_1. Consequently, the model accuracy on s_2 may change. Hence, we need to iteratively update the learning curves as more data is collected. Notice that the iterative updates also serve the dual purpose of making the learning curves more reliable.

Modeling the Influence. In general, it is difficult to model how exactly the data collection on one slice may influence other slices. However, we do suspect that the sizes of the slices and the data similarities between slices play a role. To understand these factors better, we perform an experiment on the UTKFace dataset [40] where we use slices that represent different race and gender combinations of people (more details in Section 7.1). Initially, all the slices are of the same size 300, except for the slice White_Male, which starts from size 50. As we add more examples only to White_Male, most slices have increasing losses as they are negatively affected by the bias while the slice White_Female consistently shows decreasing losses because it has similar data.

A key observation is that, the more the relative sizes of slices change, the more the losses vary as well. To capture the relative sizes, we use the notion of *imbalance ratio* [10], which is the maximum ratio between any two slices in S. For example, if the slices s_1, s_2, and s_3 have sizes 10, 20, and 30, respectively, then the imbalance ratio is $\frac{max\{10,20,30\}}{min\{10,20,30\}} = \frac{30}{10} = 3$. We also define *influence* on a slice as the change in loss. We hypothesize that, if the imbalance ratio changes, the magnitude of influence increases as well. Figure 7 uses the same information in Figure 6, but explicitly shows how a positive change in imbalance ratio results in more positive or negative influence. The results for a negative change in imbalance ratio are similar. Hence, we would like to control the imbalance ratio by limiting the amount of data collected.

Algorithm. The *Iterative* algorithm (shown in Algorithm 1) limits the change of imbalance ratio to determine how much data to collect for each slice. We obtain the slice sizes and initialize the imbalance ratio absolute change limit T to 1. While there is enough budget for data collection, we increase T per iteration using one of the strategies discussed later. The parameter L specifies the minimum slice sizes to start with and is positive. If any slice is smaller than L, we collect enough examples assuming there is enough budget. (In Section 7.3.4, we show that L can be a small value.) We then run the *One-shot* algorithm to derive how many examples to collect for each slice if we use the entire budget B. If the imbalance ratio change would exceed T if we use the entire budget, we limit the number of examples collected by multiplying *num_examples* with the maximum ratio that would not allow that to happen (*change_ratio*). This problem has nonlinear constraints, and the GetChangeRatio function uses an off-the-shelf optimization library in SciPy to derive a solution. After reflecting B, T, and IR, we repeat the same steps until we run out of budget.

For example, suppose $L = 10$, and there are two slices s_1 and s_2 with initial sizes of 5 and 10 (i.e., $sizes = [5, 10]$) with a budget of $B = 55$. First, we need to collect 5 examples for s_1 (i.e., $num_examples = [5,\ 0]$) to satisfy L (Step 4). Then we update $sizes$ to $[10, 10]$ and B to 50 (Steps 5–6). Then we set IR to $\frac{10}{10} = 1$. After running *One-shot*, suppose that $num_examples = [10,\ 40]$ (Step 9). If we collect all this data, the imbalance ratio would become $\frac{10+40}{10+10} = 2.5$, so $|After_IR - IR| = 2.5$ - $1 = 1.5$. In order to avoid exceeding $T = 1$, we compute the change ratio x such that $\frac{10+40x}{10+10x} = 2$ by invoking the GetChangeRatio function (Step 13). The solution is $x = 0.5$, and $num_examples$ thus becomes $0.5 \times [10, 40] = [5, 20]$. After collecting the data, we update $sizes$, B, T, and IR and go back to Step 8.

We now discuss strategies for updating the limit T per iteration using the IncreaseLimit function. On one hand, it is desirable to minimize the number of iterations of Algorithm 1 because each iteration invokes the *One-shot* al-

ALGORITHM 1: Iterative algorithm for Slice Tuner

Input: The slices S, budget B, minimum slice size L, and data collection cost function C

$sizes$ = SliceSizes(S);
$T = 1$;
if $\exists i\ sizes[i] < L$ **then**
  /* Ensure minimum slice size L */
  $num_examples = \max(L \times \mathbf{1} - sizes, 0)$;
  $sizes = sizes + num_examples$;
  $B = B$ - $\sum_i (C(i) \times num_examples[i])$;
IR = GetImbalanceRatio($sizes$);
while $B > 0$ **do**
  /* *One-shot* always uses the entire budget */
  $num_examples$ = OneShot($sizes$, B);
  $After_IR$ = GetImbalanceRatio($sizes + num_examples$);
  if $|After_IR - IR| > T$ **then**
    /* Do not make imbalance ratio change exceed T */
    $target_ratio = IR + T \times$ Sign($After_IR - IR$);
    $change_ratio$ = GetChangeRatio($sizes$, $num_examples$, $target_ratio$);
    $num_examples = change_ratio \times num_examples$;
  CollectData($num_examples$);
  $sizes = sizes + num_examples$;
  $B = B$ - $\sum_i (C(i) \times num_examples[i])$;
  T = IncreaseLimit(T);
  $IR = After_IR$;
return;
Function GetImbalanceRatio($sizes$):
  return $\frac{\max(sizes)}{\min(sizes)}$;

gorithm, which involves updating the learning curves and solving an optimization problem. On the other hand, we would like to update the learning curves to be as accurate as possible. While there are many ways to update T, we propose the following representative strategies:

- *Conservative*: a conservative strategy where for each iteration, we leave T as a constant, which limits the imbalance ratio to change linearly. The advantage of this approach is that we can avoid mistakenly collecting too much data due to inaccuracies in the learning curves. On the other hand, the number of iterations may be high.
- *Moderate*: a moderate strategy where for each iteration, we increase T by a constant c. Compared to *Conservative*, this approach reduces the number of iteration, but may collect data unnecessarily.
- *Aggressive*: an aggressive strategy where for each iteration, we multiply T by a constant $c > 1$. Compared to *Moderate*, this strategy collects data more aggressively using fewer iterations.

