Chapter 2. Collecting, Labeling, and Validating Data

In production environments, you discover some interesting things about the importance of data. We asked ML practitioners at Uber and Gojek, two businesses where data and ML are mission critical, about it. Here's what they had to say:

> *Data is the hardest part of ML and the most important piece to get right...Broken data is the most common cause of problems in production ML systems.*
>
> —ML practitioner at Uber

> *No other activity in the machine learning lifecycle has a higher return on investment than improving the data a model has access to.*
>
> —ML practitioner at Gojek

The truth is that if you ask any production ML team member about the importance of data, you'll get a similar answer. This is why we're talking about data: it's incredibly important to success, and the issues for data in production environments are very different from those in the academic or research environment that you might be familiar with.

OK, now that we’ve gotten that out of the way, let’s dive in!

Important Considerations in Data Collection

In programming language design, a *first-class citizen* in a given programming language is an entity that supports all the operations generally available to other entities. In ML, data is a first-class citizen. Finding data with predictive content might sound easy, but in reality it can be incredibly difficult.

When collecting data, it’s important to ensure that the data represents the application you are trying to build and the problem you are trying to solve. By that we mean you need to ensure that the data has feature space coverage that is close to that of the prediction requests you will receive.

Another key part of data collection is sourcing, storing, and monitoring your data responsibly. This means that when you’re collecting data, it is important to identify potential issues with your dataset. For example, the data may have come from different measurements of different types (e.g., the dataset may mix some measurements that come from two different types of thermometers that produce different measurements). In addition, simple things like the difference between an integer and a float, or how a missing value is encoded, can cause problems. As another example, if you have a dataset that measures elevation, does an entry of 0 feet mean no elevation (sea level), or that no elevation data was received for that record? If the output of other ML models is the input dataset for your model, you also need to be aware of the potential for errors to propagate over time. And you want to make sure you're looking for potential problems early in the process by monitoring data sources for system issues and outages.

When collecting data, you will also need to understand data effectiveness by dissecting which features have predictive value. Feature engineering helps maximize the predictive signal of your data, and feature selection helps measure the predictive signal.

Responsible Data Collection

In this section, we will discuss how to responsibly source data. This involves ensuring data security and user privacy, checking for and ensuring fairness, and designing labeling systems that mitigate bias.

ML system data may come from different sources, including synthetic datasets you build, open source datasets, web scraping, and live data collection. When collecting data, data security and data privacy are important. *Data security* refers to the policies, methods, and means to secure personal data. *Data privacy* is about proper usage, collection, retention, deletion, and storage of data.

Data management is not only about the ML product. Users should also have control over what data is being collected. In addition, it is important to establish mechanisms to prevent systems from revealing user data inadvertently. When thinking about user privacy, the key is to protect personal identifiable information (PII). Aggregating, anonymizing, redacting, and giving users control over what data they share can help prevent issues with PII. How you handle data privacy and data security depends on the nature of the data, the operating conditions, and regulations currently in place (an example is the General Data Protection Regulation or GDPR, a European Union regulation on information privacy).

In addition to security and privacy, you must consider fairness. ML systems need to strike a delicate balance in being fair, accurate, transparent, and explainable. However, such systems can fail users in the following ways:

Representational harm

> When a system amplifies or reflects a negative stereotype about particular groups

Opportunity denial

> When a system makes predictions that have negative real-life consequences, which could result in lasting impacts

Disproportionate product failure

When you have skewed outputs that happen more frequently for a particular group of users

Harm by disadvantage

When a system infers disadvantageous associations between different demographic characteristics and the user behaviors around them

When considering fairness, you need to check that your model does not consistently predict different experiences for some groups in a problematic way, by ensuring group fairness (demographic parity and equalized odds) and equal accuracy.

One aspect of this is looking at potential bias in human-labeled data. For supervised learning, you need accurate labels to train your model on and to serve predictions. These labels usually come from two sources: automated systems and human raters. *Human raters* are people who look at the data and assign a label to it. There are various types of human raters, including generalists, trained subject matter experts, and users. Humans are able to label data in different ways than automated systems can. In addition, the more complicated the data is, the more you may require a human expert to look at that data.

