Chapter 5. Advanced Labeling, Augmentation, and Data Preprocessing

The topics in this chapter are especially important to shaping your data to get the most value from it for your model, especially in a supervised learning setting. Labeling in particular can easily be one of the most expensive and time-consuming activities in the creation, maintenance, and evolution of an ML application. A good understanding of the options available will help you make the most of your resources and budget.

To that end, in this chapter we will discuss data augmentation, a class of methods in which you add more data to your training dataset in order to improve training, usually to improve generalization in particular. Data augmentation is almost always based on manipulating your current data to create new, but still valid, variations of your examples.

We will also discuss data preprocessing, but in this chapter we'll focus on domain-specific preprocessing. Different domains, such as time series, text, and images, have specialized forms of feature engineering. We discussed one of these, tokenizing text, in "Consider Instance-Level Versus Full-Pass Transformations". In this chapter, we'll review common methods for working with time series data.

But first, let’s address an important question: How can we assign labels in ways other than going through each example manually? In other words, can we automate the process even at the expense of introducing inaccuracies in the labeling process? The answer is yes, and the way we do it is through advanced labeling.

Advanced Labeling

Why is advanced labeling important? Well, the use of ML is growing worldwide, and ML requires training data. If you’re doing supervised learning, that training data needs labels, and supervised learning represents the vast majority of ML in production today.

But manually labeling data is often expensive and difficult, and unlabeled data is typically pretty cheap and easy to get and contains a lot of information that can help improve our model. So, advanced labeling techniques help us reduce the cost of labeling data while leveraging the information in large amounts of unlabeled data.

In this section, we’ll start with a discussion of how semi-supervised labeling works and how you can use it to improve your model’s performance by expanding your labeled dataset in directions that provide the most predictive information. We’ll follow this with a discussion of active learning, which uses intelligent sampling to assign to unlabeled data labels based on the existing data. Then, we’ll introduce weak supervision,

which is an advanced technique for programmatically labeling data, typically by using heuristics that are designed by subject matter experts.

Semi-Supervised Labeling

With semi-supervised labeling, you start with a relatively small dataset that's been labeled by humans. You then combine that labeled data with a large amount of unlabeled data, inferring the labels by looking at how the different human-labeled classes are clustered within the feature space. Then, you train your model using the combination of the two datasets. This method is based on the assumption that different label classes will cluster together within the feature space, which is typically—but not always—a good assumption.

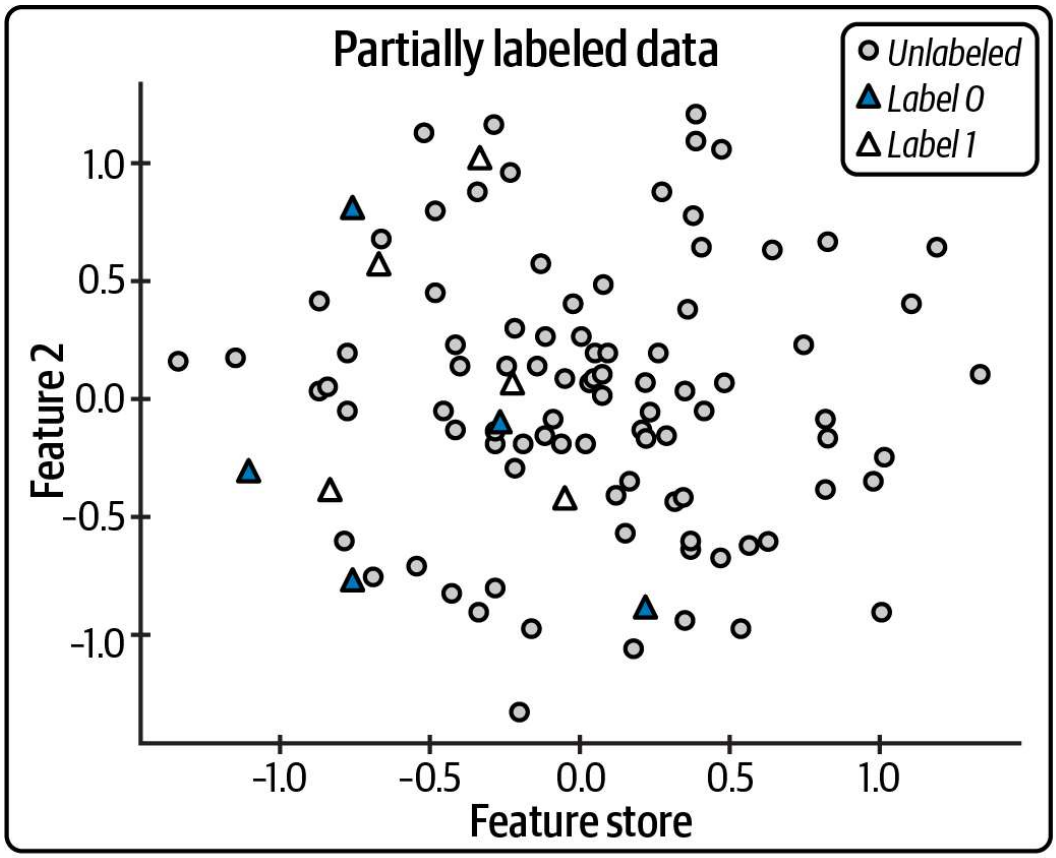
Using semi-supervised labeling is advantageous for two main reasons. First, combining labeled and unlabeled data can increase feature space coverage, which, as described in "Feature Selection", can improve the accuracy of ML models. Second, getting unlabeled data is often very inexpensive because it doesn't require people to assign labels. Often, unlabeled data is easily available in large quantities.

