System to Integrate Fairness Transparently: An Industry Approach

Emily Dodwell, Cheryl Flynn, Balachander Krishnamurthy, Subhabrata Majumdar, Ritwik Mitra
AT&T Labs - Research, USA
{emily,cflynn,bala,subho,ritwik}@research.att.com

ABSTRACT

There have been significant research efforts to address the issue of unintentional bias in Machine Learning (ML). Many well-known companies have dealt with the fallout *after* the deployment of their products due to this issue. In an industrial context, enterprises have large-scale ML solutions for a broad class of use cases deployed for different swaths of customers. Trading off the cost of detecting and mitigating bias across this landscape over the lifetime of each use case against the risk of impact to the brand image is a key consideration. We propose a framework for industrial uses that addresses their methodological and mechanization needs. Our approach benefits from prior experience handling security and privacy concerns as well as past internal ML projects. Through significant reuse of bias handling ability at every stage in the ML development lifecycle to guide users we can lower overall costs of reducing bias.

KEYWORDS

algorithmic bias, fairness, fairness in machine learning

1 INTRODUCTION

With the increasingly widespread use of Machine Learning (ML) in our daily lives, concomitant concerns of unintentional bias have received significant attention in the popular press and research literature. Most of the academic work defining fairness has been analytical, theoretical, and legal [5]. Technical ways to detect and remedy bias [27] and toolkits [6] have also been proposed.

Numerous ML bias-related problems have surfaced in applications like image identification[1], hiring[2], and targeted advertising [14]. However, the problem of bias in an industrial setting with use cases far more diverse than the handful of academic literature exemplars remains poorly covered. Industry has to serve under-represented communities and not prioritize the targeting or delivering of services in a discriminating way. It has a responsibility to continually monitor ML processes for bias and mitigate any identified to ensure business product integrity, preserve customer loyalty, and protect brand image. There are several barriers to a systemic solution [22], including the need for fairness-aware data gathering, identification of blind spots, use-case specific guidance, and human oversight.

We propose SIFT (System to Integrate Fairness Transparently) as an operational framework to identify and mitigate bias at different stages of an industry ML project workflow. SIFT enables an industrial ML team to define, document, and maintain their project's bias history. SIFT guides a team via mechanized and human components to monitor fairness issues in all parts of a project's workflow. Longitudinally, SIFT lowers the cost for dealing with fairness through reuse of techniques and past lessons.

SIFT draws valuable lessons from attempts to improve security and privacy on the Internet. In the security arena, many attempts were made to codify intrusion detection to prevent attacks. For example, the widely-used open-source intrusion detection and prevention system Snort[3] uses both signatures and rules while conducting real-time analysis of Internet traffic to match against attacks. Signatures that are stored, updated, and shared include methods to detect an attack, while rules detect specific vulnerabilities. ClamAV [4] is an anti-virus toolkit that uses complex routines to detect attacks via user-contributed signatures and allows automatic remote database updates to constantly enhance its functionality. SIFT mirrors notions of signatures and rules in its collection of an ML project's *bias history*. Over the course of an ML project this is updated with information on methods to detect and mitigate concerns like sparse groups, marginalized groups, and proxy variables. SIFT uses a collection of ML projects in the enterprise to record information about any bias detected in their bias histories, th enabling future use.

Similar to Privacy by Design [8], which pushed for key privacy notions to be embedded in social networks, we want bias detection and mitigation to be considered early on by ML project managers. Just as we find new vectors of attacks on security and instances of privacy leakages, there will be new vectors of potential ML bias arising from data and model reuse, or repurposing of ML approaches for alternate use cases. Given the costs of gathering data and the time pressure associated with deploying new projects in large enterprises, this may occur with higher frequency than expected.

SIFT introduces several human decision-making steps in the ML workflow to enforce risk assessment. Use-case specific factors to guide these steps include domain knowledge, bias history of past projects in the enterprise, legal guidelines, and cost considerations of brand impact or regulatory penalties. Similar to privacy, regulation may be proposed in the space of ML. Defensive steps to handle regulatory concerns and transparency mechanisms to demonstrate fairness are thus proactively built into the ML development lifecycle. We may not identify *all* potential bias and mitigate them, but by including human-level checks we *reduce* the likelihood of fairness concerns impacting an enterprise and amortize its cost by using the bias history of an increasing number of projects.

ML Projects in large enterprises that are non-customer facing, or those with no demographic or geographic proxies may have *no* potential bias concerns. Managers of such projects would want to quickly move on with their work. If several risk factors are not involved in any stage of the ML lifecycle, SIFT allows the project

[1] https://www.theverge.com/2018/1/12/16882408/google-racist-gorillas-photo-recognition-algorithm-ai
[2] https://www.telegraph.co.uk/technology/2018/10/10/amazon-scraps-sexist-ai-recruiting-tool-showed-bias-against
[3] https://www.snort.org/faq/what-is-snort
[4] https://www.clamav.net/about

to move ahead with more confidence that potential bias is not a concern. However, if fairness concerns are too high without a clear path to mitigation, we recommend an early exit from the SIFT process for the manager to reconsider the project design, for example, by collecting additional data.

Section 2 provides background on related work, enumerates industry-specific challenges, and summarizes our contributions. Section 3 presents the four pipelines of SIFT (Information gathering, Pre-model, Model-involved, and Outcome-involved) and their key components. The pipelines outline classes and steps that are reflected in pseudo-code to show that our ideas are deployable. Motivated by key industry areas with known high bias potential (e.g., advertising, personalization etc.), we design SIFT to help a nascent ML project address three specific questions: is fairness a concern; if so, at *what* lifecycle stage(s) should it be addressed, and finally *how* bias should be mitigated. Section 4 focuses on two specific industry segments with real-life use cases and Section 5 discusses present limitations of SIFT and our ongoing work.

2 MOTIVATION AND CONTRIBUTION

We begin with an overview of prior art before enumerating SIFT's contributions. Section 2.1 reviews fairness-related ML research highlighting challenges faced by industry ML practitioners. Section 2.2 summarizes the salient features of SIFT and how SIFT differs from prior art in addressing existing challenges.

2.1 Related work and industry challenges

There are two broad categories of research in bias and fairness.

Methodology. Depending on the use case and modeling objectives, several sources of bias and discrimination may exist in the data. A recent enumeration [27] lists 23 types of bias and 6 types of discrimination associated with ML models, exposing three types of fairness concerns: individual, group, and subgroup fairness. Research on bias detection and mitigation methodology include methods aimed at the pre-, in- or post-processing parts of an ML project using classification or regression modeling (see Section 3.3). Fair versions of other ML or statistical techniques such as clustering [3], community detection [28], and causal models [37] have also been proposed.

Tools. AI Fairness 360 (AIF360) [6] is a well-known tool that packages bias detection and mitigation methods in the literature for reuse. Among other similar packages, which are mostly open-source, FairML [1], Themis [18], FairTest [33], Aequitas[5], and Fairness Measures[6] offer bias detection, while Themis-ml [4] offers both detection and mitigation through an expandable platform. Tools for data and model documentation, like FactSheets [2], Datasheets [19], or Model Cards [29], can also be adapted to detect bias concerns in the data or different stages of the ML workflow.

Notwithstanding these tools and methods, industry ML practitioners continue to face several challenges for integrating fairness considerations into ML project workflows [22, 34]. A survey of industry ML practitioners [22] that spans use case domains identifies a few common themes among them.