When considering fairness with respect to human-labeled data, there are many things to think about. For instance, you will want to ensure rater pool diversity, and you will want to account for rater context and incentives. In addition, you’ll want to evaluate rater tools and consider cost, as you need a sufficiently large dataset. You will also want to consider data freshness requirements.

Labeling Data: Data Changes and Drift in Production ML

When thinking about data, you must also consider the fact that data changes often. There are numerous potential causes of data changes or problems, which can be categorized as those that cause gradual changes or those that cause sudden changes.

Gradual changes might reflect changes in the data and/or changes in the world that affect the data. Gradual data changes include those due to trends or seasonality, changes in the distribution of features, or changes in the relative importance of features. Changes in the world that affect the data include changes in styles, scope and process changes, changes in competitors, and expansion of a business into different markets or areas.

Sudden changes can involve both data collection problems and system problems. Examples of data collection problems that cause sudden changes in data include moved, disabled, or malfunctioning sensors or cameras, or problems in logging. Examples of system problems that can cause sudden changes in data include bad software updates, loss of network connectivity, or a system delay or failure.

Thinking about data changes raises the issues of data drift and concept drift. With *data drift*, the distribution of the data input to your model changes. Thus, the data distribution on which the model was trained is different from the current input data to the model, which can cause model performance to decay in time. As an example of data drift, if you have a model that predicts customer clothing preferences that was trained with data collected mainly from teenagers, the accuracy of that model would be expected to degrade if data from older adults is later fed to the model.

With *concept drift*, the relationship between model inputs and outputs changes over time, which can also lead to poorer model performance. For example, a model that predicts consumer clothing preferences might degrade over time as new trends, seasonality, and other previously unseen factors change the customer preferences themselves.

To handle potential data change, you must monitor your data and model performance continuously, and respond to model performance decays over time. When ground truth changes slowly (i.e., over months or years), handling data change tends to be relatively easy. Model retraining can be driven by model improvements, better data, or changes in software or systems. And in this case, you can use curated datasets built using crowd-based labeling.

When ground truth changes more quickly (i.e., over weeks), handling data change tends to become more difficult. In these cases, model retraining can be driven by the factors noted previously, but also by declining model performance. Here, datasets tend to be labeled using direct feedback or crowd-based labeling.

When ground truth changes even more quickly (i.e., over days, hours, or minutes), things become even more difficult. Here, model retraining can be driven by declining model performance, the desire to improve models, better training data availability, or software system changes. Labeling in this scenario could be through direct feedback (discussed next), or through weak supervision for applying labels quickly.

Labeling Data: Direct Labeling and Human Labeling

Training datasets need to be created using the data available to the organization, and models often need to be retrained with new data at some frequency. To create a current training dataset, examples must be labeled. As a result, labeling becomes an ongoing and mission-critical process for organizations doing production ML.

We will start our discussion of labeling data by taking a look at direct labeling and human labeling. *Direct labeling* involves gleaning information from your system—for example, by tracking click-through rates. *Human labeling* involves having a person label examples with ground truth values —for example, by having a cardiologist label MRI scans as a subject matter expert rater. There are also other methods, including semi-supervised labeling, active learning, and weak supervision, which we will discuss in later chapters that address advanced labeling methods.

Direct labeling has several advantages: it allows for a training dataset to be continuously created, as labels can be added from logs or other system-collected information as data arrives; it allows labels to evolve and adapt quickly as the world changes; and it can provide strong label signals. However, there are situations in which direct labeling is not available or has disadvantages. For example, for some types of ML problems, labels cannot be gleaned from your system. In addition, direct labeling can require custom designs to fit your labeling processes with your systems.

In cases where direct labeling is useful, there are open source tools that you can use for log analysis. Two such tools are Logstash and Fluentd. Logstash is a data processing pipeline for collecting, transforming, and storing logs from different sources. Collected logs can then be sent to one of several types of outputs. Fluentd is a data collector that can collect, parse, transform, and analyze data. Processed data can then be stored or connected with various platforms. In addition, Google Cloud provides log analytics services for storing, searching, analyzing, monitoring, and alerting on logging data and events from Google Cloud and Amazon Web Services

(AWS). Other systems, such as AWS Elasticsearch and Azure Monitor, are also available for log processing and can be used in direct labeling.