By the way, don't confuse semi-supervised labeling with semi-supervised training, which is very different. We'll discuss semi-supervised training in a later chapter.

Label propagation

Label propagation is an algorithm for assigning labels to previously unlabeled examples. This makes it a semi-supervised algorithm, where a subset of data points have labels. The algorithm propagates the labels to data points without labels based on the similarity or community structure of the labeled data points and the unlabeled data points. This similarity or structure is used to assign labels to the unlabeled data.

In Figure 5-1, you can see some labeled data (the triangles) and a lot of unlabeled data (the circles). With label propagation, you assign labels to the unlabeled data based on how they cluster with their neighbors.


Partially labeled data
Unlabeled
Label 0
Label 1
Feature 2
1.0
0.5
0.0
-0.5
-1.0
-1.0
-0.5
0.0
0.5
1.0
Feature store

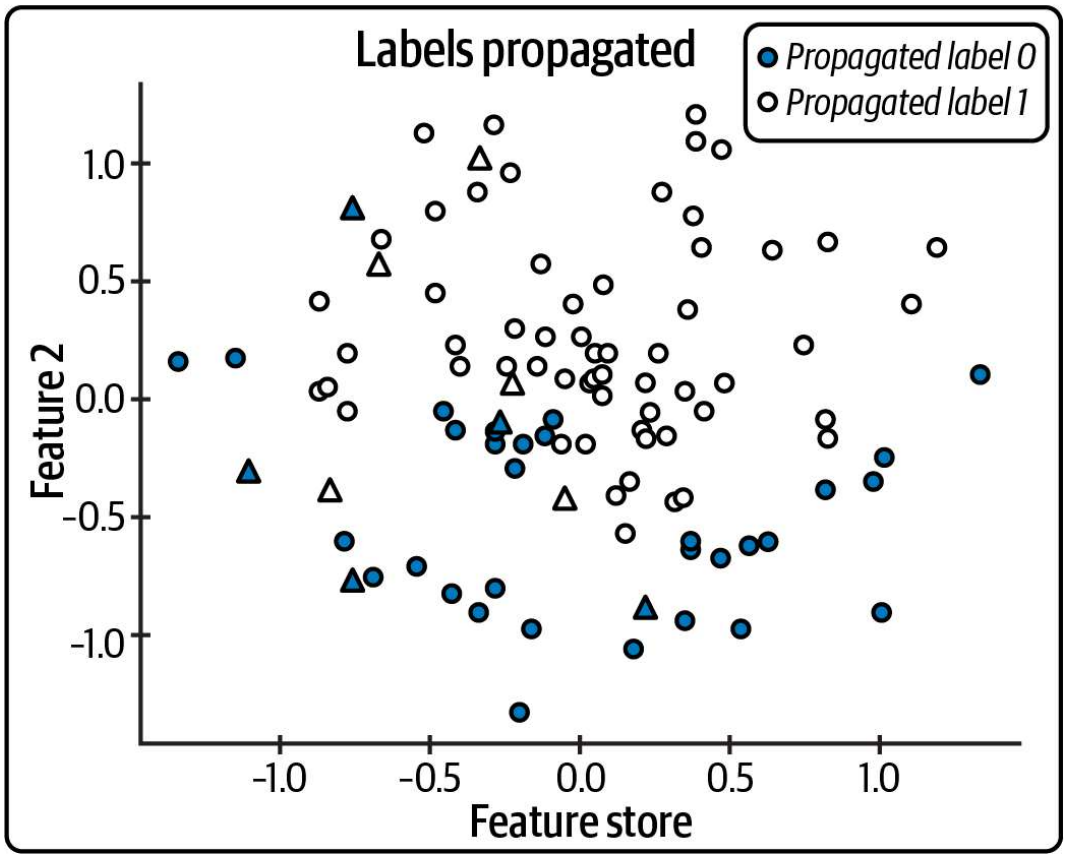

Labels propagated
Propagated label 0
Propagated label 1
Feature 2
1.0
0.5
0.0
-0.5
-1.0
-1.0
-0.5
0.0
0.5
1.0
Feature store

Figure 5-1. Label propagation

The labels are then propagated to the rest of the clusters, as indicated with different shades. We should mention that there are many different ways to do label propagation—graph-based label propagation is only one of several techniques. Label propagation itself is considered *transductive learning*, meaning we are mapping from the examples themselves, without learning a function for the mapping.

Sampling techniques

Typically, your labeled dataset will be much smaller than the available unlabeled dataset. If you're going to add to your labeled dataset by labeling new data, you need some way to decide which unlabeled examples to label. You could just select them randomly, which is referred to as *random sampling*. Or you could try to somehow select the best examples, which are those that improve your model the most. There are a variety of techniques for trying to select the best examples, and we'll introduce a few of these next.