[5]https://github.com/dssg/aequitas
[6]http://fairness-measures.org

Data Collection: Most fairness-aware methods focus on detection and mitigation algorithms applied to fixed datasets [22, 27]. Little guidance is available in the data gathering stage, e.g. identifying and documenting sensitive features, locating similar internal projects with their sensitive features and bias histories, and bias risk assessment given a project's data sources and sensitive features.
Blind spots: Lack of guidance on finding project areas with bias concerns, and tools to learn from past projects limit the ability of teams to detect bias in their new project. This hampers getting more data to mitigate sparse group issues by missing out on relevant sensitive sub-populations for the specific use case.
Use-case diversity: Fairness research has received unequal attention across ML domains and stages of the ML workflow [27]. The above challenges and lack of structured tools lower guidance for teams on issues like domain-specific metrics and methods, or finding appropriate proxies when individual-level demographic data is missing.
Need for human oversight: When and how to rely on human decision making is a key concern. Examples include choosing a mitigation strategy, the amount of additional data collected for it, risk and cost assessment for mitigation, consequences and trade-offs of optimizing for a fairness metric, and changes to project design to minimize the need for any fairness auditing.

2.2 Our contributions

Toolkits like AIF360 are mostly post-facto; they do not enjoin data scientists and program managers to consider bias at every stage of the ML workflow. SIFT brings key non-technical players (compliance, legal, PR) into the equation and reaps the benefit of domain expertise, along with reusing practical lessons from past ML projects. SIFT starts by defining a bias history object and then points out specific steps in the project for bias considerations, allowing practitioners to use relevant existing methods to detect and mitigate bias. Guided by use case and team requirements, these can be implemented either directly from the literature or from toolkits like AIF360. The bias history object sequentially records all such instances of bias-related checks, methods, and decisions. These are among the crucial benefits of SIFT from an industry perspective.

Each of the four pipelines in SIFT uses mechanized and human components in tandem. Human oversight and blind spot checks are thus an integral part of the process. It further allows the modeling process to complement non-technical but important considerations like business, contractual and legal constraints, and potential customer impact. This flexibility can be valuable for the development of fairness-aware ML systems in industry areas that are less explored in the fairness literature until now, e.g. recommendation systems, spatial models, and speech recognition.

Finally, cost considerations are central to ML project planning in the industry. Each new project using SIFT looks for similar projects internally and maintains its own model history and bias history. As more projects use SIFT, finding similar projects (along with their model/bias history) becomes more likely. Bias detection and mitigation becomes increasingly proactive, thereby reducing cost of fairness-aware project planning over time.

3 SIFT

We now describe the four pipelines of SIFT and their key components. We assume the existence of a database of existing ML projects in the company with the schema discussed below. We refer to the team working on the ML project as the SIFT user.

The SIFT framework has four classes: project, data, bias history and model history. The **`sift_project`** class is the top-level class for the system with all project-related information. We assume that a SIFT user starts a new project with (1) a `name`, (2) a `description` summarizing the background and objective, and (3) `data_location` of the database that stores data for the project, such as a remote directory, network drive, or URL. Based on these three input arguments, the user initiates a `sift_project` as described in Section 3.1. We assume that these inputs can be used at the outset to extract any available information regarding sensitive features, project personnel, and older versions of the same project. The full list of components of a `sift_project` are:

- `name`, `description`, and `data_location`,
- `project_id` – a unique identifier for the ML project,
- `sift_data`, `bias_history`, and `model_history` are objects of different classes described below and in Appendix A,
- `metadata` – dictionary with project personnel/status information
- `model_flow` – 'Standard' or 'Custom' bias-aware model building strategy (see Section 3.3),
- `similar_projects` – pointers to similar projects in projects database,
- `older_versions` – pointers to previous versions of the project in projects database,
- `timeout` – set to a company-specified time frame for terminated projects to be removed from the project database; defaults to None otherwise.

The **`sift_data`** class stores information about the data available for use in the project. This includes the raw data used in the project, data definitions, a list of feature variables and the target variable, and the predicted outcomes from any existing pre-built model. This class also contains the list of sensitive features relevant to the project, and a summary variable that is a *dictionary of dictionaries* with any additional information relevant to bias investigation, such as sparse groups, proxy features, or marginalized groups. See Appendix A.1 for a full list of `sift_data` class components.

We do not assume that every project has all of the components in the `sift_data` class available at the outset. For example, a new project may not initially know the relevant sensitive features. SIFT's identification of similar projects helps with this part of the project discovery, thereby reducing the overall cost of addressing bias.

The **`bias_history`** class is a novel contribution of the SIFT framework and is used to track each stage in the bias and mitigation process. The components of the `bias_history` class are:

- `step` – a counter capturing the place in the sequence of bias and mitigation tasks performed,
- `sift_pipeline`,
- `bias_features` – the sensitive features under consideration in the current step,
- `bias_detection_function`,
- `bias_mitigation_function`,
- `mitigation_success_status` – indicates whether bias is not detected after implementing the mitigation algorithm,
- `details` – additional information such as the results of the bias investigation or the actions taken by the SIFT user.

Steps in the bias history are added to document each stage of the bias detection and mitigation process. Methods associated with the class allow SIFT to access the current step of the bias history, add components to the current step, and add a next step in the history. We describe these methods in Appendix A.2.

We record longitudinal bias-related information across a large number of projects through `bias_history`, adding to the transparency on exactly *how* fairness is weaved into the ML lifecycle in an enterprise. This has numerous advantages. First, this enables information reuse for future projects and lowers cost across the enterprise in handling bias. Second, the precise nature of bias detected, the specific pipeline where it was first detected, the algorithm that helped locate it and its mitigation success status suggest the first steps for a new project. The success status is key in deciding if a recommended mitigation algorithm should be reused. Note that this value is not dispositive; a different mitigation algorithm might work better for a different use case or project. However it is still useful information that is traditionally not tracked in the ML lifecyle. The `bias_history` is returned as a result for queries for similar projects, indicating that the manner of resolution is visible to anyone in the future. Such transparency helps organizations defend the actions they have taken to address bias.

The **`model_history`** class tracks the history of the ML model through development and training. This class includes information on the training and test sets, the fitted model object, the performance metric(s) used to evaluate the model, and its deployment status at each stage of the modeling process. The class components are described in Appendix A.3.

SIFT assists the user through four pipelines (Figure 1). Progress through the pipelines is *not* sequential: After initiating a new project in Information gathering (P1), users move to Outcome-involved (P4) if the model is already deployed, otherwise to Pre-model (P2) followed by Model-involved (P4) pipelines. We present below more details on each pipeline with pseudo-code (written with respect to a single sensitive feature, so note that functions are run for each when there are multiple). Function names with an `M_` prefix indicate mechanizability while `H_` implies need for a human in the loop; auxiliary functions are described in Appendix B.

3.1 Information gathering pipeline

This pipeline checks in three steps if a new project has fairness concerns based on its description and intended audience.

(1) Project initialization. The pipeline starts by initializing the `sift_project` object which adds the prior model history (if any) obtained from the `data_location`, and any details on the data and metadata. It initializes the `bias_history` object, which is updated alongside bias-related steps taken throughout the project.