With human labeling, raters examine data and manually assign labels. Typically, raters are recruited and given instructions to guide their assignment of ground truth values. Unlabeled data is collected and divided among the raters, often with the same data being assigned to more than one rater to improve quality. The labels are collected, and conflicting labels are resolved.

Human labeling allows more labels to be annotated than might be possible through other means. However, there are disadvantages to this approach. Depending on the dataset, it might be difficult for raters to assign the correct label, resulting in a low-quality dataset. Quality might also suffer due to rater inexperience and other factors. Human labeling can also be an expensive and slow process, and can result in a smaller training dataset than could be created through other methods. This is particularly the case for domains that require significant specialization or expertise to be able to label the data, such as medical imaging. In addition, human labeling is subject to the fairness considerations discussed earlier in this chapter.

Validating Data: Detecting Data Issues

As discussed, there are many ways in which your data can change or in which the systems that impact your data can cause unanticipated issues. Especially in light of the importance of data to ML systems, detecting such issues is essential. In this section, we will discuss common issues to look for in your data, and the concepts involved in detecting those issues. In the next section, we'll explore a specific tool for detecting such data issues.

As we noted earlier, issues can arise due to differences in datasets. One such issue or group of issues is drift, which, as we mentioned previously, involves changes in data over time. With *data drift*, the statistical properties of the input features change due to seasonality, events, or other changes in the world. With *concept drift*, the statistical properties of the labels change over time, which can invalidate the mapping found during training.

Skew involves changes between datasets, often between training datasets and serving datasets. *Schema skew* occurs when the training and serving datasets do not conform to the same schema. *Distribution skew* occurs when the distribution of values in the training and serving datasets differs.

Validating Data: TensorFlow Data

Validation

Now that you understand the basics of data issues and detection workflows, let’s take a look at TensorFlow Data Validation (TFDV), a library that allows you to analyze and validate data using Python and Apache Beam. Google uses TFDV to analyze and validate petabytes of data every day across hundreds or thousands of different applications that are in production. The library helps users maintain the health of their ML pipelines by helping them understand their data and detect data issues like those discussed in this chapter.

TFDV allows users to do the following:

- Generate summary statistics over their data
- Visualize those statistics, including visually comparing two datasets
- Infer a schema to express the expectations for their data
- Check the data for anomalies using the schema
- Detect drift and training–serving skew

Data validation in TFDV starts with generating summary statistics for a dataset. These statistics can include feature presence, values, and valency, among other things. TFDV leverages Apache Beam’s data processing capabilities to compute these statistics over large datasets.

Once TFDV has computed these summary statistics, it can automatically create a schema that describes the data by defining various constraints including feature presence, value count, type, and domain. Although it is useful to have an automatically inferred schema as a starting point, the expectation is that users will tweak or curate the generated schema to better reflect their expectations about their data.

With a refined schema, a user can then run anomaly detection using TFDV. TFDV can do several types of anomaly detection, including comparison of a single set of summary statistics to a schema to ensure that the data from which the statistics were generated conforms to the user's expectations. TFDV can also compare the data distributions between two datasets—again using TFDV-generated summary statistics—to help identify potential drift or training–serving skew (discussed further in the next section).

The results of TFDV's anomaly detection process can help users further refine the schema or identify potentially problematic inconsistencies in their data. The schema can then be maintained over time and used to validate new data as it arrives.

Skew Detection with TFDV

Let's take a closer look at TFDV's ability to detect anomalies such as data drift and training–serving skew between datasets. For our discussion, *drift* refers to differences across iterations of training data and *skew* refers to differences between training and serving data.

You can use TFDV to detect three types of skew: schema skew, feature skew, and distribution skew, as shown in Figure 2-1.