Active Learning

Active learning is a way to intelligently sample your data, selecting the unlabeled points that would bring the most predictive value to your model. This is very helpful in a variety of contexts, including when you have a

limited data budget. It costs money to label data, especially when you're using human experts to look at the data and assign a label to it. Active learning helps you make sure you focus your resources on the data that will give you the most bang for your buck.

If you have an imbalanced dataset, active learning is an efficient way to select rare classes at the training stage. And if standard sampling strategies do not help improve accuracy and other target metrics, active learning can often offer a way to achieve the desired accuracy.

An active learning strategy relies on being able to select the examples to label that will best help the model learn. In a fully supervised setting, the training dataset consists of only those examples that have been labeled. In a semi-supervised setting, you leverage your labeled examples to label some additional, previously unlabeled examples in order to increase the size of your labeled dataset. Active learning is a way to select which unlabeled examples to label.

A typical active learning cycle proceeds as follows:

1. You start with a labeled dataset, which you use to train a model, and a pool of unlabeled data.
2. Active learning selects a few unlabeled examples, using intelligent sampling (as described in more detail in the sections that follow).

3. You label the examples that were selected with human annotators, or by leveraging other techniques. This gives you a larger labeled dataset.
4. You use this larger labeled dataset to retrain the model, potentially starting a new iteration of the active learning cycle.

But this begs the question: How do we do intelligent sampling?

Margin sampling

Margin sampling is one widely used technique for doing intelligent sampling. Margin sampling is a valuable technique for active learning that focuses on querying the most uncertain samples, those closest to the decision boundary, to improve the model's learning efficiency and performance.

In Figure 5-2, the data belongs to two classes. Additionally, there are unlabeled data points. In this setting, the simplest strategy is to train a binary linear classification model on the labeled data, which outputs a decision boundary.

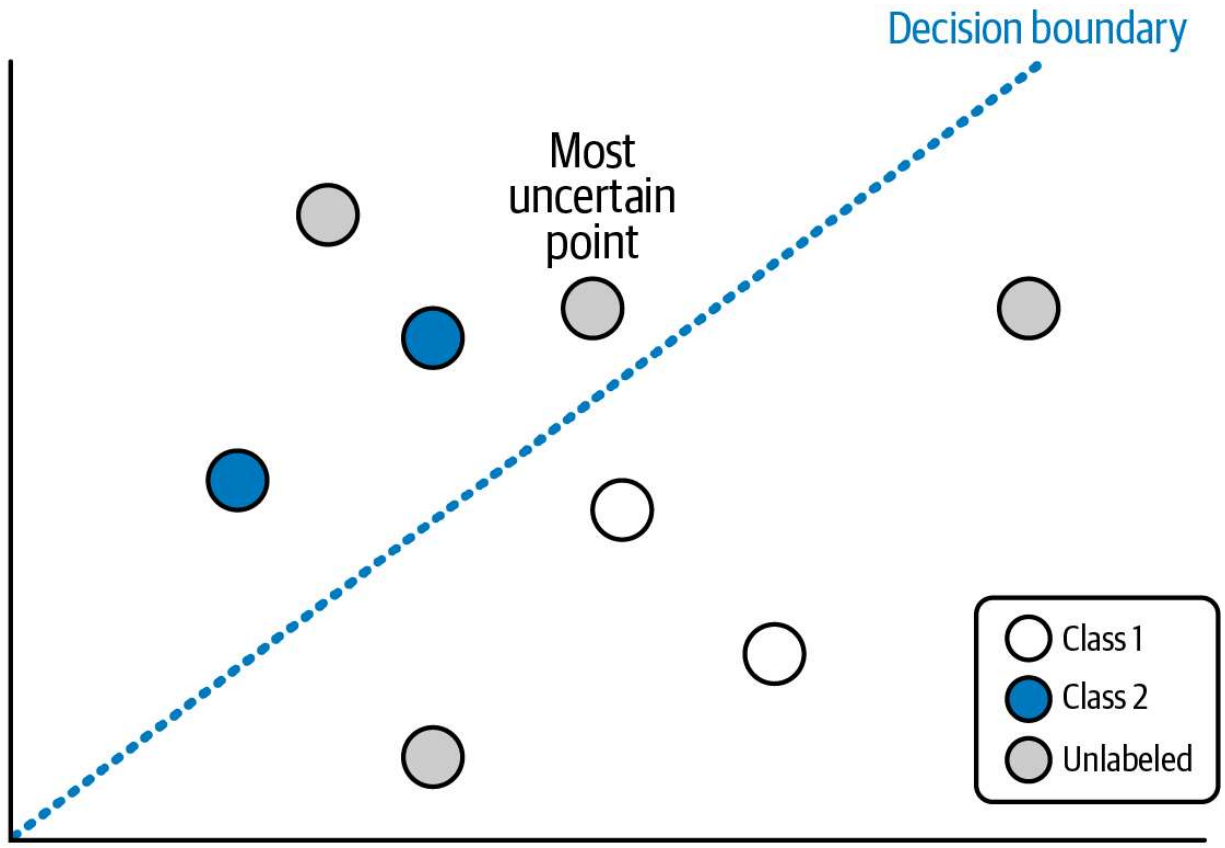


Figure 5-2. Margin sampling, initial state

With active learning, you select the most uncertain point to be labeled next and added to the dataset. Margin sampling defines the most uncertain point as the one that is closest to the decision boundary.