(2) Similar project identification. We now search the ML projects database for similar projects, its location specifed by `db_location`. The database can have a simple Web query interface whence `db_location` would be a URL. The search uses normal information retrieval steps: remove non-alphanumeric characters,

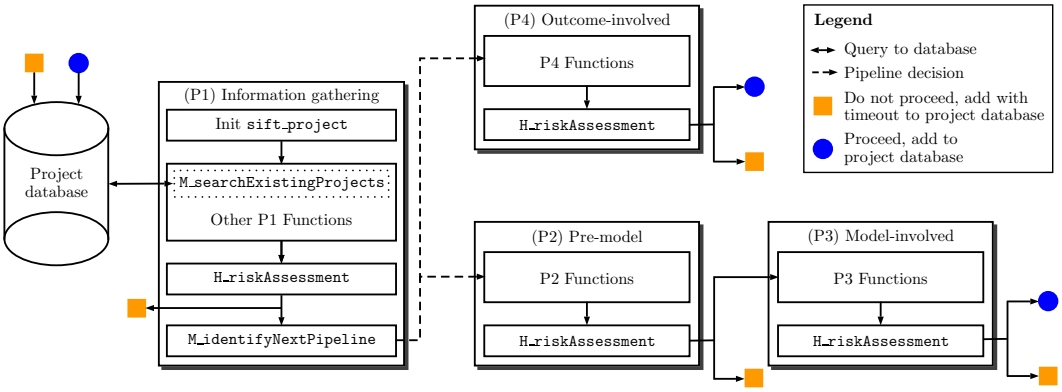

Figure 1: The four pipelines of SIFT.

normalize case, lemmatize words, remove stop words before vectorizing the data and calculating a cosine similarity score.

```
def M_searchExistingProjects(current_project, db_location,
  ngram, threshold):
  matched_projects_list = []; results_indices = []
  removeNonAlphaNumericChars (current_project)
  lowerCase (current_project)
  removeStopWords (current_project)
  lemmatize (current_project)
  ComputeTFIDFVectors_with_Bigrams (db_location, current_project)
  computeCosineSimilarity (db_location, current_project)
  if cosine_similarity > threshold: results_indices.append()
  matched_projects_list = database[results_indices]
  return matched_projects_list
```

The SIFT user human-verifies the list of matched projects via `H_verifySimilarity` and adds relevant ones to `similar_projects`. The user finds a relevant set of sensitive features by considering those in `similar_projects` and other factors (e.g., legal constraints and domain knowledge). `H_identifySensitiveCategories` captures this action and stores them as `sens_features`. In absence of similar projects, external considerations can identify sensitive features. `H_verifySimilarity` and `H_identifySensitiveCategories` are the *Other P1 Functions* in Figure 1.

(3) Preliminary risk assessment. The function `H_riskAssessment` now considers the information collected in the current `sift_project` object, with the list of similar projects, factoring in business, contractual, and legal constraints, as well as customer impact (see Section 2.2). If the decision is not to proceed due to absence of fairness concerns or high risk of bias, the project object is correspondingly updated. The `timeout` component is set based on company guidelines and the project is added to the project database before exiting SIFT. Else, the next pipeline is determined by the function `M_identifyNextPipeline`, which sets it to Outcome-involved if the model is deployed or to Pre-model otherwise. A deployed model is one that is beyond model training and development (e.g., the model is field-trial ready or already in production). Based on the `sift_pipeline` determination, the project is moved to either the Pre-model pipeline (Section 3.2) or the Outcome-involved pipeline (Section 3.4).

3.2 Pre-model pipeline

The Pre-model pipeline checks the raw data for issues that could lead to a biased ML model. A typical example of this pipeline would involve the following steps. Steps 1-4 among them are the *P2 Functions* of Figure 1.

At the end of each step of this pipeline, the bias history is updated with the task performed and the results, and a new step is added to the bias history. Detected sparse groups, proxy features, and marginalized groups are added to the `sens_features_summary`.

To simplify the flow through the pipelines, we implement all algorithmic mitigation strategies in the Model-involved pipeline, but allow the user to make changes to the raw data in this pipeline. Below we present pseudo-code for each step of the pipeline.

(1) Data preparation. Standard practice in ML workflows, this step involves data cleaning and feature engineering and is typically completed prior to performing any bias checks. While some aspects of this could be mechanizable, data preparation often involves some input and inspection of the data by the ML practitioner, handled by `H_prepareData`. The final data for the project should be updated and stored in the `sift_data` object.

(2) Sparse group detection. To train a fair ML model we need a sufficient number of training samples for each of the subgroups defined by the sensitive attributes. Otherwise, the ML model can have poor performance when predicting results for samples of the under-represented subgroup in practice. For example, Amazon abandoned an ML system intended to automate the hiring process by identifying resumes of top technical talent; its training on past resumes penalized female applicants due to historical gender imbalance within the tech industry. We check for sparse group representation in the data using the function defined below.

```
def M_detectSparseGroups(sens_feature):
  sparsityFunc, threshold = M_selectSparsityFunction()
  subgroup_list = []; sparse_groups_dict = {}
  subgroups = sens_feature.unique()
  for g in subgroup.iteritems():
    sparsity = sparsityFunc(sens_feature, g)
    if sparsity > threshold: subgroup_list.append(g)
  if subgroup_list: sparse_groups_dict[sens_feature] = subgroup_list
  return sparse_group_dict, sparsityFunc
```

If sparse groups are detected, the user can collect additional data or terminate the project with `H_verifyGetMoreData`. Else, the Model-involved pipeline will attempt to address this issue using an algorithmic pre-processing strategy such as reweighing or resampling [23].

(3) Proxy feature detection. Removing sensitive attributes from the set of features will not guarantee an unbiased ML model. One way bias may remain in the data is through the existence of proxy variables. For example, in the context of targeted advertising on Facebook, [32] found that many features provided on the Facebook ad platform were strongly correlated with sensitive attributes like gender and race. SIFT checks for strong pairwise correlations between sensitive and non-sensitive attributes.

```
def M_detectProxyFeatures(sens_feature, nonsens_feature):
  depFunc, threshold =
    M_selectDependenceFunction(sens_feature, nonsens_feature)
  feature_list = []; proxy_dict = {}
  for x in nonsens_features.itercols():
    dep = depFunc(sens_feature, nonsens_features[x])
    if dep > threshold: feature_list.append(x)
  if proxy_list: proxy_dict[proxy_features] = feature_list
  return proxy_dict, depFunc
```

Pairwise correlation checks do not guarantee the removal of all proxy variables: further bias checks are needed in the Model-involved pipeline. In particular, when the number of sensitive attributes and non-sensitive attributes is large, combinations of non-sensitive attributes could create a proxy for a sensitive attribute even when individual variables don't. Such multivariate proxies would not be detected at this step. When univariate proxy variables are detected, the SIFT user can drop the proxy variable from consideration in later modeling steps via `H_verifyDropProxy`.

(4) Marginalized group detection. Bias present in the target variable will be learned by the ML model. This step checks the target variable for marginalized groups to alert the SIFT user to this potential issue using the function `M_detectBias` (see Section 3.3). If marginalized groups are detected, then an algorithmic mitigation strategy can be implemented later in the Model-involved pipeline.

(5) Pre-model risk assessment. The last step of this pipeline asks the SIFT user to perform a risk assessment given the information learned in this pipeline–information that could fundamentally change the project plan. For example, if a key predictor variable is found to be a proxy for a sensitive attribute, the user may not wish to proceed with the project. The `bias_history` object captures the steps and results of each bias-related action taken in the pipeline. The user reviews this in `H_riskAssessment`. If the user decides not to proceed, then the project status is set to 'Terminated', `timeout` is set pursuant to company guidelines, and the project is added to the project database. Else, SIFT begins the Model-involved pipeline.

3.3 Model-involved pipeline

The Model-involved pipeline checks for bias introduced when training the ML model and implements mitigation strategies if bias is detected. This pipeline attempts to mitigate bias detected in Section 3.2 that could not be addressed by dropping variables or collecting additional data. It further tries to mitigate bias detected in the model outcome, which could arise even if the raw data passes previous checks due to complex data patterns. For example, a deep learning model could learn a non-linear function of non-sensitive features that creates a proxy variable for a sensitive feature.