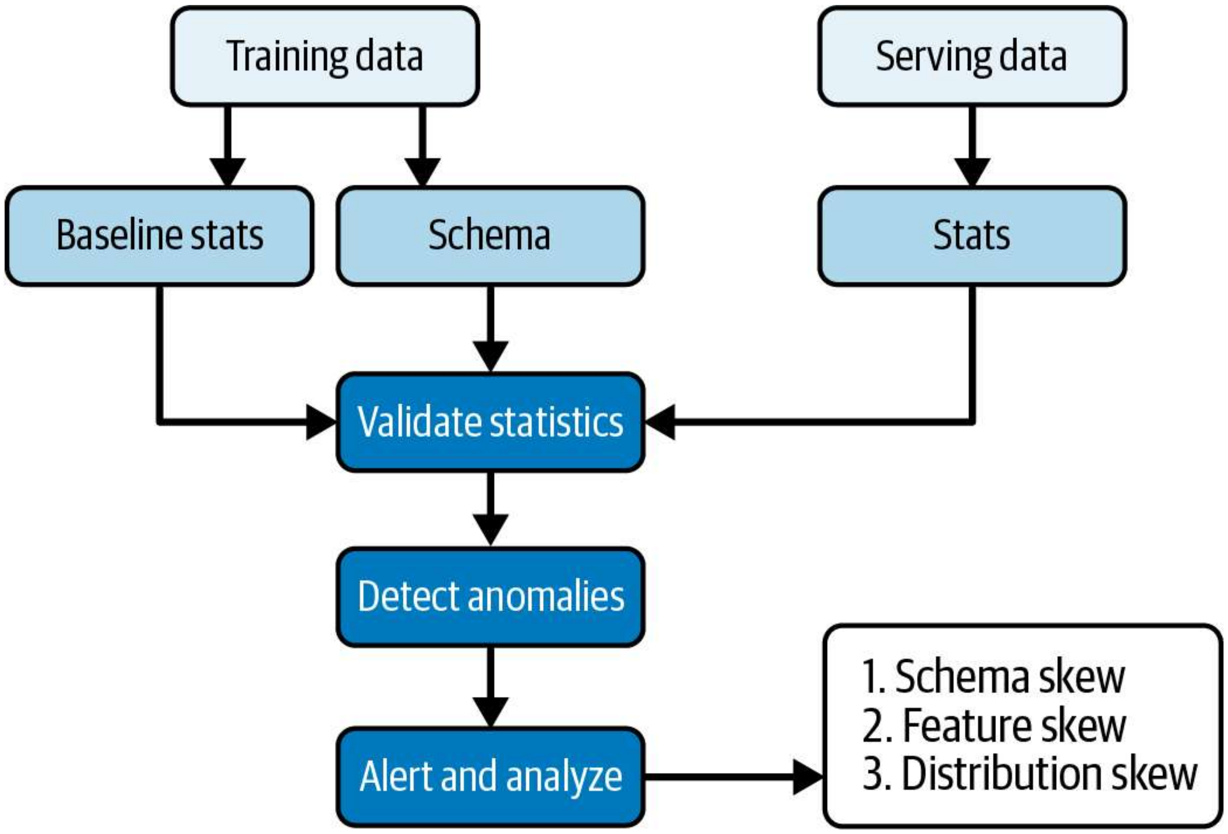


Figure 2-1. Skew detection with TFDV

Types of Skew

Schema skew occurs when the training data and serving data do not conform to the same schema; for example, if Feature A is a float in the training data but an integer in the serving data. Schema skew is detected similarly to single-dataset anomaly detection, which compares the dataset to a specified schema.

Feature skew occurs where feature values that are supposed to be the same in both training data and serving data differ. To identify feature skew, TFDV joins the training and serving examples on one or more specified identifier features, and then compares the feature values to identify the resulting pairs. If they differ, TFDV reports the difference as feature skew. Because feature skew is computed using examples and not summary statistics, it is computed separately from the other validation steps.

Distribution skew occurs when there is a shift in the distribution of feature values across two datasets. TFDV uses L-infinity distance (for categorical features only) and Jensen–Shannon divergence (for numeric and categorical features) to identify and measure such shifts. If the measure exceeds a user-specified threshold, TFDV will raise a distribution skew anomaly noting the difference.