As shown in Figure 5-3, using this new labeled data point, you retrain the model to learn a new classification boundary. By moving the boundary, the model learns a bit better to separate the classes. Next, you find the next most uncertain data point, and you repeat the process until the model doesn’t improve.

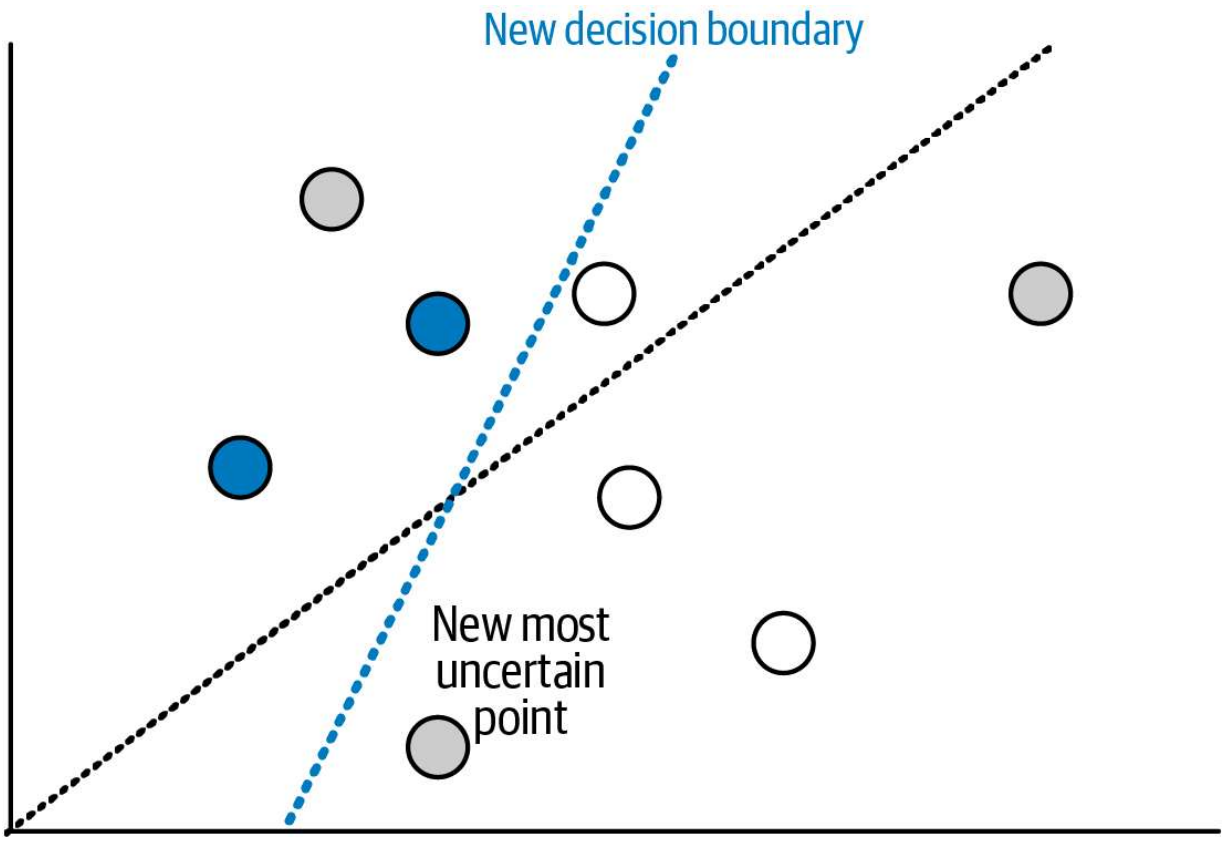


Figure 5-3. Margin sampling, after first iteration

Figure 5-4 shows model accuracy as a function of the number of training examples for different sampling techniques. The bottom line shows the results of random sampling. The top two lines show the performance of two margin sampling algorithms using active learning (the difference between the two is not important right now).