Notice that after many iterations, *Aggressive* has a similar behavior as the *One-shot* algorithm because T is large enough to not limit the amount of data that can be collected.

6. DATA SLICING

In this section, we briefly discuss various approaches for data slicing. While this topic is not the main focus of this paper, it is worth discussing the state-of-the-art methods. The straightforward way is to select slices manually based on domain knowledge. For example, for a movie recommendation system, one may select slices based on genre. Alternatively, one can determine slices based on model analysis. Manual tools for visualization [5, 20] can be used to find problematic slices where a model underperforms. Recently, there are automatic tools [13] that can find such slices.

As we mentioned in Section 2.1, a desirable property of a slice is to be unbiased so that the collected examples have similar effects on the model accuracy. Using large slices that are biased is undesirable. For example, a slice that contains all regions of Figure 1a is bad because there is a bias towards American customers. That is, adding an American customer example is not as helpful as say adding a European customer example. On the other hand, using slices that are not biased, but too small is also problematic because we may need to maintain many learning curves that are not accurate due to the small amounts of data.

In order to find the largest-possible slices that are still unbiased, one can use a method similar to decision tree training where the goal is to find partitions of the data such that the impurity (i.e., homogeneity of labels in leaf nodes) is minimized. Instead of minimizing impurity, we would have to compute the bias in slices using say an entropy-based measure. Starting from the entire dataset, we can iteratively split slices that have biases in their data for different values of attributes. The splitting can terminate once the average entropy is above some threshold.

7. EXPERIMENTS

In this section, we evaluate Slice Tuner on real datasets and address the following questions:

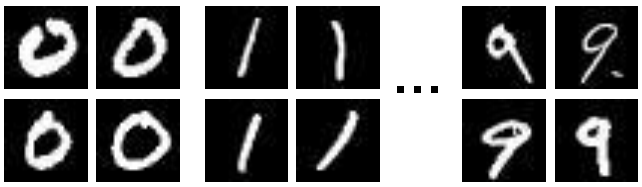
- How accurate and efficient is the learning curve generation used in Slice Tuner?
- How does Slice Tuner compare with the baselines in terms of model loss and unfairness?
- How does Slice Tuner perform on small slices where the learning curves are inaccurate?

Slice Tuner is implemented in TensorFlow [6], and we use Titan RTX GPUs for model training.

7.1 Setting

Datasets. We experiment on the following four datasets that capture different characteristics of the slices. Figure 8 shows samples of the image datasets. While the AdultCensus dataset is the most widely-used in the fairness literature, Slice Tuner is not limited to any particular dataset because our unfairness measure (Definition 1) does not require sensitive attributes (e.g., race and gender).

- MNIST [24]: Contains images that represent digits from 0 to 9 where the goal is to predict the digit of each image. Here we slice the images according to their labels, i.e., there are 10 slices in total. Compared to the other datasets, the slices are the most "homogeneous" in the sense that they are all about digits.
- Fashion-MNIST [36]: Contains images that can be categorized as one of 10 type of clothes, e.g., shoes, shirts, pants, and more. Compared to MNIST, the slices have more variety in terms of the objects they represent.
- UTKFace [40]: Contains various face images of people of different (male and female) gender and race (White, Black, Asian, and Indian), used for race classification. We used 8 slices by combining two genders and four races, e.g., White male, White female, and so on.



(a) MNIST dataset

(b) Fashion-MNIST dataset

(c) UTKFace dataset

Figure 8: Sample data of the three image datasets.

- AdultCensus [23]: Contains people records containing features including age, education, and sex. The prediction task is to determine if a person makes over $50K per year. We use 4 slices by combining two races (White and Black) and two genders (male and female).

Data Collection and Cost Function. For the MNIST, Fashion-MNIST, and AdultCensus datasets, we first simulate data collection by starting from a subset and adding more examples. This approach is reasonable because we are not tied to any data collection technique. We also define the cost function to always return 1. For the UTKFace dataset, we use a real scenario where we crowdsource new images using Amazon Mechanical Turk (AMT) [1] and store them in Amazon S3. We design a task by asking a worker to find new face images of a certain demographic (e.g., $slice = White_Women$) from any website. We pay 4 cents per image found, employ workers from all possible countries, and collected images during 8 separate time periods. We do not show the workers all the images collected so far, so they may collect duplicate images. (The duplicate rate is not as high as we think because workers around the world use a wide range of websites to collect images.) In addition, some workers make mistakes and collect incorrect images that do not fit in the specified demographic. Hence, we include a post-processing step of filtering obvious errors manually, removing exact duplicates, and cropping faces using Google Cloud AI Platform services. We also define the collection cost of a slice to be proportional to the average time a task is finished. Table 1 shows the average time (seconds) to collect images for the 8 UTKFace slices. Interestingly, the collection costs can be quite different. For example, an Indian woman image takes 50% longer to collect than a Black male image and thus has a cost of 1.5.

