Learning to Validate the Predictions of Black Box Classifiers on Unseen Data

Sebastian Schelter
sebastian.schelter@nyu.edu
New York University

Tammo Rukat
tammruka@amazon.com
Amazon Research

Felix Biessmann
fbiessmann@beuth-hochschule.de
Beuth University Berlin

ABSTRACT

Machine Learning (ML) models are difficult to maintain in production settings. In particular, deviations of the unseen serving data (for which we want to compute predictions) from the source data (on which the model was trained) pose a central challenge, especially when model training and prediction are outsourced via cloud services. Errors or shifts in the serving data can affect the predictive quality of a model, but are hard to detect for engineers operating ML deployments.

We propose a simple approach to automate the validation of deployed ML models by estimating the model's predictive performance on unseen, unlabeled serving data. In contrast to existing work, we do not require explicit distributional assumptions on the dataset shift between the source and serving data. Instead, we rely on a programmatic specification of typical cases of dataset shift and data errors. We use this information to learn a *performance predictor* for a pretrained black box model that automatically raises alarms when it detects performance drops on unseen serving data.

We experimentally evaluate our approach on various datasets, models and error types. We find that it reliably predicts the performance of black box models in the majority of cases, and outperforms several baselines even in the presence of unspecified data errors.

ACM Reference Format:
Sebastian Schelter, Tammo Rukat, and Felix Biessmann. 2020. Learning to Validate the Predictions of Black Box Classifiers on Unseen Data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3318464.3380604

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3380604

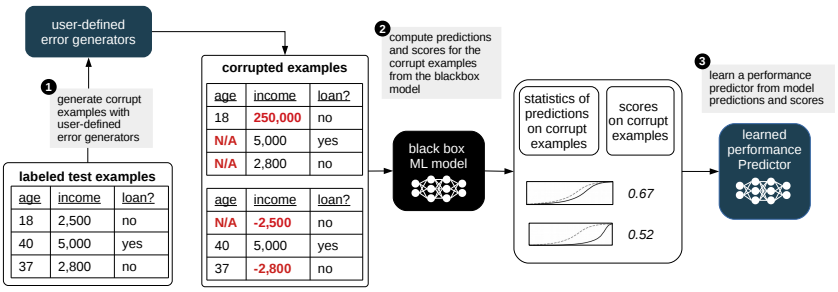
1 INTRODUCTION

Machine learning (ML) has become a central component in modern software systems. Yet, the maintenance of ML applications remains challenging [12, 19, 22]. Many problems in this space are due to unexpected shifts or errors in the data that is fed into ML models at prediction time. These shifts and errors typically originate from bugs in data preprocessing code or changes in the data generating process in the real world. This problem is exacerbated in situations where different parties are involved in the provision of the data and the training of the model. Many engineering teams, especially in smaller companies, lack ML expert knowledge, and therefore often outsource the training of ML models. They might either use cloud services for model training and hosting, such as *Google AutoML Tables*, or locally apply AutoML libraries, which automate model training. In such cases, the engineering team provides input data and retrieves predictions, but the details of the model may not be accessible.

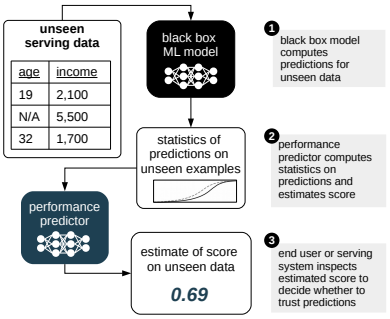
Consider the following exemplary scenario: An engineering team of an e-commerce company leverages a cloud ML service to train and deploy an ML model for predicting the sales numbers of competitor products. Some time later, an engineer accidentally introduces errors in the preprocessing code that prepares new serving data for the model, and the error changes the scale of certain numeric attributes. The deployed cloud model will still output predictions for unseen competitor products, but its predictions cannot be relied upon anymore, as its weights are not properly fitted to the changed serving data. Unfortunately, it is difficult for the engineers to validate the predictions they retrieve from the black box model in the cloud, as there is no automated way to retrieve the ground truth (the future sales numbers of the competitor in this scenario).

While ML experts have specialized knowledge to debug models and predictions in such cases [2, 9, 13], *there is a lack of automated methods for non-ML expert users to decide whether they can rely on the predictions of an ML model on unseen data.*

We tackle this issue by introducing the performance prediction problem for black box classifiers in Section 2. Next, we propose a simple approach to automatically validate the predictions of a pretrained ML model by estimating its predictive performance on unseen, unlabeled serving data. In



(a) Learning a performance predictor for a pretrained black box classifier, using synthetically corrupted data to explore how data errors impact the resulting prediction quality.



(b) Using the performance predictor to estimate the accuracy on unseen serving data.

Figure 1: Overview of our proposed approach: (*a*) We learn a *performance predictor* from synthetically corrupted input data to estimate the prediction quality of a black box model on a batch of serving data from statistics of its outputs. (*b*) This performance predictor is subsequently leveraged to estimate the prediction quality of the black box model on unseen, unlabeled serving data. End users and serving systems can raise alarms if this estimate is significantly below the expected prediction quality of the black box model.

contrast to existing work, we do not require explicit distributional assumptions on the dataset shift between the source and serving data nor force the user to define distance functions and thresholds between source and serving data. Instead, an engineer programmatically specifies typical cases of dataset shift and data errors that they expect to observe in real world data. Based on this information, we learn a *performance predictor* for a pretrained black box model to be deployed along with the original model (Section 3). The performance predictor can automatically raise alarms if it detects performance drops on unseen serving data.

In contrast to previous work [1, 20], our approach does not require access to features extracted from the raw input data but only to the model outputs, and can thus treat the ML model as a black box. This property makes it suitable for the automated monitoring of the aggregate prediction quality in ML pipelines.

We show how to implement our approach using popular python-based ML libraries in Section 4 and discuss related work in Section 5. Finally, we demonstrate its effectiveness on a variety of real world datasets in an extensive evaluation, which covers several models that consume relational data, text data and image data (Section 6).