A typical example of this pipeline can be broken down into six steps that we describe shortly. We refer to functions in steps 1-5 as *P3 Functions* in Figure 1. Unlike the previous pipelines, the user may not proceed sequentially through all the six steps. Constraints of time or computational resources may influence the choice and ordering of mitigation strategies. For example, if the ML model is computationally expensive to retrain and time-to-market is a concern, then the user may limit focus to only post-processing strategies. There is also no guarantee that any one mitigation strategy will resolve detected bias issues; multiple mitigation strategies may be required. Further, there may not exist a mitigation strategy that will address the source of bias, requiring designing of a novel mitigation strategy.

We thus allow for a choice of flow processes in this pipeline: standard or custom. The enterprise would determine the appropriate *standard* flow that a majority of projects would follow. For example, it could be set to closely follow [12], which works sequentially through the steps listed above. The *custom* flow allows the user control over the sequencing and implementation of the steps of the pipeline. The user can restrict attention to a specific set of bias detection metrics and mitigation algorithms, copy a routine from a similar project, or run a novel bias detection and mitigation strategy designed for the application. The ability to copy bias detection metrics and mitigation strategies used in similar ML projects is a key feature of SIFT that helps reduce the cost of reducing bias as more projects are added to the database. The `model_flow` input to the `sift_project` object indicates the selected flow for the project.

Below we present pseudo-code for each step of the pipeline. We assume that default settings for the functions would be standardized across the enterprise but could depend on the project application. These functions would make use of open-source bias mitigation algorithms or designed for company-specific projects. For reference, we provide examples of pre-, in-, and post-processing mitigation algorithms, that have open-source code available through [6]. An example standard flow process that works sequentially through the six steps is provided in Appendix C.

(1) Pre-processing mitigation. Pre-processing algorithms [7, 17, 23, 35] transform the raw data to reduce if not remove bias. These algorithms address bias in the raw data, for example, due to an under-representation of samples from a protected group, and do not require access to the training model or the model output.

As an example of a standard flow process, the system first summarizes the information collected during Pre-model pipeline using `M_getPreModelBiasStatus`. In case bias is detected, the system runs a pre-processing mitigation strategy using `M_preProcessingMitigation`. This function selects and implements the pre-processing strategy using the information collected in the Pre-model pipeline. It then returns the transformed dataset, and the pre-processing function, which is recorded in the `bias_history` object. The pre-processing function involves a transformation of the raw data, so the user will need to train the ML model regardless of prior model availability.

(2) Model training. This step of the pipeline uses the function `M_trainModel` to train the ML model.

```
def M_trainModel(current_project):
  seed, train_index, test_index =
    M_dataSplit(current_project.data.rawdata, split_ratio)
  modelFunc, modelMetricFunc = M_selectModel(current_project)
```

```
fitted_model = modelFunc(current_project.data, train_index)
perf_metric =  modelMetricFunc(current_project.data,
  fitted_model_object, test_index)
return (seed, train_index, test_index, fitted_model, perf_metric)
```

All iterations in this model development process will be documented in the `model_history` object.

(3) Model outcome bias detection. The existence of bias in the model outcome should be quantified using one or more bias detection metrics [6]. Often such metrics check that the model outputs are equivalent across different values of a sensitive feature or demographic subpopulations. The function `M_detectBias` computes these metrics with the model output as the `feature`.

```
def M_detectBias(current_project, feature, sens_feature):
  biasFunc, fairness_range = M_selectBiasFunction(current_project)
  biased_groups_list = []
  subgroups = sens_feature.unique()
  for g in subgroups.iteritems():
    bias = biasFunc(feature, sens_feature, g)
    if bias not in fairness_range: biased_groups_list.append(g)
  return biased_groups_list, biasFunc
```

In the above, `fairness_range` is the interval for which no bias is detected. Examples of bias detection metrics include disparate impact, equalized odds, demographic parity, and statistical parity [6, 15, 21].

(4) In-processing mitigation. In-processing algorithms incorporate one or more bias metrics directly into the prediction model and ensure that prediction accuracy is attained only under predefined fairness constraints stipulated by those bias metrics [10, 25, 36]. In SIFT, the function `M_inProcessingMitigation` selects and implements the in-processing strategy. It then returns information about the newly fitted model, which is stored as a new step in the `model_history` object, and the in-processing function, which is recorded in `bias_history`.

(5) Post-processing mitigation. Post-processing algorithms [21, 24, 30] transform the outputs from a specific trained model and are model-agnostic in the sense that they do not require access to the training data or the trained model. These methods are particularly useful when there is a high cost for re-training the underlying model. The function `M_postProcessingMitigation` selects and implements the post-processing mitigation strategy and subsequently returns the new predicted outcome and the selected post-processing mitigation function, which is recorded in the `bias_history` object.

(6) Model-involved risk assessment. The last step of this pipeline asks the SIFT user to perform a risk assessment given the results of this pipeline using the function `H_riskAssessment`. Since no mitigation strategy is guaranteed to remove all forms of bias, this is a key step in the process and will depend on factors such as the extent to which the ML model might impact customers and the potential risk to the enterprise's brand image. This assessment should also take into account any degradation in utility due to bias mitigation steps. If the user decides to proceed, then SIFT will mark the project as scheduled for deployment and record the project in the project database. Otherwise, SIFT marks the project as terminated and records the project in the project database with the appropriate timeout specified.

3.4 Outcome-involved pipeline

If the project focuses on a deployed model, then the user comes directly here from Information gathering. Examples of deployed models include ones in production that require periodic checks for bias, models that have passed training and development and are to be tried in the field, or third-party models used by the company.

This pipeline checks for bias in the model outcome and may mitigate detected bias using the post-processing algorithms described in Section 3.3. In this pipeline, we do not require access to either the training data or the trained model. Unlike the workflow in the Pre-model and Model-involved pipelines, the data fed through the model may be independent of the model's original training and test datasets. If an earlier version of the project exists, we assume that the previous iteration has been documented as a `sift_project` object, the project passed the appropriate bias risk assessments, and that the project has at least one sensitive feature identified in its data input. When an earlier version exists, this pipeline also checks for a distributional shift in the data between the two time points.

As a typical example, this pipeline would involve: (1) Data change detection, (2) Model outcome bias detection, (3) Post-processing mitigation, and (4) Outcome-involved risk assessment. Steps 1-3 are the *P4 Functions* in Figure 1. Steps 2 and 3 are same as ones in Section 3.3; pseudo-code for steps 1 and 4 are provided below.

Data change detection. Relevant variables may change over time, triggering a check against recorded statistics about the model's original training data for similarity [29]. For example, features could be deleted, missing values or new categories could be introduced, or there could be a distributional shift in one or more variables. If an earlier model iteration that was trained on a separate set of samples exists, then a typical example of this step would check for a covariate shift in the feature variables via the function:

```
def M_detectCovariateShift(current_data, prior_data, features):
  covariateShiftFunc, threshold =
    M_selectCovariateShiftFunction(current_data, prior_data)
  covariate_shift_list = []
  for x in features.itercols():
    shift = covariateShiftFunc(current_data[x],
      prior_data[x])
    if shift > threshold:
      covariate_shift_list.append(x)
  return covariate_shift_list, covariateShiftFunc
```

The prior data can be accessed through the list of pointers in the `older_versions` component of the `sift_project` object. If necessary, summary statistics about the current data and prior data can be compared if the full prior data set is not available. It's important to note that checking for a covariate shift in the data may not detect all changes to the underlying data, and additional safeguards should be built-in that are specific to the company's applications. If a change to the underlying data is detected, then the user may decide to exit SIFT and retrain the model, or to continue through the pipeline to see if the change to the data results in a biased outcome. This decision is captured by the function `H_verifyRetrainModel`.