Methods Compared. We compare the following methods:

	W_M	*W_F*	*B_M*	*B_F*	*A_M*	*A_F*	*I_M*	*I_F*
Avg. time (s)	82.1	81.9	67.6	79.3	94.8	77.5	91.6	104.6
Cost C	1.2	1.2	1	1.2	1.4	1.1	1.4	1.5

Table 1: The collection costs of UTKFace slices are proportional to the average times to finish tasks. Here W = White, B = Black, I = Indian, M = Male, and F = Female.

- *Uniform (baseline 1)*: collects similar amounts of data per slice.
- *Water filling (baseline 2)*: collects data such that the slices end up having similar amounts of data.
- *One-shot*: updates the learning curves and solves the optimization problem once and collects data as described in Section 5.1. As a default, we set $\lambda = 1$.
- *Iterative*: iteratively updates the learning curves and collects data as described in Section 5.2. We use three iteration strategies for increasing T per iteration: *Conservative* (fixes T to 1), *Moderate* (increases T by 1), and *Aggressive* (multiplies T by 2).

Measures. We use the model loss and unfairness measures based on our discussions in Section 2.1. For all measures, lower values are better.

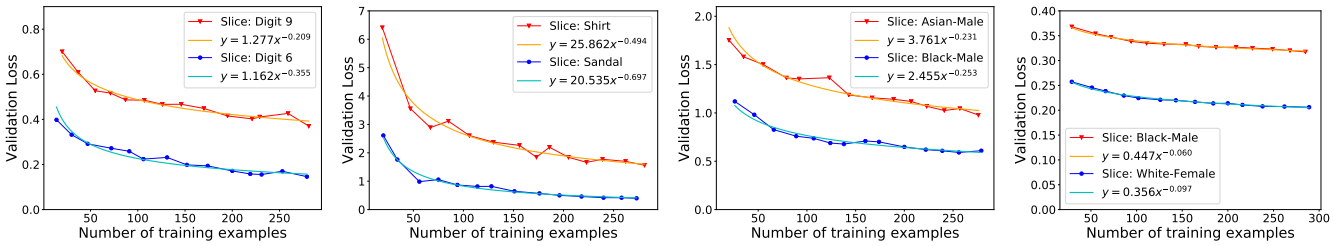
- *Loss*: the average log loss for multi-class classification.
- *Average Equalized Error Rates (Avg. EER)*: the unfairness measure in Definition 1, i.e., the avg. loss difference between an underperforming slice and its complement.
- *Maximum Equalized Error Rates (Max. EER)*: same as avg. EER, except that we take the maximum loss difference instead of the average. We use this measure to understand the worst-case unfairness.

For each slice, we split the available data into train and validation sets and measure the loss on the validation set. We set the validation set size to be 500.

Models and Hyperparameter Tuning. For the MNIST, Fashion-MNIST, and UTKFace datasets, we use basic convolutional neural networks with 1–3 hidden layers. For the AdultCensus dataset, we train a fully connected network with no hidden layers. For both datasets, we initially set the hyperparameters using simple grid search. Afterwards, we do not further change the hyperparameters while running Slice Tuner to ensure the model training is consistent.

7.2 Learning Curve Analysis

We analyze our learning curve estimation method described in Section 4. For each slice, we take $K = 10$ or $K = 20$ random samples of the data with differing sizes and use a non-linear least squares method to fit a curve. Figure 9 shows two learning curves for each dataset where the x-axis is the subset size, and the y-axis is the loss on a validation set. As the subset size increases, the loss decreases as well. Even for the most homogeneous dataset MNIST, the learning curves can be different, resulting in different amounts of data collection for the slices. We also observe how the learning curve changes as the slice itself grows in size. That is, we increase the slice size and, for each size, fit a new learning curve. Figure 10 shows the learning curves for a slice in the



(a) MNIST (b) Fashion-MNIST (c) UTKFace (d) AdultCensus

Figure 9: The learning curves of the MNIST, Fashion-MNIST, UTKFace, and AdultCensus datasets. For each dataset, we show two learning curves for different slices fitted using a validation set. Even for seemingly-homogeneous slices in the MNIST dataset, the learning curves can be quite different.

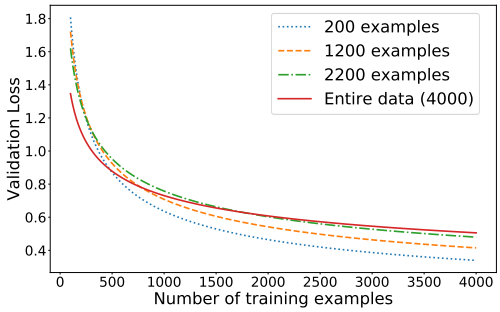


Figure 10: Given a slice in the Fashion-MNIST dataset, we increase its size and train new learning curves. As a result, the learning curves trained on small slices deviate more from the other curves and can be viewed as erroneous.

Fashion-MNIST dataset. As a result, the smaller the slice, the more the learning curve deviates from the others. This result shows that the learning curves must be updated as more data is collected, especially for slices that start small.

7.3 Selective Data Collection Optimization

We now show the loss and unfairness results of Slice Tuner and make a comparison with baselines.