The contributions of this paper are the following:

- We introduce the performance prediction problem for black box classifiers (Section 2).
- We discuss how to automatically estimate the prediction scores of black box classifiers on unseen, unlabeled, and potentially shifted or erroneous serving data (Section 3).
- We describe how to implement our approach using a popular ML library (Section 4).
- We experimentally evaluate our approach on various datasets, models and error types, including an external cloud service. We find that it reliably predicts the performance of black box models in the majority of cases, and outperforms several baselines in the performance prediction task, even in the presence of unspecified data errors (Section 6).

2 OVERVIEW & PROBLEM STATEMENT

Overview. Figure 1(b) illustrates how we address the problem of deciding whether we can rely on the predictions of a black box model on unseen, unlabeled serving data. Specifically, we want to know whether the predictions of the black box model on a batch of serving data have the same prediction quality (e.g., accuracy) as its predictions on the training data ❶ (which we would generally expect due to the generalization capabilities of a well-trained classifier). However, we cannot compute the true accuracy of the black box model on the unseen data, as we do not have access to the true labels. Therefore, we provide a learned performance predictor (tailored to the black box model), which estimates the accuracy of the black box model's predictions on the serving data ❷. This estimate can then be used to decide whether to rely on the predictions of the black box model ❸.

Figure 1(a) gives an overview of how we learn the performance predictor for a given black box model. We have an end user specify potential errors (e.g., missing values, encoding errors, etc.) that they might expect to see in the serving data. They need to specify the type of expected errors, but not their magnitude. They can draw upon their professional experience to decide upon such potential errors.

Based on these potential errors, we generate synthetically corrupted input datasets by applying error generators to the test set that was used for training the black box model ❶. We record the prediction score (which we can compute in this case as we have access to the labels in the test set) as well as statistics of the distribution of the model's outputs for each synthetically corrupted dataset ❷. We define a regression problem in which the accuracy of the black box model is predicted from statistics of its outputs. We refer to this regression model as "performance predictor" ❸.

Problem Statement. We formalize the problem addressed in this paper, before we describe our approach in detail in the next section.

Data. We are given relational data $\mathcal{D}$ with attributes $A = \{A_1, \ldots, A_N\}$ that is comprised of tuples $t \in \mathcal{D}$. The black box model is trained on the source dataset $\mathcal{D}_{\text{source}}$ and will afterwards be applied to the unseen serving dataset $\mathcal{D}_{\text{serving}}$. These datasets are disjunct $\mathcal{D}_{\text{source}} \cap \mathcal{D}_{\text{serving}} = \emptyset$ and tuples are assigned independently and at random (i.i.d.).

Black Box Model. We assume that we are provided with a black box classifier trained on a set $\{(t, y)\}$ of examples $t \in \mathcal{D}_{\text{source}}$ and labels $y \in C = \{c_1, \ldots, c_m\}$. The black box character of the model corresponds to the fact that the model applies an *unknown feature map* $\phi : A \to \mathbb{R}^d$ to compute a numerical representation of the relational input data, and that the model applies an *unknown prediction function* $f : \mathbb{R}^d \to \Delta^m$ to compute predictions for m classes from the data. We assume, however, that the model optimizes a *known scoring function* for n examples with known labels $L : \Delta^{n \times m} \times C^n \to [0, 1]$, such as accuracy.

Perturbations. We furthermore assume that there exist perturbations e which potentially introduce errors into data. We can think of them as parameterized functions, e.g., introducing missing values in certain columns with a given probability p. These missing values could for example be accidentally introduced through errors in data integration code. Further examples for errors are swapped columns, e.g., introduced through buggy input forms [8], or scaled numerical columns, e.g., when an engineer would unintentionally switch to a scale in milliseconds instead of seconds for a given feature column. We assume that the set of perturbations $E = (e_1, \ldots, e_q)$ is *known* but the probabilities of occurrence $\mathbf{p}_{\text{err}} = [p_1, \ldots, p_q]$ are *unknown* (e.g., we anecdotally know which errors might potentially occur in ML pipelines, but not how frequent they are). Note that the absence of errors is represented by $\mathbf{p}_{\text{err}} = \mathbf{0}$.

Performance Prediction Problem. Let $\mathcal{D}_{\text{maybe\_corrupt}}$ denote the result of applying the perturbations E to $\mathcal{D}_{\text{serving}}$ with unknown $\mathbf{p}_{\text{err}}$ (e.g., serving data for the model that was accidentally corrupted by buggy preprocessing code). Note that we do not know the labels $\mathbf{y}_{\text{serving}}$ for the unseen serving dataset, which means that we cannot compute the prediction quality $L(f(\phi(\mathcal{D}_{\text{maybe\_corrupt}})), \mathbf{y}_{\text{serving}})$ on the unseen (and potentially corrupted) serving data.

The predictions on the serving data should not be used if the prediction quality on the serving data is significantly lower than the quality achieved on a held-out sample from the source data during training. Our *performance predictor* addresses this as a regression problem, estimating the prediction quality $L(f(\phi(\mathcal{D}_{\text{maybe\_corrupt}})), \mathbf{y}_{\text{serving}})$ on the serving dataset without knowing the ground truth $\mathbf{y}_{\text{serving}}$. This estimate can subsequently be used to decide whether to rely on the predictions on the serving data or not.

The problem at hand can also be treated as a binary classification problem if we assume a threshold t for the relative quality loss that a user would be willing to accept (e.g., 5% compared to the training scenario). In this case, to which we refer as *performance validation*, the goal is to decide whether the prediction quality on the unseen unlabeled serving data falls within this threshold: $L(f(\phi(\mathcal{D}_{\text{maybe\_corrupt}})), \mathbf{y}_{\text{serving}}) \geq (1 - t) \cdot L(f(\phi(\mathcal{D}_{\text{test}})), \mathbf{y}_{\text{test}})$.

Note that the described performance prediction problem is different from existing work, as we (*i*) do not impose distributional assumptions on the shift between source and serving data, (*ii*) do not assume knowledge of the feature space that the model uses, and (*iii*) only require access to the model predictions. In our scenario, it is hard to apply generic dataset shift detection methods because the feature map ϕ is unknown and the model could for example ignore some attributes completely or transform them in away that is independent of a perturbation. Examples for such cases are ignored features due to L1 regularization, or feature maps such as binarization that just check whether a feature is different from zero and therefore scale invariant.