Outcome-involved risk assessment. Like previous pipelines, the last step of the Outcome-involved pipeline asks the user to make a final risk assessment via `H_riskAssessment`. Any fairness concerns that remain in the ML project must be weighed against the cost and feasibility of developing, training, and deploying a new ML model or working with an alternative third-party source. If the user decides to proceed, then the project remains in deployment.

Else, the project is terminated and added to the project database with the appropriate timeout specified.

4 INDUSTRY USE CASES

Several recent examples of industries deploying ML-based products and realizing biases as they manifest later have occurred. A reactive approach of companies redeploying their product with corrective measures leads to cost inflation and negative PR. Our proactive approach in handling bias throughout the ML lifecycle can reduce these concerns. We now show how SIFT addresses potential biases in two representative use cases.

4.1 Marketing

As targeted advertising has become standard in the digital landscape, industrial concerns of unethical or illegal advertising have also arisen. Historically marginalized groups have lost visibility into information in ads related to high paying jobs [13, 14]. Such discrimination may be unintentional on the part of the advertiser or the ad platform, but nevertheless does occur when targeting systems and ad delivery algorithms are applied without careful evaluation along the way [9] of unexpectedly introduced bias [26]. SIFT provides a thorough framework through which such an evaluation can proceed.

Suppose a company wants to identify customers likely to be early adopters of a new service being rolled out among its existing customer base to receive exclusive discounts as part of a promotional marketing campaign. To build an early adopter model, a project team surveys a small sample of customers on the likelihood of service adoption. The team constructs the binary target variable, $\mathbf{y}$, indicating whether each customer is likely to be an early adopter based on their response. As features for the model, the team merges the survey data with marketing data purchased externally. This data is available for the entire customer base, and includes demographic information and consumer segmentation data constructed from social media, online browsing, and purchase data.

Below we describe two sample projects under this setup and their steps through the SIFT pipelines. We summarize these steps in Figure 2. Note that both projects skip the Outcome-involved pipeline because neither involve a deployed model.

As data for these projects, we use the demographic data from the UCI Adult dataset[7] and simulate 50 binary features designed to represent consumer segmentation data. We simulate $\mathbf{y}$ as a binomial random variable with probability depending on a subset of the features. The simulation details are provided in Appendix D. A small subsample is selected that contains only 5% non-white samples in Project 1. The full dataset is used in Project 2.

4.1.1 *SIFT implementation - Project 1.*

Information gathering. Team A initializes a `sift_project` object with project id Svc2020. The corresponding `sift_data` object is populated with the target variable, demographic data, and consumer segmentation data. Then `M_searchExistingProjects` and `H_verifySimilarity` obtain a list of similar projects in the project database. The team connects with other business units working on the identified similar projects and learns from their subject

[7] http://archive.ics.uci.edu/ml

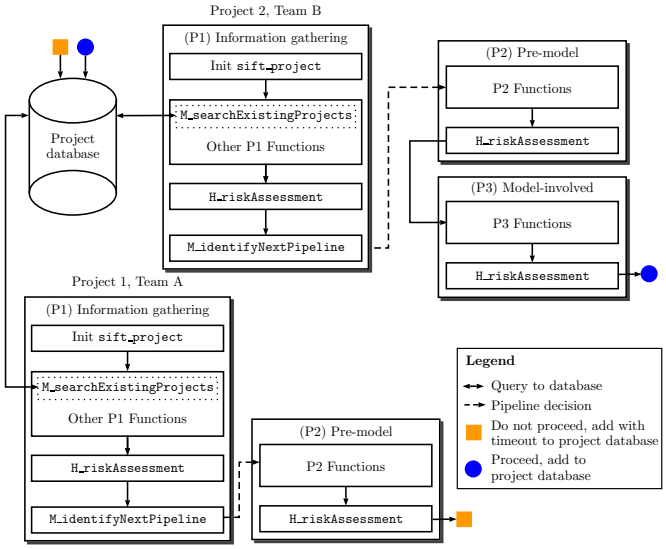

Figure 2: SIFT flow for the two marketing-related projects

matter experts. After considering the collected information, marital status, race, and sex are determined to be sensitive features.

The company would prefer to avoid the bias of offering discounts for the service disproportionally to any of the demographic subgroups identified by the sensitive features. After consulting legal and compliance, `H_riskAssessment` determines that the project should proceed through the SIFT system due to a risk of potential bias. Prior projects used the companies standard model flow, and there are no additional cost or computational constraints for this project. Thus, the standard model flow is selected. Disparate impact is the bias detection metric specified in the standard flow for marketing applications. The specified fairness range is $(0.8, 1.2)$, outside of which bias is detected. No model is deployed, thus `M_identifyNextPipeline` sets the next stage as Pre-model, and the `bias_history` and `model_history` objects are initialized.

Pre-model. After data preparation, `M_detectSparseGroups` checks that each subgroup defined by each sensitive feature makes up at least 10% of the data samples. The output of this function indicates that there is an under-representation of non-white customers in the dataset. The SIFT user decides to collect additional data for the project through the function `H_verifyGetMoreData`. The project is terminated and the `bias_history` is updated accordingly. The project is added to the Project database with a timeout set to 1-year pursuant to the company's settings.

4.1.2 *SIFT implementation - Project 2.*

Information gathering. Six-months later, after collecting additional data from a new survey, Team B initializes a `sift_project` object with project id NewSvc2020, populating `sift_data` with the new data. `M_searchExistingProjects` and `H_verifySimilarity` obtain a list of similar projects in the project database, and a pointer to Svc2020 is added to the list in `older_versions`. Based on the information available from Svc2020, `H_riskAssessment` quickly determines that NewSvc2020 should proceed through

SIFT. Project Svc2020's `sens_features` and model flow selection are copied into the new project. No model is deployed, thus `M_identifyNextPipeline` sets the next stage as Pre-model, and new `bias_history` and `model_history` objects are initialized.

Pre-model. After data preparation, `M_detectSparseGroups` does not not identify any sparse groups in the new dataset. `M_detectProxyFeatures` performs a Chi-Square test for independence on each (sensitive feature, non-sensitive feature) pair and compares the p-value against a Bonferroni corrected threshold of $0.01/m$, where m is the number of non-sensitive features. No proxy features are identified by the function. Lastly, `M_detectBias` computes Disparate Impact between y and each sensitive feature (Table 1, Column 2). All of the results are within the fairness range of (0.8, 1.2), so no marginalized groups are detected. The pipeline updates the `bias_history` after each of these steps with the corresponding bias detection algorithm and result. The SIFT user decides to proceed to the Model-involved pipeline through the function `H_riskAssessment`.

Model-involved. `M_getPreModelBiasStatus` does not indicate any bias was detected in the Pre-model pipeline. The SIFT system proceeds to `M_trainModel`, which splits the data evenly into a training and test set, and trains a logistic regression model. The model has a test-set accuracy of 77.6%. The `model_history` object is updated accordingly. The predicted outcomes for the test-set are passed to `M_detectBias`, which computes Disparate Impact as set by the company's standard flow settings (Table 1, Column 3). The results show that bias is detected on the basis of the sensitive feature sex since 0.79 is outside of the fairness range. Next, as part

Table 1: Bias detection metric results for Project 2

		Disparate Impact	
Sensitive Feature	y	Original Model	Debiased Model
marital_status	0.85	0.82	0.83
race	0.96	0.97	1.00
sex	0.84	0.79	0.88

of standard flow, `M_inProcessingMitigation` applies Adversarial Debiasing [36] to correct the detected bias. The `model_history` object is updated accordingly. A final check using `M_detectBias` confirms the absence of any bias in the predicted outcomes of the debiased model (Table 1, Column 4). The test-set accuracy of the debiased model is 76.2%. These steps are recorded in `bias_history`.