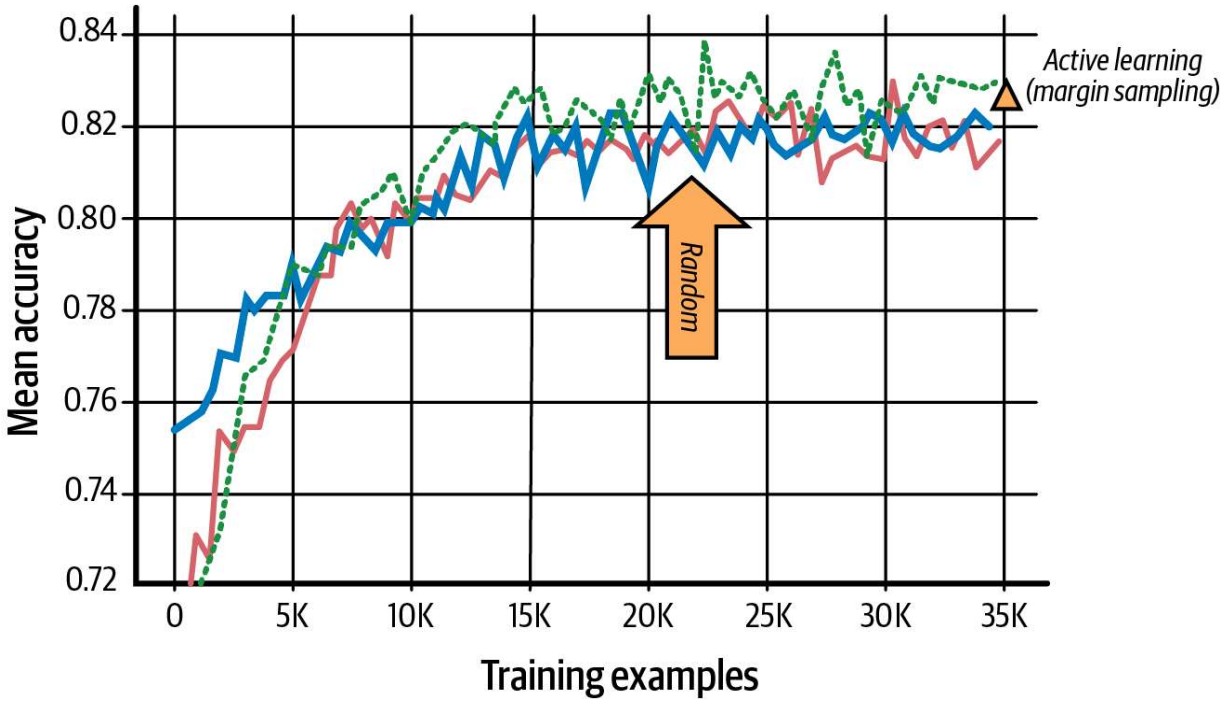


Figure 5-4. Intelligent sampling results

Looking at the x-axis you can see that margin sampling achieves higher accuracy with fewer training examples than random sampling. Eventually, as a higher percentage of the unlabeled data is labeled with random sampling, it catches up to margin sampling. This agrees with what we would expect if margin sampling intelligently selects the best examples to label.

Other sampling techniques

Margin sampling is only one intelligent sampling technique. With margin sampling, as you saw, you assign labels to the most uncertain points based on their distance from the decision boundary. Another technique is *cluster-based sampling*, in which you select a diverse set of points by using

clustering methods over your feature space. Yet another technique is *query by committee*, in which you train several models and select the data points with the highest disagreement among them. And finally, *region-based sampling* is a relatively new algorithm. At a high level, this algorithm works by dividing the input space into separate regions and running an active learning algorithm on each region.

Weak Supervision

> *Hand-labeling training data for machine learning problems is effective, but very labor and time intensive. This work explores how to use algorithmic labeling systems relying on other sources of knowledge that can provide many more labels but which are noisy.*
>
> —Jeff Dean, SVP, Google Research and AI, March 14, 2019

Weak supervision is a way to generate labels by using information from one or more sources, usually subject matter experts and/or heuristics. The resulting labels are noisy and probabilistic, rather than the deterministic labels that we're used to. They provide a signal of what the actual label should be, but they aren't expected to be 100% correct. Instead, there is some probability that they're correct.

More rigorously, weak supervision comprises one or more noisy conditional distributions over unlabeled data, and the main objective is to learn a generative model that determines the relevance of each of these noisy sources.

Starting with unlabeled data for which you don’t know the true labels, you add to the mix one or more weak supervision sources. These sources are a list of heuristic procedures that implement noisy and imperfect automated labeling. Subject matter experts are the most common sources for designing these heuristics, which typically consist of a coverage set and an expected probability of the true label over the coverage set. By “noisy” we mean that the label has a certain probability of being correct, rather than the 100% certainty that we’re used to for the labels in our typical supervised labeled data. The main goal is to learn the trustworthiness of each weak supervision source. This is done by training a generative model.

The Snorkel framework came out of Stanford in 2016 and is the most widely used framework for implementing weak supervision. It does not require manual labeling, so the system programmatically builds and manages training datasets. Snorkel provides tools to clean, model, and integrate the resulting training data that is generated by the weak supervision pipeline. Snorkel uses novel, theoretically grounded techniques to get the job done quickly and efficiently. Snorkel also offers data augmentation and slicing, but our focus here is on weak supervision.

With Snorkel, you start with unlabeled data and apply labeling functions (the heuristics that are designed by subject matter experts) to generate noisy labels. You then use a generative model to denoise the noisy labels and assign importance weights to different labeling functions. Finally, you train a discriminative model—*your* model—with the denoised labels.