3 APPROACH

We describe the procedure for learning to predict performance via a regression model. Our approach is based on the idea of analyzing the distribution of the model outputs as a signal for potential data errors and shifts. We build on work from the ML community [13], which addresses data shifts with a strict distributional assumption. In contrast to this work, we provide an estimate of the expected model accuracy on erroneous data, and do not require users to manually specify thresholds for hypothesis tests. Instead, we directly predict the accuracy of the black box model.

Training the performance predictor. The steps for training our performance predictor h are shown in Algorithm 1. We assume that we are provided with a black box classifier f which has been trained on the source data. Furthermore, we assume to have access to a set of error generators E that

Algorithm 1 Training a performance predictor h for a pretrained black box model f.

Input: $(\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}})$ disjunct partitions of $\mathcal{D}_{\text{source}}$, black box model f and feature map ϕ pretrained on $\mathcal{D}_{\text{train}}$, user-specified error generators E

1: $(\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}}) \leftarrow \mathcal{D}_{\text{test}}$
2: $\ell_{\text{test}} \leftarrow L(f(\phi(\mathbf{X}_{\text{test}})), \mathbf{y}_{\text{test}})$ ▷ score on test data
3: $M \leftarrow \emptyset$
4: **for** error generator $e \leftarrow E$ **do**
5: $\quad \mathbf{X}_{\text{corrupt}} \leftarrow$ corrupt $\mathbf{X}_{\text{test}} \in \mathcal{D}_{\text{test}}$ with e
6: $\quad$ ▷ score on corrupted examples
7: $\quad \hat{\mathbf{Y}}_{\text{corrupt}} \leftarrow f(\phi(\mathbf{X}_{\text{corrupt}}))$
8: $\quad \ell_{\text{corrupt}} \leftarrow L(\hat{\mathbf{Y}}_{\text{corrupt}}, \mathbf{y}_{\text{test}})$
9: $\quad$ ▷ statistics of model outputs
10: $\quad \zeta_{\text{corrupt}} \leftarrow$ prediction_statistics$(\hat{\mathbf{Y}}_{\text{corrupt}})$
11: $\quad M \leftarrow M \cup (\zeta_{\text{corrupt}}, \ell_{\text{corrupt}})$
12: **end for**
13: $h \leftarrow$ train a regression model on M
14: **return** h

produce certain types of randomized dataset shifts. We apply each error generator e to held-out data $\mathcal{D}_{\text{test}}$ from the training process, and thereby generate corrupted examples $\mathbf{X}_{\text{corrupt}}$ in line 5. We compute the actual prediction score ℓ_{corrupt} by comparing the model predictions $\hat{\mathbf{Y}}_{\text{corrupt}}$ with the true labels $\mathbf{y}_{\text{test}}$ in line 8. Existing work suggests that the univariate distributions of the columns of the matrix $\hat{\mathbf{Y}} = f(\phi(\mathbf{X}))$ are predictive of dataset shifts [13, 20]. We build on these findings, and leverage a univariate non-parametric estimate of the distributions of each output dimension of f. More concretely, we compute percentiles ζ_{corrupt} of the black box model outputs $\hat{\mathbf{Y}}_{\text{corrupt}} = f(\phi(\mathbf{X}_{\text{corrupt}}))$ as features for the performance predictor h. We record both the features ζ_{corrupt} and the score as training data for our regression model in the set M (lines 3-12). Finally, we train our performance predictor h on these examples in line 13 to predict the score ℓ_{corrupt} from the features ζ_{corrupt}.

Performance prediction on unseen serving data. Algorithm 2 details an example of how to apply our performance predictor h to unseen (and unlabeled) serving data $\mathbf{X}_{\text{serving}}$. First, we compute the required features ζ_{serving} in the form of percentiles of the outputs $\hat{\mathbf{Y}}_{\text{serving}}$, which the model f assigns to the examples $\mathbf{X}_{\text{serving}}$. Next, we have our performance predictor h estimate the score $\hat{\ell}_{\text{serving}} = h(\zeta_{\text{serving}})$ of f on the serving data.

We can alternatively learn a *performance validator* that predicts whether the relative quality drop exceeds a user-defined threshold (Section 2). Given a user-defined threshold t for an acceptable performance loss in comparison to score

Algorithm 2 Performance prediction for unlabeled serving data.

Input: serving data $\mathbf{X}_{\text{serving}}$, black box model f and feature map ϕ, performance predictor h

$\hat{\mathbf{Y}}_{\text{serving}} \leftarrow f(\phi(\mathbf{X}_{\text{serving}}))$
▷ statistics of model outputs
$\zeta_{\text{serving}} \leftarrow$ prediction_statistics$(\hat{\mathbf{Y}}_{\text{serving}})$
$\hat{\ell}_{\text{serving}} \leftarrow h(\zeta_{\text{serving}})$ ▷ predicted score on serving data
return $\hat{\ell}_{\text{serving}}$

on held-out test data (e.g., 5%), we turn performance prediction from a regression problem (predicting the score) into a binary classification problem (asking whether the score is within the threshold t of the expected score). We train a specialized classification model, which uses our performance predictions and leverages additional features such as the results of hypothesis tests on model outputs between source and serving data [13]. The major difference of the performance validator to Algorithms 1 & 2 is that we need to retain the predictions $\hat{\mathbf{Y}}_{\text{test}}$ which the black box model gave on the test set, as we need them to compute our features (e.g., the results of hypothesis tests) later.

4 IMPLEMENTATION

We implement our approach in Python, building upon the established python-based ML libraries scikit-learn [17] and the pandas ecosystem [14]. We assume that we are provided with a black box ML model that ingests relational data from a pandas dataframe and outputs predictions. This black box model only needs to be provided in an executable manner (e.g., as a binary file or as a network service). We only require that the model returns the predicted class probabilities for serving data (e.g., via the `predict_proba` method in python-based ML libraries).