`H_riskAssesment` confirms only a minimal drop in accuracy between the original and debiased models and shares `bias_history` with legal and compliance, who confirm that the bias has been addressed. The `sift_project` object is updated now and returned to the ML projects database as scheduled for deployment.

Given the importance of time-to-market in campaign deployment, copying the information collected by Team A and other similar projects was an important time saving measure for Team B.

4.2 Hiring

Many studies in the past few decades have found evidence of discrimination in hiring practices, especially against women and minorities [16, 31]. A study [16] of 18 algorithmic hiring vendors found that empirical analysis of their algorithms was challenging due to model opaqueness and limited public information.

We apply SIFT in a hiring arena scenario adapting the post-processing mitigation approach of [20] and applying it on the probability scores from a logistic regression model built on the UCI Adult dataset. Suppose a company performs its initial candidate screening using ML models that score and rank all applicants. Based on an existing model used to select candidates for a past position, the company selects a list of top $k = 500$ candidates, say C_k, for further interviews. However, the company first wants to ensure that this list proportionately represents candidates from all relevant sensitive categories, as compared to the full set of candidates C.

4.2.1 SIFT implementation. As the deployed model is available, the project skips the Pre-model and Model-involved pipelines.

Information gathering. The ML team initializes a new `sift_project` and `sift_data`. `M_searchExistingProjects` and `H_verifySimilarity` return an older ML project for past position as similar, which had race, sex, and country of origin as sensitive features. Consulting past team and HR, `H_identifySensitiveCategories` lets team keep same features. `M_identifyNextPipeline` returns "Outcome-involved" as the next stage as model is deployed.

Outcome-involved. The team uses `M_detectCovariateShift` to identify shifted sensitive features. Suppose this detects only race as a shifted feature. Since it is assumed that the previous project performed its own bias detection and mitigation, the new project need only consider the shifted feature. A demographic parity check (`M_detectBias`) compares non-White candidate proportions in C_k and C:

$$P(\text{race}(c) \neq \text{White} | c \in C_k) = 0.068, P(\text{race}(c) \neq \text{White} | c \in C) = 0.1399$$

Then `M_postProcessingMitigation` performs bias mitigation using the re-ranking approach given in Algorithm 1 of [20] to come up with C'_k, a new list of k candidates. Reapplying `M_detectBias` gives the new proportion:

$$P(\text{race}(c) \neq \text{White} | c \in C'_k) = 0.14$$

A final risk assessment considers the full bias history, and moves the candidates in C'_k to the next phase of the recruiting process.

Possible risk factors considered in `H_riskAssessment` at the end of the pipeline include any compliance guidelines (such as [11]), the legal threshold for demographic parity comparisons, and the difference between predicted accuracy in the lists C_k and C'_k. Model reuse from the older, deployed project shows SIFTâĂŹs cost saving advantage enabled by information reuse.

5 DISCUSSION

We have shown how bias detection and mitigation in ML may be done in an enterprise setting in a holistic and transparent manner. Industry's technical challenges include diversity of use cases, datasets, and audiences. Companies have to ensure that no demographic bias arises even well after deployment while adhering to policy recommendations and meeting compliance requirements. While SIFT does not handle all these problems, it shows how a large class of ML projects can follow a framework to reduce chances of bias going undetected until it is too late. Tracking bias information across projects enables new projects to quickly learn about potential bias issues and mitigation techniques that have worked.

For SIFT to be effective, parties involved in multiple phases of a project workflow need to make concerted efforts. Given that ML projects routinely reuse data (often with modifications or sampling) and tweaked models, it is essential to examine the provenance and reliability of the datasets, as well as any past applications and concerns. Incentivizing project managers to enforce adequate data and model documentation helps maintain the accuracy of components in a `sift_project` object. The cost for doing so is traded off against financial risk metrics that approximate the cost of negative PR and brand impact. Long-term adoption of SIFT in the enterprise will reduce projected overall cost. Such explicit analysis of risk vs. return will drive better institutional decision-making.

Decades of security research has taught us that one cannot sprinkle fairy dust on software after development to make it secure. Thus bias monitoring must be done as a *first* step for *every* ML project using bias histories of similar past projects. Risk of *malicious* bias is relatively low within an enterprise. However, if a platform like SIFT is adopted broadly, there is a risk of deliberate attacks against any bias detection system. For example, input data could be manipulated or models tampered with for later exploitation. False bias claims need to be anticipated with transparency playing the defensive role to effectively disprove such claims.

We now address some limitations of SIFT. During the early stages of implementing SIFT in an enterprise there will be few "similar" projects leading to higher per-project cost for bias detection and mitigation. However, as time progresses SIFT's novel reuse of bias history will be progressively more effective, thus significantly diminishing the *overall* cost of bias monitoring to the enterprise. Secondly, it is non-trivial to quickly determine how changes to data or model may impact bias at different stages of the lifecycle. Research remains to be done to compartmentalize the potential impact of such modifications. Effectively, we would like to limit the parts of the lifecycle that would be impacted by any modifications to the input data, changes to the model, application to new use cases etc. Next, for specific use cases, possible ways to mechanize one or more human components in the SIFT pipelines are worth exploring. Lastly, SIFT in its current form will not handle ML models such as adaptive algorithms that are constantly updating and learning from streaming data, or unsupervised learning problems. These types of problems will require different class structures and pipelines, but would still benefit from the knowledge-sharing aspects of SIFT.

REFERENCES

[1] J. A. Adebayo. 2016. *FairML : Toolbox for diagnosing bias in predictive modeling.* Master's thesis. MIT. https://github.com/adebayoj/fairml.

[2] M. Arnold et al. 2018. FactSheets: Increasing Trust in AI Services through Supplier's Declarations of Conformity. *arXiv:1808.07261* (2018).

[3] A. Backurs et al. 2019. Scalable Fair Clustering. In *ICML-2019.*

[4] N. Bantilan. 2018. Themis-ml: A Fairness-Aware ML Interface for End-To-End Discrimination Discovery and Mitigation. *J. Tech. Hum. Serv.* 36, 1 (2018), 15–30.

[5] S. Barocas and A. D. Selbst. 2016. Big data's disparate impact. *Calif. L. Rev.* 104 (2016), 671.

[6] R. K. E. Bellamy et al. 2018. AI Fairness 360: An Extensible Toolkit for Detecting, Understanding, and Mitigating Unwanted Algorithmic Bias. *arXiv:1810.01943* (2018).

[7] F. Calmon et al. 2017. Optimized Pre-Processing for Discrimination Prevention. In *NIPS-2017.*

[8] A. Cavoukian, S. Taylor, and M. E. Abrams. 2010. Privacy by Design: essential for organizational accountability and strong business practices. *Identity Inform. Soc.* 3, 2 (2010), 405–413.

[9] E. Celis, A. Mehrotra, and N. Vishnoi. 2019. Toward Controlling Discrimination in Online Ad Auctions. In *ICML-2019.*

[10] L. E. Celis et al. 2018. Classification with Fairness Constraints: A Meta-Algorithm with Provable Guarantees. *arXiv:1806.06055* (2018).