7.3.1 *Loss and Unfairness of Slice Tuner Methods*

Table 2 compares the loss and unfairness results of the Slice Tuner methods on the four datasets. *Original* is where we train a model on the current slices with no data collection. As a result, the Slice Tuner methods improve *Original* both in terms of loss and unfairness. Among the Slice Tuner methods, the iterative methods outperform *One-shot*. For Fashion-MNIST and UTKFace, *One-shot* does have a lower max. EER, but that is because it uses too much budget on a particular slice, which happens to have the worst EER, so the max. EER is improved the most by chance. Between the *Aggressive* and *Conservative* methods, *Conservative* usually has lower loss and unfairness than *Aggressive*. This result is expected because *Conservative* uses more iterations to update the learning curves more accurately. For the case for *Moderate*, the results are identical to *Aggressive* because both methods use the same number of iteration in this setting.

Table 3 shows how much data is collected for each method in Table 2 for the four datasets. For each dataset, the initial slice sizes (same as L) are specified in the *Original* row. While *One-shot* has only one chance to decide how much data to collect, the *Aggressive* and *Conservative* methods have more chances to adjust their results. For example, on the Fashion-MNIST dataset, *One-shot* overshoots and collects too much data for slice #6 while the other methods properly adjust their learning curves through more iterations. Another observation is that *Conservative* uses more iterations than *Aggressive* because it is more conservative in increasing T. The exceptions are UTKFace and AdultCensus where both methods use up their budgets after two iterations. Hence, *Conservative* can be viewed as trading off efficiency for the lower loss and unfairness results in Table 2.

Dataset	Method	Loss	Avg./Max. EER
MNIST	Original	0.310	0.110 / 0.172
	One-shot	0.202	0.076 / 0.157
	Aggressive	0.201	0.076 / 0.149
	Moderate	0.201	0.076 / 0.149
	Conservative	**0.172**	**0.033** / **0.075**
Fashion-MNIST	Original	0.756	0.541 / 1.141
	One-shot	0.519	0.240 / **0.496**
	Aggressive	0.514	0.213 / 0.556
	Moderate	0.514	0.213 / 0.556
	Conservative	**0.510**	**0.209** / 0.587
UTKFace	Original	0.830	0.118 / 0.221
	One-shot	0.786	0.091 / **0.142**
	Aggressive	**0.784**	**0.072** / 0.181
	Moderate	**0.784**	**0.072** / 0.181
	Conservative	**0.784**	**0.072** / 0.181
AdultCensus	Original	0.285	0.143 / 0.257
	One-shot	0.264	0.141 / 0.235
	Aggressive	**0.259**	**0.136** / **0.228**
	Moderate	**0.259**	**0.136** / **0.228**
	Conservative	**0.259**	**0.136** / **0.228**

Table 2: Slice Tuner methods comparison on the 4 datasets.

We also perform experiments where the initial slice sizes follow an exponential distribution. The results are similar and are shown in Table 10 and Table 9.

In summary, our iterative algorithms clearly outperform *One-shot*. Also, while *Aggressive* has slightly-worse loss and unfairness results than *Conservative*, it uses few iterations. In the next sections, we will only use *Aggressive*.

Dataset	Method	0	1	2	3	4	5	6	7	8	9	# iterations
MNIST ($B = 8K$)	Original	300	300	300	300	300	300	300	300	300	300	n/a
	One-shot	112	281	2305	523	488	611	1	87	2394	1198	1
	Aggressive	50	102	1690	637	404	594	0	733	1802	1988	3
	Moderate	50	102	1690	637	404	594	0	733	1802	1988	3
	Conservative	89	422	1608	978	349	810	0	621	1646	1477	6
Fashion-MNIST ($B = 6K$)	Original	300	300	300	300	300	300	300	300	300	300	n/a
	One-shot	525	208	940	404	682	278	2370	223	263	107	1
	Aggressive	648	144	1326	263	1236	181	1936	98	125	43	3
	Moderate	648	144	1326	263	1236	181	1936	98	125	43	3
	Conservative	848	83	1293	318	1193	139	1891	86	119	30	6
UTKFace ($B = 3K$)	Original	350	350	350	350	350	350	350	350	-	-	n/a
	One-shot	689	146	118	228	320	169	516	167	-	-	1
	Aggressive	530	351	90	228	475	130	398	142	-	-	2
	Moderate	530	351	90	228	475	130	398	142	-	-	2
	Conservative	530	351	90	228	475	130	398	142	-	-	2
AdultCensus ($B = 200$)	Original	120	120	120	120	-	-	-	-	-	-	n/a
	One-shot	0	0	122	78	-	-	-	-	-	-	1
	Aggressive	0	0	136	64	-	-	-	-	-	-	2
	Moderate	0	0	136	64	-	-	-	-	-	-	2
	Conservative	0	0	136	64	-	-	-	-	-	-	2

Table 3: Selective data collection results for the experiments in Table 2 where slices are listed numbered from 0 up to 9 (the ordering is not important). The *Original* row contains the initial number of examples for each slice while the other rows show the additional number of examples collected.

Dataset	λ	Loss	Avg./Max. EER
MNIST	0	0.181	0.056 / 0.115
	0.1	0.186	0.034 / 0.078
	1	0.196	0.024 / 0.062
	10	0.201	0.019 / 0.062
Fashion-MNIST	0	0.483	0.321 / 0.690
	0.1	0.494	0.241 / 0.618
	1	0.506	0.228 / 0.603
	10	0.511	0.210 / 0.564
UTKFace	0	0.777	0.093 / 0.182
	0.1	0.785	0.107 / 0.159
	1	0.785	0.066 / 0.175
	10	0.790	0.056 / 0.170
AdultCensus	0	0.267	0.141 / 0.222
	0.1	0.266	0.142 / 0.224
	1	0.270	0.140 / 0.225
	10	0.273	0.137 / 0.222

Table 4: Results of *Aggressive* when varying λ.