Furthermore, we assume that users indicate the types of errors which they expect to potentially occur (e.g., missing values in certain columns). To this end, they can either choose from error generators provided in our library or implement their own in a few lines of python code by implementing the `corrupt` method of an abstract `ErrorGen` base class. In general, an error generator for our approach simply iterates over the rows of a pandas dataframe and randomly corrupts selected columns. Thereby, users can leverage the full expressivity of Python to generate errors. We illustrate how to simulate missing values and encoding errors:

```
class MissingValues(ErrorGen):
 def corrupt(self, data, column, prob)
  for row in data:
    if random() < prob:
      row[column] = NA # Introduce missing values
```

```
class EncodingErrors(ErrorGen):
  def corrupt(self, data, column, prob):
      for row in data:
        if random() < prob:
          row[column] = row[column]
            .replace('E', 'É') # Introduce
            .replace('ö', 'œ') # encoding
            .replace('ü', 'œ') # errors
```

We implement Algorithm 1 by learning a random forest-based regression model as performance predictor. Our approach generates the features for this predictor as follows. We apply the corresponding error generators with randomly chosen magnitudes on a held-out part of the black box model's training data. These error generators copy the dataframe and randomly inject the specified errors into the data. We then compute the model outputs via the `predict_proba` method of the model, as well as the true prediction score on the corrupted data. We featurize the model outputs by computing their class-wise percentiles (collecting the 0th, 5th, 10th, ... percentile). Finally, we train a `RandomForestRegressor` with five-fold cross-validation, and grid search over the number of trees. The objective of this regressor is to minimize the mean absolute error between the predicted scores and the true scores. We then use the resulting regression model as our performance predictor.

The performance validator (Section 2) is implemented analogously, except that we use a gradient-boosted decision tree [4] as prediction model. We compute additional features based on Kolmogorov-Smirnov tests between the model outputs on the corrupted serving data and the held-out test data during model training

5 RELATED WORK

Applying data management techniques in the ML space is a field with growing interest in recent years [12, 19, 21, 22, 24, 25]. Several solutions were proposed for validating ML models and their predictions. Most of these originate from a statistical ML or a data management perspective.

The consequences of a general shift in the joint distribution of features and labels are virtually impossible to predict or mitigate. For an extreme case, consider data that was generated with an adversarial intent or data from a distribution that may not even share support with the source distribution. Hence, methods that emphasize statistical rigor usually start with distributional assumptions on the nature of the dataset shift [28]. Examples include *label shift*, where the marginal label distribution $p(y)$ changes while the conditional feature distribution $p(x|y)$ remains constant [9, 13, 31], and *covariate shift*, where $p(x)$ changes while $p(y|x)$ remains constant [2, 27, 29]. Such assumptions often seem inapt to describe practically relevant data changes for engineers, such as errors originating from software bugs in preprocessing pipelines, unexpected missing values, or a mix of covariate and label shifts in subsets of the data. Moreover, the proposed methods often limit themselves to adapting a particular model or learning paradigm.

Approaches from the data management community [10, 19, 23] often require manual tuning, detailed knowledge of model internals and features, or do not directly address the problem of validating model predictions. For instance, Google's TFX platform [1] offers skew detection capabilities for input data, but requires the end user to manually select an appropriate distance function and a corresponding threshold for particular features.

On a related note, there is a growing body of work on model diagnosis [6] and model unit testing for neural networks [3, 15, 18], which aims to find edge cases and inputs that crash the model. We proposed "unit tests for data" [23] in ML pipelines, but did not connect these tests to the prediction quality of ML models. Wang et al. propose "Uni-Detect" [30], an approach for automated error detection in tabular data, but do not quantify the errors' impact on the predictions of ML models.

6 EVALUATION

Overview. We experimentally evaluate our performance prediction and validation approaches under known and unknown shifts and errors (Sections 6.1 & 6.2) and compare them to several baselines. Afterwards, we demonstrate that our approach also works for models trained by automatic machine learning (AutoML) methods in Section 6.3.

Datasets. We experiment on six publicly available datasets from different domains. (1) The *income*[1] dataset contains 48,842 records about adult income data, and the target variable denotes whether a person earns more than 50K dollars per year or not. (2) The *heart*[2] dataset contains 70,001 records about cardiovascular diseases, and the target variable denotes the presence of a heart disease. (3) The *bank*[3] dataset comprises of 45,212 records of bank customer data, and the goal is to predict if a customer will subscribe a term deposit. (4) The *tweets*[4] dataset comprises of 20,002 tweets and the target variable denotes whether or not a tweet has trolling character (e.g., was intended as an insult). (5) The *digits*[5] dataset comprises of 14,000 28x28 pixel images of handwritten digits for the numbers 3 and 5. The task is to correctly identify the digit in the image. (6) The *fashion*[6]

[1] https://archive.ics.uci.edu/ml/datasets/adult
[2] https://www.kaggle.com/sulianova/cardiovascular-disease-dataset
[3] https://archive.ics.uci.edu/ml/datasets/Bank+Marketing
[4] https://dataturks.com/projects/abhishek.narayanan/Dataset%20for%20Detection%20of%20Cyber-Trolls/
[5] http://yann.lecun.com/exdb/mnist/index.html
[6] https://github.com/zalandoresearch/fashion-mnist

dataset comprises of 14,000 28x28 pixel images of fashion products for the categories "sneakers" and "ankle boots". The task is to correctly identify the category of the product in the image.

We featurize the non-image datasets as follows: we standardize all numerical attributes, one-hot encode all categorical attributes, and hash word-level n-grams of textual attributes to a large sparse vector. We concatenate the encodings of the attributes to obtain the final feature vector. We apply this featurization via a scikit-learn pipeline to ensure that all preprocessing methods are 'fitted' on the training data only and applied to the unseen test data later. For experimental runs measuring the accuracy, we resample the data to have balanced classes to make the scores easier to interpret.

Models. For the majority of our experiments, we leverage four popular classification approaches and treat them as black box models:

- We train a logistic regression model[7] refered to as *lr* using five-fold cross-validation where we grid search over regularization type and learning rate.
- We train a feed-forward neural network [5] refered to as *dnn* comprised of two layers with ReLU activation and a softmax output. We use five-fold cross-validation and grid search over the size of the layers.
- We train gradient-boosted decision trees [4] refered to as *xgb*. We use five-fold cross-validation and grid search over the number and depth of the trees used.
- We train a convolutional neural network [5] refered to as *conv* for the image classification tasks. This network uses ReLU activations and drop-out regularization, and starts with two convolutional layers of size 32 and 64, applies max pooling, followed by a dense layer of width 128 and a softmax output.

Types of errors and dataset shifts. We experiment with six different types of errors that we introduce into the unseen serving data, some of which belong to the top problems commonly encountered in industrial ML pipelines [3].

Missing values in categorical attributes. We randomly choose 1 to n (with n being the maximum number of categorical columns) categorical columns and introduce missing values at random into these columns.