[11] Code of Federal Regulations. 2017. *Uniform Guidelines on Employee Selection Procedures.* https://www.govinfo.gov/content/pkg/CFR-2017-title29-vol4/xml/CFR-2017-title29-vol4-part1607.xml

[12] B. D'Alessandro, C. O'Neil, and T. LaGatta. 2017. Conscientious Classification: A Data Scientist's Guide to Discrimination-Aware Classification. *Big Data* 5, 2 (2017), 120–134.

[13] A. Datta et al. 2018. Discrimination in Online Advertising: A Multidisciplinary Inquiry. In *FAT-2018.*

[14] A. Datta, M. C. Tschantz, and A. Datta. 2015. Automated Experiments on Ad Privacy Settings. *Priv. Enh. Technologies* 2015, 1 (2015), 102–112.

[15] S.-C. Davies, E. Pierson, A. Feller, et al. 2017. Algorithmic decision making and the cost of fairness. In *KDD-2017.*

[16] Raghavan et al. 2019. Mitigating Bias in Algorithmic Employment Screening: Evaluating Claims and Practices. arXiv:1906.09208.

[17] M. Feldman et al. 2015. Certifying and Removing Disparate Impact. In *KDD-2015.*

[18] S. Galhotra, Y. Brun, and A. Meliou. 2017. Fairness Testing: Testing software for discrimination. In *ESEC/FSE-2017.*

[19] T. Gebru et al. 2018. Datasheets for datasets. *arXiv:1803.09010* (2018).

[20] S. C. Geyik, S. Ambler, and K. Kenthapadi. 2019. Fairness-Aware Ranking in Search & Recommendation Systems with Application to LinkedIn Talent Search. In *KDD-2019.*

[21] M. Hardt, E. Price, and N. Srebro. 2016. Equality of Opportunity in Supervised Learning. In *NIPS-2016.*

[22] K. Holstein et al. 2019. Improving fairness in machine learning systems: What do industry practitioners need?. In *CHI-2019.*

[23] F. Kamiran and T. Calders. 2012. Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems* 33, 1 (2012), 1–33.

[24] F. Kamiran, A. Karim, and X. Zhang. 2016. Decision Theory for Discrimination-Aware Classification. In *NIPS-2016.*

[25] T. Kamishima et al. 2012. Fairness-Aware Classifier with Prejudice Remover Regularizer. In *Machine Learning and Knowledge Discovery in Databases.*

[26] A. Lambrecht and C. E. Tucker. 2019. Algorithmic Bias? An Empirical Study into Apparent Gender-Based Discrimination in the Display of STEM Career Ads. *Manage. Sci.* 65, 7 (2019), 2966–2981.

[27] N. Mehrabi et al. 2019. A Survey on Bias and Fairness in Machine Learning. *arXiv:1908.09635* (2019).

[28] N. Mehrabi et al. 2019. Debiasing Community Detection: The Importance of Lowly Connected Nodes. In *ASONAM-2019.*

[29] M. Mitchell et al. 2019. Model Cards for Model Reporting. In *FAT*-2019.*

[30] G. Pleiss et al. 2017. On Fairness and Calibration. In *NIPS-2017.*

[31] L. Quillian et al. 2017. Meta-analysis of field experiments shows no change in racial discrimination in hiring over time. *Proc. Natl. Acad. Sci.* 114 (2017), 10870–10875.

[32] T. Speicher et al. 2018. Potential for Discrimination in Online Targeted Advertising. In *FAT-2018.*

[33] A. Tramer et al. 2017. FairTest: Discovering unwarranted associations in data-driven applications. In *EuroS&P-2017.*

[34] M. Veale, M. Van Cleek, and R. Binns. 2018. Fairness and accountability design needs for algorithmic support in highstakes public sector decision-making. In *CHI-2018.*

[35] R. Zemel et al. 2013. Learning Fair Representations. In *ICML-2013.*

[36] B. Zhang, B. Lemoine, and M. Mitchell. 2018. Mitigating Unwanted Biases with Adversarial Learning. In *AAAI-2018.*

[37] L. Zhang, Y. Wu, and X. Wu. 2017. A Causal Framework for Discovering and Removing Direct and Indirect Discrimination. In *IJCAI-2017.*

APPENDIX

A CLASS COMPONENTS AND METHODS

We provide additional information about the `sift_data` class components, `bias_history` methods, and the `model_history` class components here.

A.1 SIFT data class components

The list of components of the `sift_data` class are:

- `raw_data` – a connection to the data for the project, such as a dataframe or a connection to a distributed file system that provides access to the data,
- `data_definitions` – a dictionary with definitions for each variable in the dataset,
- `y` – a variable with the name of the response variable,
- `X` – a list with the names of the predictors,
- `outcome` – a connection to the predicted outcomes from the ML model; This will be pre-populated if the model is deployed else populated after model training,
- `sens_features` – a list with the names of the sensitive features that contain categories that might suffer from biased treatment,
- `sens_features_summary` – a dictionary of dictionaries where each key denotes a specific characteristic relevant to bias-investigation, such as sparse groups, proxy features, or marginalized groups, and each corresponding value defines a dictionary with keys denoting sensitive features as identified in `sens_features` and values being lists describing the corresponding characteristics.

The components of the `sift_data` class may be populated manually or via mechanizable extraction functions that take the `data_location` and project description as arguments.

A.2 Bias history class methods

During the course of the project, the `bias_history` object is updated through methods associated with the class to reflect the bias detection and mitigation steps performed. To get the current step we use the method:

```
def getLatestStep(sift_project):
  return sift_project.bias_history[-1]['step']
```

Components are added to the current step to track bias investigation at each step using the class method:

```
def insertBiasHistoryAt(sift_project, insert_at, **kwargs):
  if (insert_at > sift_project.getLatestStep()):
    return 'cannot insert outside current history range'
  #      fill in components and corresponding values from **kwargs
  for key, value in kwargs.item():
    if key in list(sift_project.bias_history[0].keys()):
      sift_project.bias_history[insert_at][key] = value
    else:
      print(f'{key} is not an attribute of bias history')
```

Lastly, the next step in the bias history is added using the class method:

```
def addBiasHistoryStep(sift_project, **kwargs):
  # append new step
  sift_project.bias_history.append(
    {'step'                    :sift_project.getLatestStep() + 1,
    'sift_pipeline'            :None,
    'bias_features'            :None,
    'bias_detection_function'  :None,
    'bias_mitigation_function' :None,
    'mitigation_success_status':None,
    'details'                  :None})

  #      fill in components and corresponding values from **kwargs
  for key, value in kwargs.item():
    if (key != 'step') and
      (key in  list(sift_project.bias_history[0].keys())):
      insert_at = sift_project.getLatestStep()
      sift_project.bias_history[insert_at][key] = values
```

A.3 Model history class components

The SIFT model history class keeps track of the modeling efforts. The list of components of the `model_history` class are:

- `step` – a counter capturing the place in the sequence of modeling tasks performed,
- `seed` – the random seed used in the modeling process to ensure reproducibility,
- `train_index` – the set of indices in the raw data used for training the ML model,
- `test_index` – the set of indices in the raw data used for testing the ML model,
- `fitted_model` – includes the loss function to be optimized and its value for the fitted model, the tuning parameters, and the estimated model,
- `perf_metric` – a dictionary that includes the name of the performance metric used to evaluate the model and the performance metric value for the test-set; For example, the performance metric for a classification problems could be accuracy, precision, recall, etc.,
- `is_deployed` – a flag indicating if the model is deployed. If `True` at step 0, then this indicates that the project was initiated with an earlier model already deployed.