Let's take a look at what a couple of simple labeling functions might look like in code. Here is an easy way to create functions to label spam using Snorkel:

```
from snorkel.labeling import labeling_function
 @labeling_function()
def lf_contains_my(x):
    # Many spam comments talk about 'my channel',
    return SPAM if "my" in x.text.lower() else AB
 @labeling_function()
def lf_short_comment(x):
    # Non-spam comments are often short, such as
    return NOT_SPAM if len(x.text.split()) < 5 el
```

The first step is to import the `labeling_function` from Snorkel. With the first function (`lf_contains_my`), we label a message as spam if it contains the word *my*. Otherwise, the function returns `ABSTAIN` , which means it has no opinion on what the label should be. The second

function (`lf_short_comment`) labels a message as not spam if it is shorter than five words.

Advanced Labeling Review

Supervised learning requires labeled data, but labeling data is often an expensive, difficult, and slow process. Let’s review the key points of advanced labeling techniques that offer benefits over supervised learning:

Semi-supervised learning

: Falls between unsupervised learning and supervised learning. It works by combining a small amount of labeled data with a large amount of unlabeled data. This improves learning accuracy.

Active learning

: Relies on intelligent sampling techniques that select the most important examples to label and add to the dataset. Active learning improves predictive accuracy while minimizing labeling cost.

Weak supervision

: Leverages noisy, limited, or inaccurate label sources inside a supervised learning environment that tests labeling accuracy. Snorkel is a compact and user-friendly system to manage all these operations and to establish training datasets using weak supervision.

Data Augmentation

In the previous section, we explored methods for getting more labeled data by labeling unlabeled data, but another way to do this is to augment your existing data to create more labeled examples. With data augmentation, you can expand a dataset by adding slightly modified copies of existing data, or by creating new synthetic data from your existing data.

With the existing data, it is possible to create more data by making minor alterations/perturbations in the existing examples. Simple variations such as flips or rotations in images are an easy way to double or triple the number of images in a dataset, while retaining the same label for all the variants.

Data augmentation is a way to improve your model's performance, and often its ability to generalize. This adds new, valid examples that fall into regions of the feature space that aren't covered by your real examples.

Keep in mind that if you add invalid examples, you run the risk of learning the wrong answer, or at least introducing unwanted noise, so be careful to only augment your data in valid ways! For example, consider the images in Figure 5-5.

Figure 5-5. An invalid variant

Let’s begin with a concrete example of data augmentation using CIFAR-10, a famous and widely used dataset. We’ll then continue with a discussion of some other augmentation techniques.

Example: CIFAR-10

The CIFAR-10 dataset (from the Canadian Institute for Advanced Research) is a collection of images commonly used to train ML models and computer vision algorithms. It is one of the most widely used datasets for ML research.

CIFAR-10 contains 60,000 color images measuring 32 × 32 pixels. There are 10 different classes with 6,000 images in each class. Let’s take a practical look at data augmentation with the CIFAR-10 dataset:

```
def augment(x, height, width, num_channels):
    x = tf.image.resuize_with_crop_or_pad(x, heig
    x = tf.image.random_crop(x, [height, width, n
    x = tf.image.random_flip_left_right(x)
    return x
```

This code creates new examples that are perfectly valid. It starts by cropping the padded image to a given height and width, adding a padding of 8 pixels. It then creates random translated images by cropping again, and then randomly flips the images horizontally.

Other Augmentation Techniques

Apart from simple image manipulation, there are other advanced techniques for data augmentation that you may want to consider. Although we won't be discussing them here, these are some techniques for you to research on your own:

- Semi-supervised data augmentation
- Unsupervised Data Augmentation (UDA)
- Policy-based data augmentation (e.g., with AutoAugment)

While generating valid variations of images is easy to imagine and fairly easy to implement, for other kinds of data the augmentation techniques and the types of variants generated may not be as straightforward. The

applicability of different augmentation techniques tends to be specific to the type of data, and sometimes to the domain you're working in. This is another one of those areas where the ML engineering team's skill and knowledge of the data and domain are critical.

Data Augmentation Review

Data augmentation is a great way to increase the number of labeled examples in your dataset. Data augmentation increases the size of your dataset, and the sample diversity, which results in better feature space coverage. Data augmentation can reduce overfitting and increase the ability of your model to generalize.

Preprocessing Time Series Data: An Example

Data comes in a lot of different shapes, sizes, and formats, and each is analyzed, processed, and modeled differently. Some common types of data include images, video, text, audio, time series, and sensor data. Preprocessing for each of these tends to be very specialized and can easily fill a book, so instead of discussing all of them, we're going to look at only one: time series data.

A *time series* is a sequence of data points in time, often from events, where the time dimension indicates when the event occurred. The data points may or may not be ordered in the raw data, but you will almost always want to order them by time for modeling. Inherently, time series problems are almost always about predicting the future.

> *It is difficult to make predictions, especially about the future.*
>
> —Danish proverb

Time series forecasting does exactly that: it tries to predict the future. It does this by analyzing data from the past. Time series is often an important type of data and modeling for many business applications, such as financial forecasting, demand forecasting, and other types of forecasting that are important for business planning and optimization.