Outliers in numeric attributes. We randomly choose 1 to n (with n being the maximum number of numeric columns) numeric columns. For each column, we corrupt a fraction of its values by adding gaussian noise centered in the data point with a standard deviation randomly scaled from the interval of 2 to 5.

Swapped column values. We choose different pairs of categorical and numerical columns and randomly swap a given proportion of the contained values between the columns.

Scaling. We randomly scale a subset of the values by 10, 100 or 1000. This perturbation mimics cases where the scale of an attribute is accidentally changed in preprocessing code (e.g., because a developer accidentally changes the code to record durations in milliseconds instead of seconds).

Adversarial text. This perturbation is designated for the *tweets* dataset exclusively and simulates an adversarial attack. We assume that the attackers try to fool our classifier by changing the spelling of their trolling tweets to 'leetspeak', e.g., by converting the string "hello world" into "h3110 w041d".

Image noise:. We add noise sampled from a gaussian with zero mean and a variance randomly chosen between [-0.5,0.5] to a proportion of the input images.

Image rotation:. We rotate a proportion of the input images by randomly chosen angles.

Model-entropy based missing values.: We use an active learning based approach as a more challenging variant of missing values. We rank all samples by how difficult they are to classify and discard values from 'easy' samples for which the classifier was certain about its prediction. We use $1 - p_{\max}$ where $p_{\max}$ is the maximal probability assigned to a class for a given data point [26] as uncertainty measure.

If not described otherwise, we generate the synthetically corrupted data for our experiments as follows. We repeatedly choose a random selection of columns from the input dataframe, randomly sample the fraction of cells to corrupt and repeat this procedure 100 times per column and error combination. Thereby we typically generate a few thousand corrupted datasets to train a performance predictor.

We make the source code as well as serialized datasets and models of our experimental evaluation publicly available.[8]

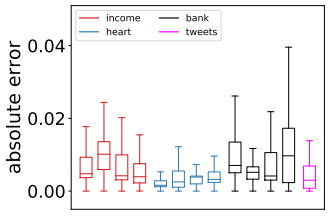
6.1 Prediction Score Estimation

In the following, we evaluate our performance prediction approach on known and unknown shifts and errors.

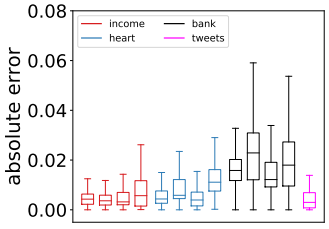
6.1.1 Prediction Score Estimation for Known Shifts. For every experimental run, we randomly partition a dataset into one partition designated as *source data* and another disjoint partition designated as *serving data*. Next, we decide for a particular type of data error and train one of our black box models as well as the performance predictor on the source data as described in Section 3. Afterwards, we apply error generators to the unseen serving data with randomly sampled probabilities and thereby create randomly corrupted versions of that data, injecting the expected type of errors.

[7] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

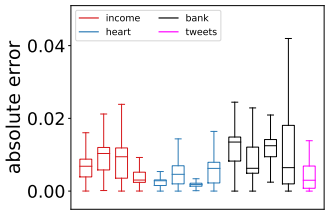
[8] https://github.com/schelterlabs/learning-to-validate-predictions



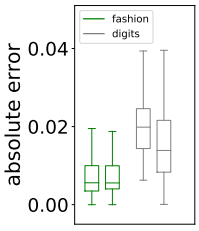
(a) Prediction error for the accuracy of a logistic regression model with missing values, outliers, swapped columns and scaled values (and adversarial examples in the *tweets* dataset).



(b) Prediction error for the accuracy of a neural network on serving datasets with missing values, outliers, swapped columns and scaled values (and adversarial examples in the *tweets* dataset).



(c) Prediction error for the accuracy of gradient-boosted decision trees on serving datasets with missing values, outliers, swapped columns and scaled values (and adversarial examples in the *tweets* dataset).



(d) Prediction error for the accuracy of a convolutional neural network on noisy and rotated images in the *fashion* and *digits* datasets.

Figure 2: Estimation of the prediction quality in the presence of known types (but unknown magnitudes) of errors in the serving data. We evaluate four different models on six different datasets and seven different errors and shifts.

We then have our performance predictor estimate the score on the unseen corrupted data, and compare this estimate to the actual score (which we can compute in this virtual setup because we have access to the true labels). We run experiments to predict both accuracy as well as the area under the ROC curve (*AUC*); We measure the prediction error using the absolute error, which is easy to interpret. We only plot results for the prediction of accuracy as the results for AUC do not significantly differ from these.

Figure 2 demonstrates the results of the corresponding experiments with respect to the prediction of the classifiers' accuracies. We plot the distribution of the absolute error of our performance predictor with respect to the accuracy of the black box model on the serving data. We run four different experiments per dataset and model, injecting missing values, outliers, column swaps and scaling errors for the *income*, *heart* and *bank* datasets. Additionally, we apply the adversarial attack on the *tweets* dataset. We repeat this experiment for the logistic regression model (Figure 2(a)), the neural network (Figure 2(b)) and gradient-boosted decision trees (Figure 2(c)). We also test the performance prediction for the convolutional neural network, where we inject noise and rotation errors into the *digits* and *fashion* datasets (Figure 2(d)).

In the majority of cases the median absolute error is not larger than 0.01 with low variance which means that our approach is able to reliably predict the prediction quality of the models on the unseen, corrupted serving data. We observe that the effect of scaled numerical input columns is harder to predict for all three models on the *bank* dataset, where we encounter a higher variance. Additionally, our approach also has problems with predicting the performance of the neural network in the presence of outliers on the *bank* dataset. The performance prediction for the convolutional neural network works better on the *digits* dataset with a very low median absolute error than on the *fashion* dataset where the median absolute error is closer to 0.02.

In general, our predictions are close to the true performance score on the unseen data in this scenario, where we know the type of error that might occur in our serving data. We thereby enable end users to distinguish catastrophic model failures (such as cases where the model tends to randomly guess) from cases where the performance is only slightly affected by the shifts in the data.