B AUXILIARY FUNCTIONS

We provide descriptions of the auxiliary functions that appear in Section 3. We assume these functions would be standardized by the company, but could depend on the specific data or project application and could be overridden by the user when necessary. We provide examples of these functions based on their implementation in the use cases in Section 4.

`M_selectSparsityFunction`: This selects the sparsity function and its threshold. The sparsity function calculates the sparsity of a particular subgroup of a sensitive feature. For example, the sparsity function could compute the percent of samples that belong to the subgroup.

`M_selectDependenceFunction`: This selects the dependence function and its threshold. The dependence function computes the statistical dependence between a sensitive attribute and non-sensitive attribute in order to identify non-sensitive features that might act as a proxy for other sensitive categories. For example, the dependence function could return the p-value from a Chi-Square test of independence between the sensitive feature and a non-sensitive feature.

`M_selectBiasOutcomeFunction`: This selects the function to compute the bias detection metric for the model outcome and its corresonding fairness ranges. As an example, `M_selectBiasOutcomeFunction` selects a function that computes Disparate Impact in the marketing use case.

M_dataSplit: This function sets the random seed and splits the data into a training and test set using the specified split ratio.
M_selectModel: This function selects the ML model and performance metric. For example, in the marketing use case, this selects logistic regression as the ML model and classification accuracy as the performance metric.
M_selectCovariateShiftFunction: This selects the covariate shift function, which calculates the shift in distribution of a feature from one period to the next. For example, the covariate shift function could compute the Kullback-Leibler distance between two features.

C EXAMPLE STANDARD FLOW

Here we provide a sample standard flow process that works sequentially through the six steps described in Section 3.3. For ease of presentation, this example assumes there is only one sensitive feature.

```
premodel_status = M_getPreModelStatus(
  current_project.data.sens_features_summary)
if premodel_status:
  # Step: Pre-processing mitigation
  current_project.data, preProcFunc =
    M_preProcessingMitigation(current_project)
  ## Update bias history and add a step
  current_bias_step = getLatestStep(current_project)
  current_project.insertBiasHistoryAt(current_bias_step,
    **{'bias_features' = current_project.data.sens_features,
    'bias_mitigation_function' = preProcFunc.__name__})
  current_project.addBiasHistoryStep(
    **{'sift_pipeline' = "Model-involved"})
# Step: Model training
fitted_model = M_trainModel(current_project)
## Update model history and add a step
...
# Step: Model outcome bias detection
biased_groups_list, biasFunc = M_detectBiasModelOutput(
  current_project.data.outcome,
  current_project.data.raw_data[current_project.data.sens_features])
## Update bias history
current_bias_step = getLatestStep(current_project)
current_project.insertBiasHistoryAt(current_bias_step,
  **{'bias_features' = current_project.data.sens_features,
  'bias_detection_function' = biasFunc.__name__})
if not biased_groups_list:
  ## Set mitigation_success_status in previous step to TRUE
  ...
else:
  ## Set mitigation_success_status in previous step to FALSE
  current_project.insertBiasHistoryAt(current_bias_step - 1,
    **{'mitigation_success_status' = FALSE})
  # Step: In-processing mitigation
  new_fitted_model, inProcFunc =
    M_inProcessingMitigation(current_project)
  ## Update bias history and add a step
  current_project.insertBiasHistoryAt(current_bias_step,
    **{'bias_mitigation_function' = inProcFunc.__name__}
  current_project.addBiasHistoryStep(
    **{'sift_pipeline' = "Model-involved"})
  ## Update model history
  ...
  # Step: Model outcome bias detection
  biased_groups_list, biasFunc = M_detectBiasModelOutput(
    current_project.data.outcome,
    current_project.data.raw_data[current_project.data.sens_features])
  ## Update bias history
  current_bias_step = getLatestStep(current_project)
  current_project.insertBiasHistoryAt(current_bias_step,
    **{bias_features' = current_project.data.sens_features,
    'bias_detection_function' = biasFunc.__name__})
  if not biased_groups_list:
    ## Set mitigation_success_status in previous step to TRUE
    ...
  else:
    ## Set mitigation_success_status in previous step to FALSE
    current_project.insertBiasHistoryAt(current_bias_step - 1,
      **{'mitigation_success_status' = FALSE})
    # Step: Post-processing mitigation
    current_project.data.outcome, postProcFunc =
      M_postProcessingMitigation(current_project)
    ## Update bias history
    current_project.insertBiasHistoryAt(current_bias_step,
      **{'bias_mitigation_function' = postProcFunc.__name__})
    # Step: Model outcome bias detection
    bias_status, biasFunc = M_detectBiasModelOutput(
      current_project.data.outcome,
      current_project.data.raw_data[current_project.data.sens_features])
    ## Update bias history
    current_bias_step = getLatestStep(current_project)
    current_project.insertBiasHistoryAt(current_bias_step,
      **{'bias_features' = current_project.data.sens_features,
      'bias_detection_function' = biasFunc.__name__,
      'mitigation_success_status' = (len(biased_groups_list) == 0)})
# Step: Model-involved risk assessment
go_ahead_status = H_riskAssessment(current_project)
if go_ahead_status is "Do not proceed":
  current_project.metadata['project_status'] = "Terminated"
  current_project.timeout = company_timeout
else:
  current_project.metadata.project_status =
    "Scheduled for deployment"
```

D SIMULATION DETAILS FOR MARKETING USE CASE

We use demographic data from the UCI Adult dataset and remove all examples with missing information, resulting in $n = 45,222$ examples. In our experiments, we use income, sex,

- age - binned as [17, 25], [26, 35], [36, 45], [46, 55], [56, 65], [66, 75], or 75+, and converted to its one hot encoding,
- marital_status - converted to "married" or "single",
- race - converted to "white" or "non-white".

We treat {marital_status, race, sex} as the set of sensitive features.

To generate consumer segments that are correlated with the sensitive features, we simulate $C^c_{i,j} \sim Bin(\tilde{p_i})$ for $j = 1, \ldots, 5$ and $i = 1, \ldots, n$, where

$$\tilde{p_i} = \frac{\exp\left(-1 \ + \ \mathbb{1}_{\{marital\_status_i = Married\}} \ + \ \mathbb{1}_{\{sex_i = Male\}}\right)}{1 + \exp\left(-1 \ + \ \mathbb{1}_{\{marital\_status_i = Married\}} \ + \ \mathbb{1}_{\{sex_i = Male\}}\right)}.$$

In addition, we simulate consumer segments $C^u_{i,j} \sim Bin(p_j)$ for $j = 1, \ldots, 45$ and $i = 1, \ldots, n$, where $p_j \sim U(0.2, 0.8)$, to be consumer segments that are uncorrelated with the sensitive features.

We define $\mathbf{X} = [\text{age}, \text{income}, \mathbf{C^c}, \mathbf{C^u}]$ to be the set of features for model training. To simulate the target variable, $\mathbf{y}$, we define $\boldsymbol{\beta}$ to be a vector of coefficients corresponding to the features in $\mathbf{X}$. We simulate coefficients for income, $\mathbf{C^c}$, and the first 10 features in $\mathbf{C^u}$ from $U(-2, 2.5)$. We set all other coefficients to zero. Then $y_i \sim Bin(p^y_i)$ for $i = 1, \ldots, n$, where

$$p^y_i = \frac{exp(-0.5 \ + \ \mathbf{X}_i\boldsymbol{\beta} \ + \ z_i)}{1 + exp(-0.5 \ + \ \mathbf{X}_i\boldsymbol{\beta} \ + \ z_i)}.$$

Here $z_i \sim N(0, 1)$, for $i = 1, \ldots, n$, prevents a perfect model fit.