Various factors can cause the distribution of serving and training datasets to differ significantly, including faulty sampling during training, use of different data sources for training and serving, and trend, seasonality, or other changes over time. Once TFDV helps identify potential skew, you can investigate the shift to determine whether it’s a problem that needs to be remedied.

Example: Spotting Imbalanced Datasets with TensorFlow Data Validation

Let’s say you want to visually and programmatically detect whether your dataset is imbalanced. We consider datasets to be *imbalanced* if the sample quantities per label are vastly different (e.g., you have 100 samples for one category and 1,000 samples for another category). Real-world datasets will almost always be imbalanced for various reasons—for example, because the costs of acquiring samples for a certain category might be too high—but datasets that are too imbalanced hinder the model training process to generalize the overall problem.

TFDV offers simple ways to generate statistics of your datasets and check for imbalance. In this section, we’ll take you through the steps of using TFDV to spot imbalanced datasets.

Let’s start by installing the TFDV library:

```
$ pip install tensorflow-data-validation
```

If you have TFX installed, TFDV will automatically be installed as one of the dependencies.

With a few lines of code, we can analyze the data. First, let’s generate the data statistics:

```
import tensorflow_data_validation as tfdv
stats = tfdv.generate_statistics_from_csv(
    data_location='your_data.csv',
    delimiter=',')
```

TFDV provides functions to load the data from a variety of formats, such as Pandas data frames (`generate_statistics_from_dataframe`) or TensorFlow’s TFRecords (`generate_statistics_from_tfrecord`):

```
stats = tfdv.generate_statistics_from_tfrecord(
    data_location='your_data.tfrecord')
```

It even allows you to define your own data connectors. For more information, refer to the TFDV documentation.

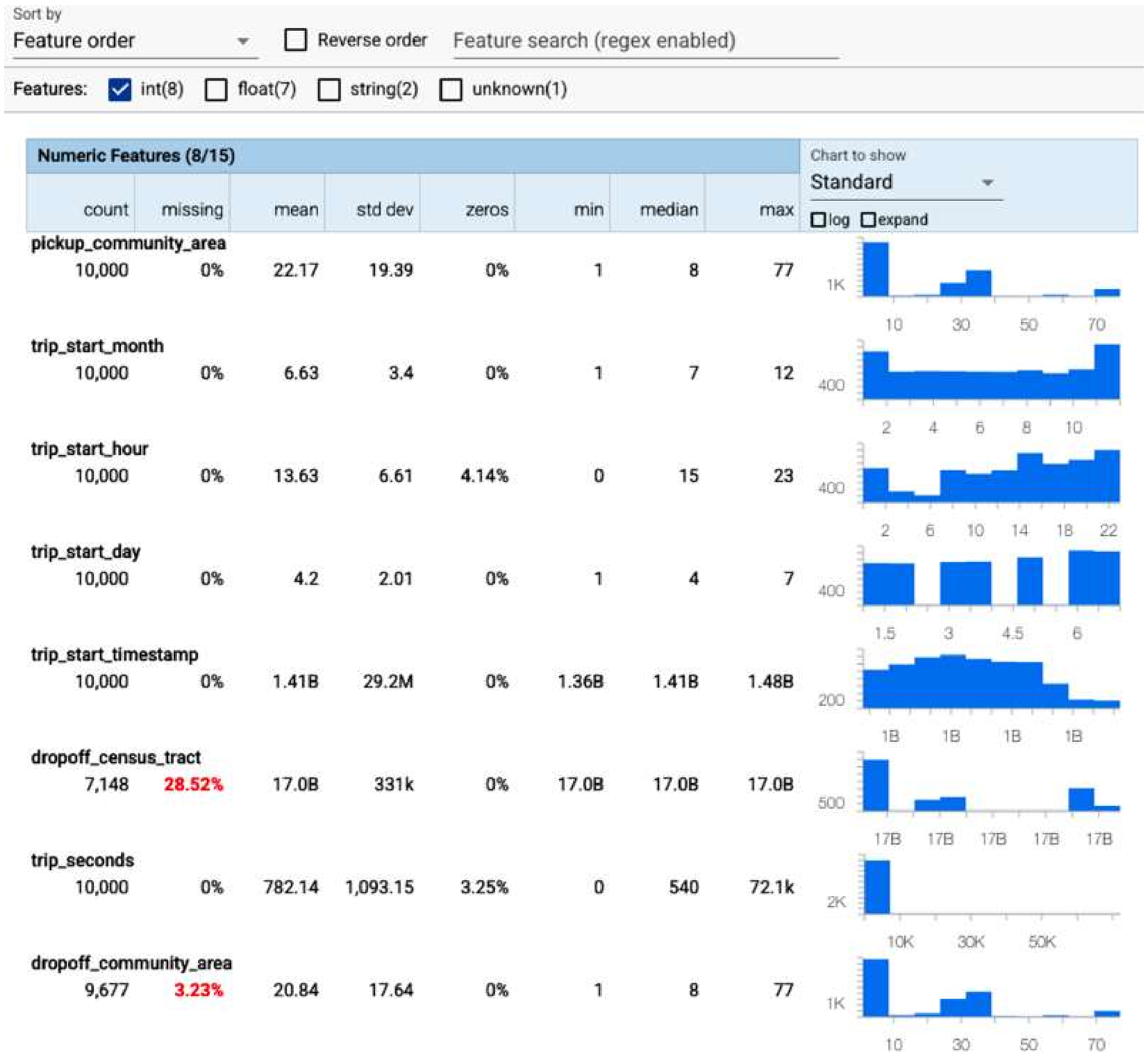
If you want to programmatically check the label distribution, you can read the generated statistics. In our example, we loaded a spam detection dataset with data samples marked as `spam` or `ham`. As in every real-world example, the dataset contains more nonspam examples than spam examples. But how many? Let’s check:

```
print(stats.datasets[0].features[0].string_stats
buckets {
  label: "ham"
  sample_count: 4827.0
}
buckets {
  low_rank: 1
  high_rank: 1
  label: "spam"
  sample_count: 747.0
}
```