6.1.2 Prediction Score Estimation for Mixed and Unknown Shifts. In general, we cannot assume that we know the type of error that we might encounter in real world data. We emulate this scenario of unknown or partially observed errors by blending a fraction of samples affected by a given error into the serving data for evaluating our performance predictor. In this setup, our performance predictor will be trained on an error distribution different from the one present in the serving data. We choose a random numerical column and a random categorical column for each model and dataset combination. All error types are applied to the serving dataset. But in contrast to the previous experiments we only perturb a fraction of the source data used for training the performance predictor. If that fraction is 0, then no data point in the source data is affected by a given error type, and the performance predictor has not seen this error type in its training data. We include swapped columns, scaling errors, outliers, missing values and model entropy-based missing values as potential error types for this experiment.

Figure 3 shows the results for predicting the performance of linear and nonlinear models in this scenario. We observe that the prediction error for the linear model increases with

the amount of unknown errors[9], while the prediction quality of nonlinear models can be predicted with low variance even in these cases. This indicates that our approach is able to generalize from known errors.

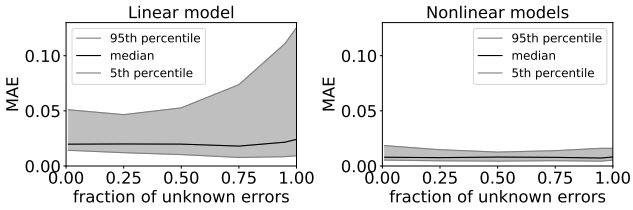


Figure 3: Performance prediction quality for linear and non-linear models under increasing magnitudes of unknown error types.

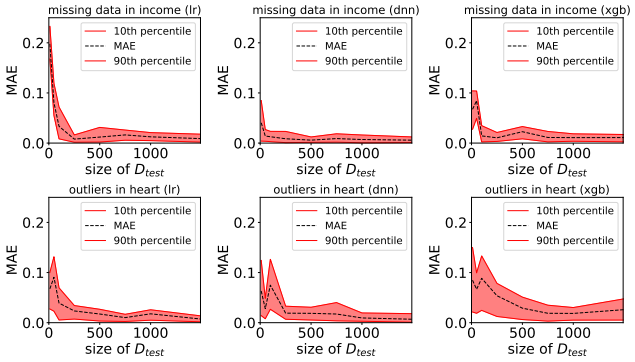


Figure 4: Sensitivity of the performance predictor to the sample size $|\mathcal{D}_{\text{test}}|$ for different errors and models on the *income* and *heart* datasets.

6.1.3 Sensitivity of the Predictor to the Sample Size. Next, we investigate the sensitivity of our approach to the size of the held-out data $|\mathcal{D}_{\text{test}}|$, (Algorithm 1), which we need to generate synthetically corrupted examples for the performance predictor. In an online serving setting, we would want the performance predictor to be quickly retrainable from small batches of observed data. We repeat the experiments with missing values in the *income* dataset and with outliers in the *heart* dataset from Section 6.1.1, and vary the size of the held-out data $\mathcal{D}_{\text{test}}$ between 10, 50, 100, 250, 500, 750, 1000, and 1500 data points. We train performance predictors for different models based on this sample data, and plot the the resulting MAE as well as the 10th and 90th percentile of the distribution of the absolute error in Figure 4. In all cases, we observe that the performance predictor achieves low prediction errors after having access to a few hundred examples.

6.2 Comparison against Task-Independent Dataset Shift Detection Methods

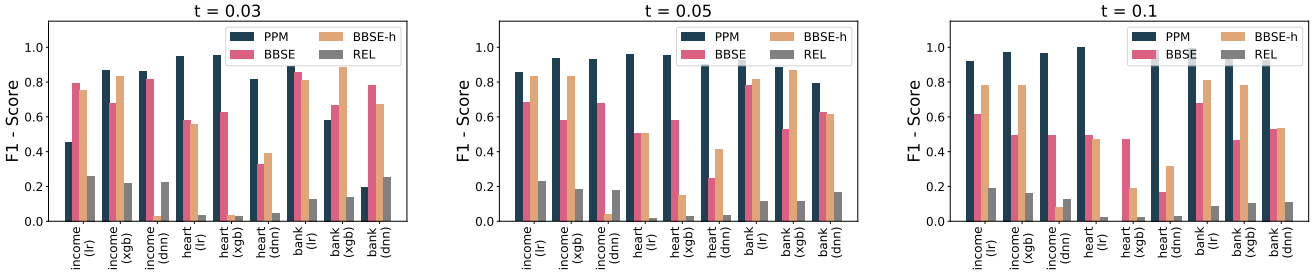
In the next set of experiments, we evaluate the quality of our performance validation approach, which aims to decide whether the prediction quality on a (potentially) corrupted serving dataset is within a given threshold of the prediction quality that the classifier achieved during evaluation on the test set at the training stage (Section 2). We compare our approach against the following three task-independent baselines.

- Relational shift detection (*REL*): This baseline approach applies shift detection techniques to the raw input data instead of the model outputs. We apply multiple univariate shift detection tests between the columns of the training and serving data (Kolmogorov-Smirnov tests for numeric columns, and χ^2-tests for categorical columns).
- Black Box-Shift Detection (*BBSE*) for assigned class probabilities from Lipton et al. [13], which evaluates a Kolmogorov-Smirnov test between the softmax outputs of the black box model on the test and serving data.
- Black Box-Shift Detection (*BBSEh*) [20] for predicted classes, which evaluates a χ^2-test between the counts of the predicted classes of the black box model on the test and serving data.

Note that *REL* directly compares statistics of the raw input data and serving data to detect whether the data distribution has changed, and is thereby independent of the applied black box classifier. Following [20], we compare the resulting p-value of each test to 0.05 to decide about the test outcome and apply Bonferroni correction to account for multiple tests in case of the *REL* baseline. For our approach, we train a performance validator (as described in Section 2), refered to as *PPM*, and report its decisions as well. We record how well each approach predicts whether the accuracy on the serving data is within the given threshold and report the resulting *F1* scores, which trade off both the precision and recall of the approaches.

6.2.1 Mixtures of Known Shifts and Errors. In the first set of experiments, we train the performance validator using randomly chosen mixtures of four different error types (missing values, outliers, swapped columns and scaling errors) and apply a randomly chosen mixture of the same error types (with different probabilities) onto the serving dataset. Next, we have our performance validator as well as the three baseline methods predict whether the accuracy drops less than a given threshold compared to the accuracy on the held-out test set. We run this experiment for the logistic regression,

[9]We attribute the bad performance of the linear model to numeric overflows in the `SGDClassifier` model that are caused by the scaling perturbations.