For example, to predict the future temperature at a given location we could use other meteorological variables, such as atmospheric pressure, wind direction, and wind velocity, that have been recorded previously. In fact, we would probably be using a weather time series dataset similar to the one that was recorded by the Max Planck Institute for Biogeochemistry. That dataset contains 14 different features including air temperature, atmospheric pressure, and humidity. The features were recorded every 10 minutes beginning in 2003.

Let’s take a closer look at how that data is organized and collected. There are 14 variables including measurements related to humidity, wind velocity and direction, temperature, and atmospheric pressure. The target for prediction is the temperature. The sampling rate is 1 observation every 10 minutes, so there are 6 observations per hour and 144 in a given day (6 × 24). The time dimension gives us the order, and order is important for this dataset since there is a lot of information in how each weather feature changes between observations. For time series, order is almost always important.

Figure 5-6 shows a plot of a temperature feature over time. You can see that there’s a pattern to this that repeats over specific intervals of time. This kind of repeating pattern is referred to as *seasonality*, but it can be any kind of repeating pattern and does not need to have anything to do with the seasons of the year. There’s clear seasonality here, which we need to consider when doing feature engineering for this data.

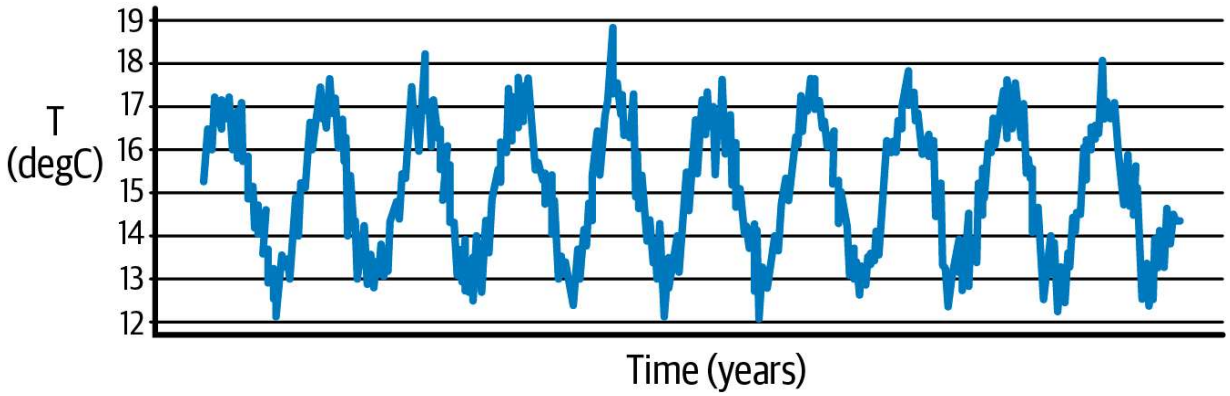


Figure 5-6. Weather periodicity showing seasonality

We should consider doing seasonal decomposition, but to keep things simple in this example we won't be doing that. Instead, we'll be focusing on windowing and sampling, which can be used with or without seasonal decomposition. Seasonal decomposition is used to improve the data and focus on the residual, and is often used in anomaly detection.

Windowing

Using a windowing strategy to look at dependencies with past data seems to be a natural path to take. Windowing strategies in time series data become pretty important, and they're kind of unique to time series and similar types of sequence data. The example in Figure 5-7 shows one windowing strategy that you might use for a model you want to use to make a prediction one hour into the future, given a history of six hours.

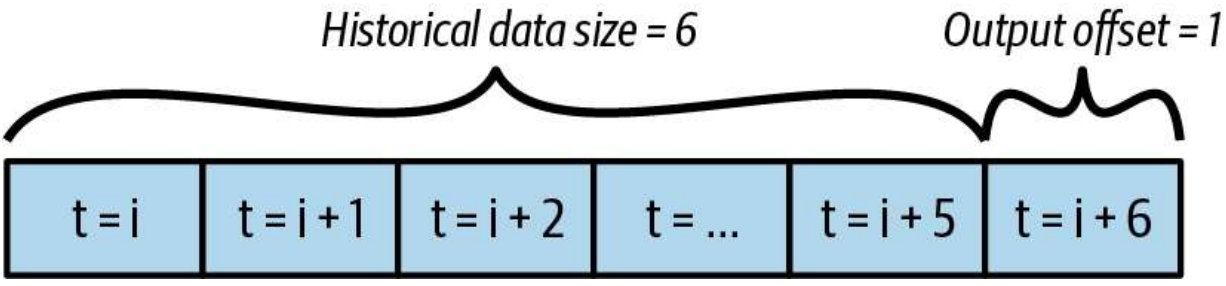


Figure 5-7. An example of a windowing strategy for making a prediction one hour into the future, given a history of six hours

Figure 5-8 shows a windowing strategy that you might use if you want to make a prediction 24 hours into the future, given 24 hours of history, so in that case, your history size is 24.