The output shows that our dataset contains 747 spam examples and 4,827 ham (benign) examples.

Furthermore, you can use TFDV to quickly generate a visualization of statistics, as shown in Figure 2-2 for another dataset, with the following function:

```
tfdv.visualize_statistics(stats)
```



Sort by
Feature order
Reverse order
Feature search (regex enabled)
Features: int(8) float(7) string(2) unknown(1)
Chart to show
Standard
log expand

Numeric Features (8/15)

	count	missing	mean	std dev	zeros	min	median	max
pickup_community_area	10,000	0%	22.17	19.39	0%	1	8	77
trip_start_month	10,000	0%	6.63	3.4	0%	1	7	12
trip_start_hour	10,000	0%	13.63	6.61	4.14%	0	15	23
trip_start_day	10,000	0%	4.2	2.01	0%	1	4	7
trip_start_timestamp	10,000	0%	1.41B	29.2M	0%	1.36B	1.41B	1.48B
dropoff_census_tract	7,148	28.52%	17.0B	331k	0%	17.0B	17.0B	17.0B
trip_seconds	10,000	0%	782.14	1,093.15	3.25%	0	540	72.1k
dropoff_community_area	9,677	3.23%	20.84	17.64	0%	1	8	77

Figure 2-2. Visualizing a dataset

ALTERNATIVES TO TENSORFLOW DATA VALIDATION

While the simplicity of TFDV is amazing, data scientists might prefer a different analysis tool, especially if they don’t use TensorFlow as their ML framework of choice. A number of open source data analysis tools have been released alongside TFDV. Following are some alternatives:

Great Expectations

Started as an open source project, but is now a commercial cloud solution. It allows you to connect with a number of data sources out of the box, including in-memory databases.

Evidently

Allows users to analyze and visualize datasets with a focus on dataset monitoring. It supports drift detection for unstructured text data.

Conclusion

In this chapter, we discussed the many things to consider when collecting and labeling the data used to train ML models. Given the importance of data to the health of your ML system, the potential issues with collecting and labeling data, and the potential for data changes for various and sometimes difficult-to-foresee reasons, it is imperative to develop effective systems for managing and validating your data.