(a) Validation for a 3% threshold.

(b) Validation for a 5% threshold.

(c) Validation for a 10% threshold.

Figure 5: F1-Scores for performance validation for mixtures of unknown shifts and errors.

neural network and gradient-boosted tree models, on the *income*, *heart* and *bank* dataset. We find that our performance validator (PPM) outperforms the baselines in the vast majority of cases, often by a wide margin with an F1-score between 0.8 and 0.9. BBSE only beats our approach in three cases: for logistic regression on the *income* dataset with a 3% threshold, for the neural network on the *bank* dataset with a 3% threshold and for the neural network on the *bank* dataset with a 5% threshold. There is no clear second best approach as both BBSE and BBSEh outperform each other in many cases (and perform badly in other cases), while REL works poorly in the majority of cases. In general, we observe that it becomes easier for our approach to correctly predict the performance with a larger threshold, (as the F1-scores grow for higher thresholds). This indicates that the performance validator is able to reliably identify catastrophic performance drops (e.g., with a 10% drop in accuracy).

6.2.2 Validation under Unknown Shifts and Errors. Next, we evaluate how well the performance validation works under unseen shifts and errors. We again use the setup from the previous experiment, where we train on randomly chosen mixtures of four different error types (missing values, outliers, swapped columns and scaling errors), but we evaluate on three newly defined error types which are unknown to our performance validator:

- *Typos in categorical values:* We introduce typos into a random proportion of the values of a categorical attribute.
- *"Smearing" of numerical attributes:* We change a random proportion of the values of a numeric attribute by a randomly chosen amount between -10% to 10%.
- *Flipped sign in numerical attributes:* We multiply a random proportion of the values of a numeric attribute by -1 to change their sign.

We apply a randomly chosen mixture of the unseen error types (with different probabilities) onto the serving dataset. Next, we again have our performance validator as well as the three baseline methods predict whether the accuracy drops less than a given threshold compared to the accuracy on the held-out test set. We run this experiment for the logistic regression, neural network and gradient-boosted tree models, on the *income*, *heart* and *bank* dataset. We evaluate all approaches for thresholds of 3%, 5% and 10%.

We plot the results in Figure 5. We observe that PPM predicts the performance very reliably in the majority of cases, and again only fails to beat the baselines in three settings: logistic regression on the *income* dataset with a 3% threshold, the *neural network* on the bank dataset with a 3% threshold, and the gradient-boosted decision trees on the *heart* dataset with a 10% threshold. In many cases, the F1 scores achieved by our approach in this setting are even better than in the setting with the known errors, which we attribute to the fact the unknown errors under test here have less drastic effects than some of the known errors, such as the scaling errors, which we have been shown to be difficult to handle in Section 6.1.1

Note that while the error types applied here are unknown to our performance predictor, it is still be able to learn about their effects from the known error types it has seen. We attribute this to the fact that their impact on the predictions of the black box model is similar to the impact of the known errors. E.g., the impact of a typo in a categorical column is identical to the impact of a missing value in that column, as the feature map will produce a zero vector when trying to one-hot encode the resulting value. The same is true for the "smeared" numerical values: their effect might be similar to gaussian noise added to the values for our outlier errors. This experiment confirms our findings from Section 6.1.2 that our approach is able to generalize from known errors to unobserved errors in many cases.

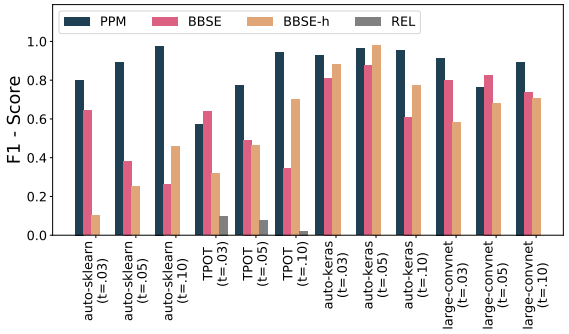


Figure 6: Performance validation for black box models trained by various AutoML methods in the presence of mixtures of known shifts and errors.

6.3 Application to AutoML Models

6.3.1 AutoML Libraries. We experiment with a series of models trained by three different automatic machine learning (AutoML) libraries, which automatically decide on model internals such as feature maps or ensembling techniques.

We have *auto-sklearn* [7] and *TPOT* [16] learn models for the *income* dataset. Additionally, we have *auto-keras* [11] conduct neural architecture search for a convolutional neural network to apply to the *digits* dataset. We again train our performance validator using randomly chosen mixtures of four different error types (missing values, outliers, swapped columns and scaling errors for the income dataset; rotations and noise for the digits dataset) and apply a randomly chosen mixture of the same error types (with different probabilities) to the serving dataset. Next, we have our performance validator as well as the three baseline methods predict whether the accuracy drops less than a given threshold compared to the accuracy on the held-out test set. We evaluate all approaches for thresholds of 3%, 5% and 10%.

The results (plotted in Figure 6) are in line with our previous findings from Section 6.2.1: Our approach (PPM) outperforms black box shift estimation (BBSE, BBSEh) as well as multiple univariate tests on the raw input data (REL) in the majority of cases, often by a wide margin. The BBSE and BBSEh baselines beat our aproach only in two cases: for TPOT for 3% threshold and for auto-keras with a 5% threshold, yet only with a slight difference in F1 scores. REL again performs very poorly (and was not applicable to the image dataset used for *auto-keras*). The results confirm that our approach is able to "tailor" the performance validator to various black box models.

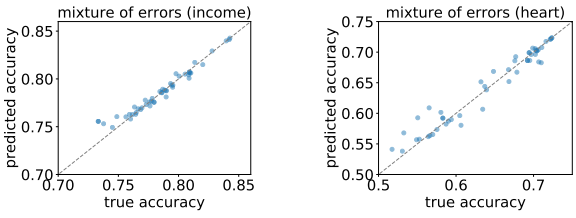


Figure 7: Prediction quality for black box models trained and hosted via *Google AutoML Tables* on a mixture of errors in the *income* and *heart* datasets.