7.3.2 Balancing λ

We study the effect of balancing λ. Recall that a higher λ means there is more emphasis on optimizing fairness. How to set λ depends on whether the loss or unfairness is more important to minimize for the given application. Table 4 shows the loss and unfairness results on the four datasets when varying λ using the *Aggressive* method and the same initial data and budget as in Table 3. As λ increases, the avg. and max. EER results decrease while the loss increases.

Table 5 shows the amounts of data collected per slice for different λ values using the Fashion-MNIST dataset. The results for the other datasets are similar and not shown here. In our example, slices #2, #4, and #6 start with higher losses than other slices, and the experiments with higher λ values results tend to be more aggressive in collecting data for those three slices in order to reduce the unfairness.

Method	0	1	2	3	4	5	6	7	8	9
Original	400	400	400	400	400	400	400	400	400	400
$\lambda = 0$	633	174	789	497	789	313	1229	221	253	102
$\lambda = 0.1$	628	134	860	441	866	268	1339	186	212	66
$\lambda = 1$	836	3	1110	138	1073	68	1730	13	29	0
$\lambda = 10$	753	0	1159	119	1135	0	1833	0	0	0

Table 5: Selective data collection results for the Fashion-MNIST experiments in Table 4.

7.3.3 Comparison with Baselines

We make a detailed comparison between *Aggressive* and the baselines *Uniform* and *Water filling* in Table 6 where we use three settings: (1) a basic setting where slices have the same amounts of data, (2) a pathological setting for *Uniform* (called "Bad for Uniform") where there are many slices with low loss, and (3) a pathological setting for *Water filling* (called "Bad for Water filling") where there is a large slice with high loss and a small slice with low loss. For each setting, we compare the three methods by varying the budget B to simulate many scenarios. For the AdultCensus dataset, however, setting B to 150 is already enough to obtain high accuracy, so we do not use larger budgets. As a result, for setting (1), *Aggressive* has both lower loss and avg. EER than the baselines for all datasets. *Aggressive* shows the best improvements on the Fashion-MNIST dataset because the slices have quite different learning curves, which

Dataset	Measure	Alg.	Basic			Bad for Uniform			Bad for Water filling		
			Initial	1000	3000	Initial	1000	3000	Initial	1000	3000
MNIST	*Loss (# Iters)*	*Uni*		0.346	0.283		0.289	0.229		0.279	0.220
		WF	0.387	0.346	0.283	0.350	0.267	0.237	0.290	0.263	0.241
		Agg		**0.337** (2)	**0.277** (2)		**0.236** (1)	**0.226** (1)		**0.242** (1)	**0.209** (1)
	Avg. EER	*Uni*		0.137	0.096		0.138	0.094		0.141	0.070
		WF	0.118	0.137	0.096	0.360	0.112	0.094	0.174	0.083	0.079
		Agg		**0.063**	**0.069**		**0.059**	**0.065**		**0.060**	**0.063**
Fashion-MNIST	*Loss (# Iters)*	*Uni*		0.663	0.581		0.704	0.591		0.616	0.541
		WF	0.757	0.663	0.581	0.854	0.645	0.582	0.659	0.617	0.560
		Agg		**0.618** (1)	**0.550** (2)		**0.637** (1)	**0.558** (2)		**0.588** (2)	**0.527** (1)
	Avg. EER	*Uni*		0.502	0.446		0.573	0.481		0.409	0.378
		WF	0.557	0.502	0.446	0.720	0.497	0.438	0.329	0.458	0.427
		Agg		**0.298**	**0.270**		**0.412**	**0.255**		**0.264**	**0.179**
UTKFace	*Loss (# Iters)*	*Uni*		0.880	0.850		0.951	0.892		0.876	0.838
		WF	0.918	0.880	0.850	0.998	0.927	0.854	0.896	0.882	0.836
		Agg		**0.878** (1)	**0.836** (1)		**0.917** (1)	**0.839** (1)		**0.858** (1)	**0.819** (1)
	Avg. EER	*Uni*		0.121	0.148		0.502	0.421		0.108	0.129
		WF	0.165	0.121	0.148	0.565	0.401	0.271	0.149	0.153	0.106
		Agg		**0.088**	**0.104**		**0.362**	**0.243**		**0.080**	**0.097**
			Initial	150		Initial	150		Initial	150	
AdultCensus	*Loss (# Iters)*	*Uni*		0.269			0.273			0.268	
		WF	0.285	0.269		0.288	0.270		0.277	0.273	
		Agg		**0.257** (1)			**0.261** (1)			**0.264** (1)	
	Avg. EER	*Uni*		0.138			0.147			0.135	
		WF	0.144	0.138		0.146	0.151		0.142	0.140	
		Agg		**0.136**			**0.143**			**0.127**	

Table 6: A detailed comparison of *Aggressive(Agg)* against the two baselines – *Uniform(Uni)* and *Water filling(WF)* – on the four datasets where we use three settings and vary the budget B. We also set $\lambda = 0.1$.

is advantageous for Slice Tuner. Another notable result is that *Aggressive* is the best method for the UTKFace dataset where the data is collected through AMT. The main reason the improvements are not as large as Fashion-MNIST is that the slices are more homogeneous and thus have similar learning curves. Looking at the two baselines, they have similar performances, so it is not clear which one performs better. In settings (2) and (3), we observe *Uniform* and *Water filling* performing much worse, respectively. On the other hand, *Aggressive* consistently performs the best.