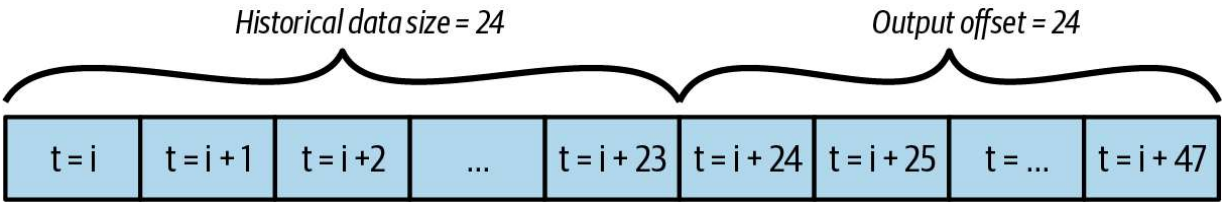


Figure 5-8. An example of a windowing strategy for making a prediction 24 hours into the future, given a history of 24 hours

In Figure 5-8, the offset size is also 24, so you could use a total window size of 48, which would be the history plus the output offset. It's also important to consider when "now" is, and to make sure you omit data pertaining to the future (i.e., time travel). In this example, if "now" is at t = 24, we need to be careful not to include the data from t = 25 to t = 47 in our training data. We could do that in feature engineering, or by reducing the window to include only the history and the label. If during training we were to include data about the future in our features, we would not have that data available when we use the model for inference, since the future hasn't happened yet.

Sampling

It's also important to design a sampling strategy. You already know that there are six observations per hour in our example, one observation every 10 minutes. In one day, there will be 144 observations. If you take five days of past observations and make a prediction six hours into the future, that means our history size will be 5 × 144 or 720 observations, the output offset will be 6 × 6 or 36, and the total time window size will be 792. Figure 5-9 shows visually what we mean by the total window size, history, and offset.

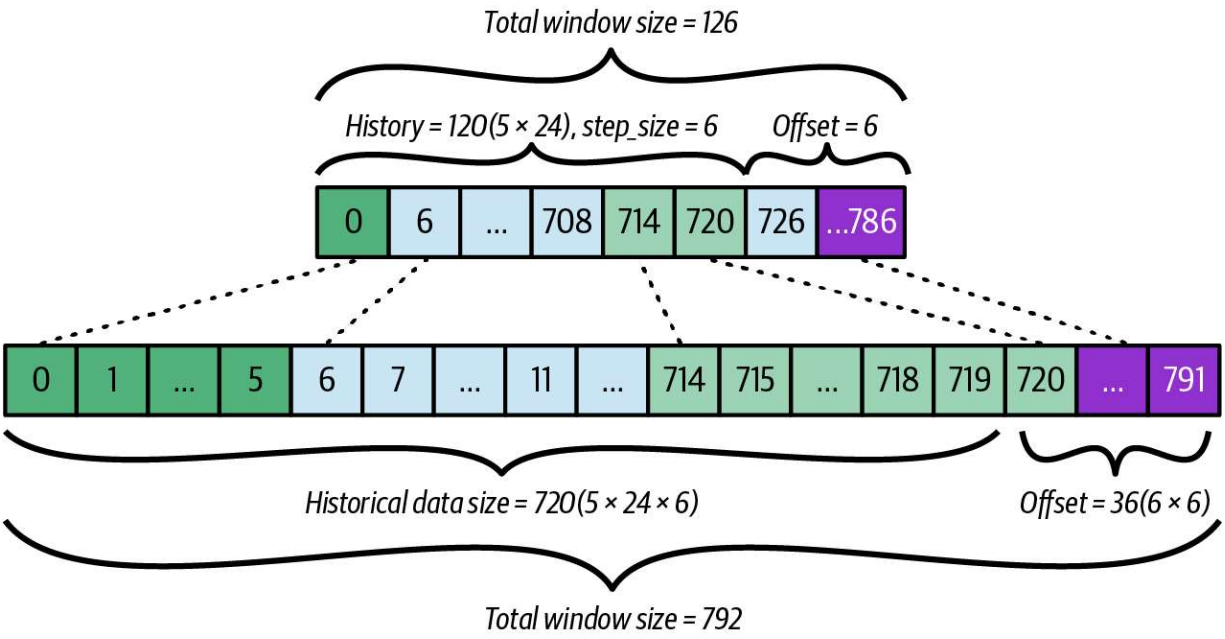


Figure 5-9. Improving a sampling strategy

Since observations in one hour are unlikely to change much, let's sample one observation per hour. We could take the first observation in the hour as a sample, or even better, we could take the median of the observations for each hour.

Then our history size becomes 5 × 24 × 1 or 120, and our output offset will be 6, so our total window size becomes 126. In this way, we've reduced the size of our feature vector from 792 to 126 by either sampling within each hour, or aggregating the data for each hour by taking the median.

This example is intended to be a short introduction to time series data. For a more in-depth look at time series data, you can refer to the TensorFlow documentation.

Conclusion

This chapter provided some background and perspective on the importance of labeling and preprocessing to successful modeling, along with the advantages of data augmentation as a method to expand on the information in the dataset. With new modeling techniques being developed at an amazing pace for generative AI and new forms of deep learning, new techniques for labeling, preprocessing, and data augmentation are also being developed. While this chapter did not cover all the techniques that exist today, it should give you a good understanding of the kinds of approaches to take and ways to think about these important areas. Remember, your model is only as good as the information in your data, and anything you can do to make it easier for your model to learn from that information will result in a better model.