6.3.2 Google AutoML Tables in the Cloud. In this experiment, we evaluate the applicability of our approach to black box models hosted in the cloud. We train a classifier for the *income* dataset, via the *Google AutoML Tables* cloud service at https://cloud.google.com/automl-tables. Note that the resulting model is trained and hosted in the cloud, and both the applied learning algorithm as well as the model's feature map are unknown to us.

Next, we generate synthetically corrupted, held-out test data, by applying a mixture of perturbations such as missing values, swapped columns, outliers and scaling errors. We retrieve predictions for the corrupted test data and train our performance predictor. Finally, we generate corrupted serving data (again with a random mixture of the previous perturbations), have our performance predictor predict the accuracy of the black box model on this data, and retrieve the predictions from the cloud model to compute the true accuracies. The results in Figure 7 are in line with our findings from Section 6.1.1: our approach predicts the resulting accuracy scores well with an MAE of only 0.0038. We repeat this experiment for the *heart* dataset, and achieve similar results with an MAE of 0.0101.

7 CONCLUSION AND FUTURE WORK

We proposed a simple approach to automate the validation of black box classifiers by estimating the model's predictive performance on unseen, unlabeled serving data. The approach is complementary to existing techniques, such as schema validation for model input data [3].

In future work, we intend to investigate the effects of more error types, and aim to empirically study whether there is a set of errors for training which generalizes to the majority of real world cases. We would also be interested in better understanding how the distribution of the model outputs encodes the predictive performance. Additionally, we would like to quantify the effects of our investigated error types for selected models (e.g., neural networks) more formally.

REFERENCES

[1] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. *KDD*, 1387–1395.

[2] Steffen Bickel, Michael Brückner, and Tobias Scheffer. 2009. Discriminative learning under covariate shift. *JMLR* 10, 2137–2155.

[3] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Whang, and Martin Zinkevich. 2019. Data Validation for Machine Learning. *SysML*.

[4] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. *KDD*, 785–794.

[5] François Chollet et al. 2015. Keras `tensorflow.org/guide/keras`.

[6] Yeounoh Chung, Tim Kraska, Steven Euijong Whang, and Neoklis Polyzotis. 2018. Slice finder: Automated data slicing for model interpretability. *SysML*.

[7] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. *NeurIPS*, 2962–2970.

[8] Joseph M Hellerstein. 2008. Quantitative data cleaning for large databases. *United Nations Economic Commission for Europe (UNECE)*.

[9] Jiayuan Huang, Arthur Gretton, Karsten M Borgwardt, Bernhard Schölkopf, and Alex J Smola. 2007. Correcting sample selection bias by unlabeled data. *NeurIPS*, 601–608.

[10] Nick Hynes, D Sculley, and Michael Terry. 2017. The Data Linter: Lightweight, Automated Sanity Checking for ML Data Sets. *ML Systems Workshop @ NeurIPS*.

[11] Haifeng Jin, Qingquan Song, and Xia Hu. 2018. Auto-Keras: Efficient Neural Architecture Search with Network Morphism. arXiv:cs.LG/cs.LG/1806.10282

[12] Arun Kumar, Robert McCann, Jeffrey Naughton, and Jignesh M Patel. 2016. Model selection management systems: The next frontier of advanced analytics. *SIGMOD Record* 44, 4, 17–22.

[13] Zachary C Lipton, Yu-Xiang Wang, and Alex Smola. 2018. Detecting and Correcting for Label Shift with Black Box Predictors. *ICML*.

[14] Wes McKinney et al. 2010. Data structures for statistical computing in python. *Python in Science* 445, 51–56.

[15] Augustus Odena and Ian Goodfellow. 2018. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. *arXiv preprint arXiv:1807.10875*.

[16] Randal S. Olson, Ryan J. Urbanowicz, Peter C. Andrews, Nicole A. Lavender, La Creis Kidd, and Jason H. Moore. 2016. Automating Biomedical Data Science Through Tree-Based Pipeline Optimization. *EvoApplications*.

[17] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *JMLR* 12, Oct, 2825–2830.

[18] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. *SOSP*, 1–18.

[19] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2018. Data Lifecycle Challenges in Production Machine Learning: A Survey. *SIGMOD Record* 47, 2, 17.

[20] Stephan Rabanser, Stephan Günnemann, and Zachary C Lipton. 2019. Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift. *NeurIPS*.

[21] Sergey Redyuk, Sebastian Schelter, Tammo Rukat, Volker Markl, and Felix Biessmann. 2019. Learning to Validate the Predictions of Black Box Machine Learning Models on Unseen Data. *Human-in-the-Loop Data Analytics workshop at SIGMOD*.

[22] Sebastian Schelter, Felix Biessmann, Tim Januschowski, David Salinas, Stephan Seufert, and Gyuri Szarvas. 2018. On Challenges in Machine Learning Model Management. *IEEE Data Engineering Bulletin* 41.

[23] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann, and Andreas Grafberger. 2018. Automating large-scale data quality verification. *PVLDB* 11, 12, 1781–1794.

[24] S. Schelter, J. Soto, V. Markl, D. Burdick, B. Reinwald, and A. Evfimievski. 2015. Efficient sample generation for scalable meta learning. *ICDE*, 1191–1202.

[25] D Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. *NeurIPS*, 2503–2511.

[26] Burr Settles. 2010. *Active Learning Literature Survey*. Technical Report 1648. University ofWisconsin–Madison.

[27] Masashi Sugiyama and Motoaki Kawanabe. 2012. *Machine Learning in Non-Stationary Environments - Introduction to Covariate Shift Adaptation*. MIT Press.

[28] Masashi Sugiyama, Neil D Lawrence, Anton Schwaighofer, et al. 2017. *Dataset shift in machine learning*. MIT Press.

[29] Paul von Bünau, Frank C. Meinecke, Franz C. Király, and Klaus-Robert Müller. 2009. Finding Stationary Subspaces in Multivariate Time Series. *Phys. Rev. Lett.* 103. Issue 21.

[30] Pei Wang and Yeye He. 2019. Uni-Detect: A Unified Approach to Automated Error Detection in Tables. *SIGMOD*, 811–828.

[31] Kun Zhang, Bernhard Schölkopf, Krikamol Muandet, and Zhikun Wang. 2013. Domain Adaptation under Target and Conditional Shift. *ICML* 28, 819–827.