In addition, for all experiments, *Aggressive* mostly performs one iteration and occasionally two iterations, as shown in the parentheses in Table 6. This result suggest that, in most scenarios, *Aggressive* is just as efficient as the *One-shot* method while obtaining low loss and unfairness.

7.3.4 Small Data Results

We lower the initial slice sizes to $L = 20$ where the learning curves are noisy and unreliable as shown in Figure 11. Nonetheless, Table 7 shows that Slice Tuner still performs better than the baselines because it can leverage the relative losses among the learning curves. Even if the learning curves are completely useless, Slice Tuner's performance should fall back to those of the baselines. In a sense, the learning curves are used as hints and do not have to be perfect. However, as more data is collected, the learning curves will gradually become more accurate and useful.

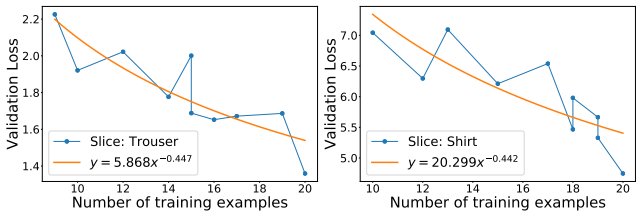


Figure 11: Learning curves are noisy for small slices.

7.4 Efficient Learning Curve Generation

We evaluate the efficient learning curve generation method described in Section 4.2 as shown in Table 8. The default *Aggressive* method uses the efficient learning curve generation. We make a comparison with a modified version of *Aggressive* where the learning curve generation is done exhaustively (called "Exhaustive"). We also vary the initial slice size and budget. As a result, *Aggressive* is 32–55x faster than Exhaustive as expected and obtains similar or even lower loss and unfairness results. We consider these runtimes to be practical because the main bottleneck of Slice Tuner is the time to actually collect data, which may take say hours. While it may sound counter intuitive that *Aggressive* can have lower loss and unfairness, this may happen because our optimization of taking $X\%$ examples of all slices together has the effect of removing bias among the slices.

Init. Size	Method	Loss	Avg. / Max. EER
20 ($B = 500$)	Original	2.696	1.896 / 3.186
	Uniform	1.369	0.897 / 1.780
	Water filling	1.369	0.897 / 1.780
	Aggressive	**1.296**	**0.628** / **1.217**

Table 7: Loss and unfairness results of Slice Tuner methods for a small slice sizes on the Fashion-MNIST dataset.

Method	Loss	Avg. / Max. EER	Runtime (sec)
Init. size = 200, B = 2K			
Exhaustive	0.613	0.275 / 0.479	49,211
Aggressive	0.610	0.239 / 0.624	891
Init. size = 300, B = 3K			
Exhaustive	0.546	0.245 / 0.424	40,387
Aggressive	0.540	0.247 / 0.467	1,244

Table 8: A comparison between a method using exhaustive generation of learning curves versus the *Aggressive* method on the Fashion-MNIST dataset for different initial slice sizes and budget values.

8. RELATED WORK

While there is a significant literature on analyzing models [5, 13], there is relatively little work on actually fixing their problems. To our knowledge, Slice Tuner is the first system to provide selective data collection as concrete action items to make models both accurate and fair.

Data Collection. There are many works on data collection [30, 33] including crowdsourcing methods [1, 2], weak supervision [28], and simulator-based data collection [22]. Crowdsourcing gives the highest quality labels, but tends to be the most expensive approach. Weak supervision generates labels with lower quality, but automatically at scale. Simulator-based labeling gives the most flexibility, but assumes that there is a simulator that models the world. Slice Tuner can use any of these techniques for collecting data. In addition, Slice Tuner solves the new problem of how much data to collect for each slice.

Active learning [31] is a heavily-studied topic where the goal is to select the most useful unlabeled examples for crowd workers to label in order to maximize the model accuracy with the least manual effort. In comparison, Slice Tuner optimizes both model accuracy and fairness using slices. In addition, Slice Tuner involves collecting new examples while active learning is focused on labeling existing examples.

Model Fairness. Model fairness [35] is becoming a critical issue for any machine learning application. Due to the subjective nature of the notion of fairness, many measures have been proposed to capture discrimination where the most prominent ones are disparate impact [15], equalized odds [18], equal opportunity [18], disparate mistreatment [38], and equalized error rates [35]. Slice Tuner uses an extended version of equalized error rates, which we believe is the most practical measure in settings where machine learning products needs to provide even service quality to users. More recently, there has been a surge in unfairness mitigation techniques [9, 37], which improve the model fairness by either fixing the training data (pre-processing), model training (in-processing), or the trained model (post-processing). Most of these approaches assume a fixed dataset and attempt to modify the data say by adjusting weights of examples before model training. In comparison, Slice Tuner takes the completely different approach of selectively collecting new data for different slices to improve model fairness.

Learning Curves. Estimating learning curves has mainly been studied in the computer vision community [19, 12, 16, 17]. Research by Baidu [19] analyzes the three different areas of how the learning curve changes depending on the amount of data. Another paper [14] compares multiple types of curves that best fit the real trend. The conclusion is that using a power-law curve is reasonable. All of these works assume that there is a single learning curve that shows the model accuracy on the entire data. In comparison, Slice Tuner solves the more challenging problem of estimating learning curves of multiple slices. Slices are not only smaller than the entire dataset, but may also influence each other, so Slice Tuner needs to iteratively update the learning curves.

Data Slicing. A line of work solves the problem of data slicing where the goal is to find problematic slices where the model underperforms. There are largely two approaches for data slicing: manual and automatic. Visualization tools like TensorFlow Model Analysis [5] and MLCube [21] can be used to observe the model accuracies on different slices. More recently, Slice Finder [13] has been proposed to automatically find slices that have significantly low model accuracies, but are large enough to be of interest. Slice Tuner can use any of these approaches and only recommends that the slices be unbiased. In addition, Slice Tuner solves the new problem of selective data collection.

More recently, slice-based learning [11] has been proposed to focus the model training on certain slices that the model underperforms. While this approach assumes the training data is static, Slice Tuner takes the complementary approach of actually collecting more data for slices and also optimizing both accuracy and fairness together.

9. CONCLUSION

We proposed Slice Tuner, which performs selective data collection to optimize model accuracy and fairness. Slice Tuner estimates learning curves for different slices and solving an optimization problem to determine how much data collection is needed for each slice given a budget. To address the challenges of unreliable learning curves and dependencies among slices, we proposed iterative algorithms that repeatedly update the learning curves as more data is collected, using the imbalance ratio change to estimate influence. We demonstrated on real datasets that our iterative algorithms are practically efficient and obtain lower loss and unfairness than the two baseline methods that do not exploit the learning curves, even if the slice sizes are initially small. In addition, we demonstrated the practicality of Slice Tuner by collecting new data using Amazon Mechanical Turk.

10. ACKNOWLEDGMENTS

This work was supported by a Google AI Focused Research Award. We also thank Hyungseung Hwang for his help with implementation.

11. REFERENCES

[1] Amazon mechanical turk. `https://www.mturk.com/`. Accessed March 1st, 2020.

[2] Amazon sagemaker. `https://aws.amazon.com/sagemaker/`. Accessed March 1st, 2020.

[3] Slice tuner github repository. `https://github.com/khtae8250/SliceTuner`. Accessed March 1st, 2020.

[4] Software 2.0. `https://medium.com/@karpathy/software-2-0-a64152b37c35`. Accessed March 1st, 2020.

[5] Tensorflow model analysis. `https://github.com/tensorflow/model-analysis`. Accessed March 1st, 2020.

[6] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *OSDI*, pages 265–283, 2016.

[7] S. H. Bach, D. Rodriguez, Y. Liu, C. Luo, H. Shao, C. Xia, S. Sen, A. Ratner, B. Hancock, H. Alborzi, R. Kuchhal, C. Ré, and R. Malkin. Snorkel drybell: A case study in deploying weak supervision at industrial scale. In *SIGMOD*, pages 362–375, 2019.

[8] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, et al. Tfx: A tensorflow-based production-scale machine learning platform. In *KDD*, pages 1387–1395, 2017.

[9] R. K. E. Bellamy, K. Dey, M. Hind, S. C. Hoffman, S. Houde, K. Kannan, P. Lohia, J. Martino, S. Mehta, A. Mojsilovic, S. Nagar, K. N. Ramamurthy, J. T. Richards, D. Saha, P. Sattigeri, M. Singh, K. R. Varshney, and Y. Zhang. AI fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias. *CoRR*, abs/1810.01943, 2018.

[10] M. Buda, A. Maki, and M. A. Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249 – 259, 2018.

[11] V. S. Chen, S. Wu, A. J. Ratner, J. Weng, and C. Ré. Slice-based learning: A programming model for residual learning in critical data slices. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *NeurIPS*, pages 9392–9402, 2019.

[12] J. Cho, K. Lee, E. Shin, G. Choy, and S. Do. Medical image deep learning with hospital PACS dataset. *CoRR*, abs/1511.06348, 2015.

[13] Y. Chung, T. Kraska, N. Polyzotis, K. H. Tae, and S. E. Whang. Slice finder: Automated data slicing for model validation. *IEEE TKDE*, 2019.

[14] T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, pages 3460–3468, 2015.

[15] M. Feldman, S. A. Friedler, J. Moeller, C. Scheidegger, and S. Venkatasubramanian. Certifying and removing disparate impact. In *KDD*, pages 259–268, 2015.

[16] R. L. Figueroa, Q. Zeng-Treitler, S. Kandula, and L. H. Ngo. Predicting sample size required for classification performance. *BMC Med. Inf. & Decision Making*, 12:8, 2012.

[17] K. Hajian-Tilaki. Sample size estimation in diagnostic test studies of biomedical informatics. *Journal of Biomedical Informatics*, 48:193–204, 2014.

[18] M. Hardt, E. Price, and N. Srebro. Equality of opportunity in supervised learning. In *NeurIPS*, pages 3315–3323, 2016.

[19] J. Hestness, S. Narang, N. Ardalani, G. F. Diamos, H. Jun, H. Kianinejad, M. M. A. Patwary, Y. Yang, and Y. Zhou. Deep learning scaling is predictable, empirically. *CoRR*, abs/1712.00409, 2017.

[20] M. Kahng, D. Fang, and D. H. P. Chau. Visual exploration of machine learning results using data cube analysis. In *HILDA*, page 1. ACM, 2016.

[21] M. Kahng, D. Fang, and D. H. P. Chau. Visual exploration of machine learning results using data cube analysis. In *HILDA@SIGMOD*, page 1, 2016.

[22] H. Kim, K. Lee, G. Hwang, and C. Suh. Crash to not crash: Learn to identify dangerous vehicles using a simulator. In *AAAI*, pages 978–985, 2019.

[23] R. Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *KDD*, pages 202–207, 1996.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press, 2001.

[25] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich. Data lifecycle challenges in production machine learning: A survey. *SIGMOD Record*, 47(2):17–28, 2018.

[26] Proakis. *Digital Communications 5th Edition*. McGraw Hill, 2007.

[27] A. J. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *PVLDB*, 11(3):269–282, Nov. 2017.

[28] A. J. Ratner, B. Hancock, and C. Ré. The role of massively multi-task and weak supervision in software 2.0. In *CIDR*, 2019.

[29] A. J. Ratner, C. D. Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. In *NIPS*, pages 3567–3575, 2016.

[30] Y. Roh, G. Heo, and S. E. Whang. A survey on data collection for machine learning: a big data - AI integration perspective. *CoRR*, abs/1811.03402, 2018.

[31] B. Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[32] V. S. Sheng, F. J. Provost, and P. G. Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In *KDD*, pages 614–622, 2008.

[33] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *ICCV*, pages 843–852, 2017.

[34] P. Varma and C. Ré. Snuba: Automating weak supervision to label training data. *PVLDB*, 12(3):223–236, 2018.

[35] S. Venkatasubramanian. Algorithmic fairness: Measures, methods and representations. In *PODS*, page 481, 2019.

[36] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[37] D. Xu, S. Yuan, L. Zhang, and X. Wu. Fairgan: Fairness-aware generative adversarial networks. In *ICBD*, pages 570–575, 2018.

[38] M. B. Zafar, I. Valera, M. Gomez-Rodriguez, and K. P. Gummadi. Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment. In *WWW*, pages 1171–1180, 2017.

[39] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie, and C. Zumar. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018.

[40] S. Y. Zhang, Zhifei and H. Qi. Age progression/regression by conditional adversarial autoencoder. In *CVPR*. IEEE, 2017.

APPENDIX

A. ZIPFIAN DISTRIBUTION RESULT

In Section X, we showed results for uniform distributions. In this section, we show results when the initial slice sizes form an exponential distribution. In Table 9, the key trends are similar that *Aggressive* and *Conservative* outperform *One-shot* because *One-shot* tends to collect too much data for certain slice as shown in Table 10. Moreover, while *Conservative* uses more iterations than *Aggressive*, it has slightly-better loss and unfairness results.

Dataset	Method	Loss	Avg./Max. EER
MNIST	Original	0.319	0.130 / 0.271
	One-shot	0.213	0.145 / 0.201
	Aggressive	0.181	0.031 / 0.071
	Conservative	**0.180**	**0.021** / **0.039**
Fashion-MNIST	Original	0.767	0.589 / 1.314
	One-shot	0.512	0.313 / **0.545**
	Aggressive	0.506	0.237 / 0.585
	Conservative	**0.498**	**0.217** / 0.562
UTKFace	Original	1.292	0.626 / 1.231
	One-shot	0.975	0.184 / **0.294**
	Aggressive	0.966	0.162 / 0.393
	Conservative	**0.956**	**0.160** / 0.400
AdultCensus	Original	0.284	0.133 / 0.215
	One-shot	0.270	**0.133** / **0.215**
	Aggressive	**0.264**	**0.133** / 0.217
	Conservative	**0.264**	**0.133** / 0.217

Table 9: Slice Tuner methods comparison on the 4 datasets where the initial slice sizes follow an exponential distribution.

Dataset	Method	0	1	2	3	4	5	6	7	8	9	# iterations
MNIST ($B = 8K$)	Original	232	240	250	261	275	292	314	346	396	500	n/a
	One-shot	436	179	3248	207	664	903	178	531	0	1654	1
	Aggressive	200	74	1602	1737	396	617	61	558	965	1760	3
	Conservative	170	51	1666	1604	641	780	47	951	790	1300	5
Fashion-MNIST ($B = 6K$)	Original	189	200	212	226	244	268	300	346	424	600	n/a
	One-shot	525	208	940	404	682	278	2370	223	263	107	1
	Aggressive	635	132	1274	450	1275	218	1873	89	54	0	3
	Conservative	769	114	1250	409	1266	196	1859	82	54	0	4
UTKFace ($B = 4K$)	Original	62	71	83	100	125	166	250	500	-	-	n/a
	One-shot	849	930	485	511	276	325	0	0	-	-	1
	Aggressive	913	971	458	489	242	301	0	0	-	-	3
	Conservative	933	976	463	495	214	297	0	0	-	-	4
AdultCensus ($B = 200$)	Original	75	86	106	150	-	-	-	-	-	-	n/a
	One-shot	0	0	0	200	-	-	-	-	-	-	1
	Aggressive	0	0	66	134	-	-	-	-	-	-	2
	Conservative	0	0	66	134	-	-	-	-	-	-	2

Table 10: Selective data collection results for the experiments in Table 9 where slices are listed numbered from 0 up to 9 (the ordering is not important). The *Original* row contains the initial number of examples for each slice while the other rows show the additional number of examples collected.