EXPLORIUM

How to Improve Your Training Data for Vastly Better Machine Learning

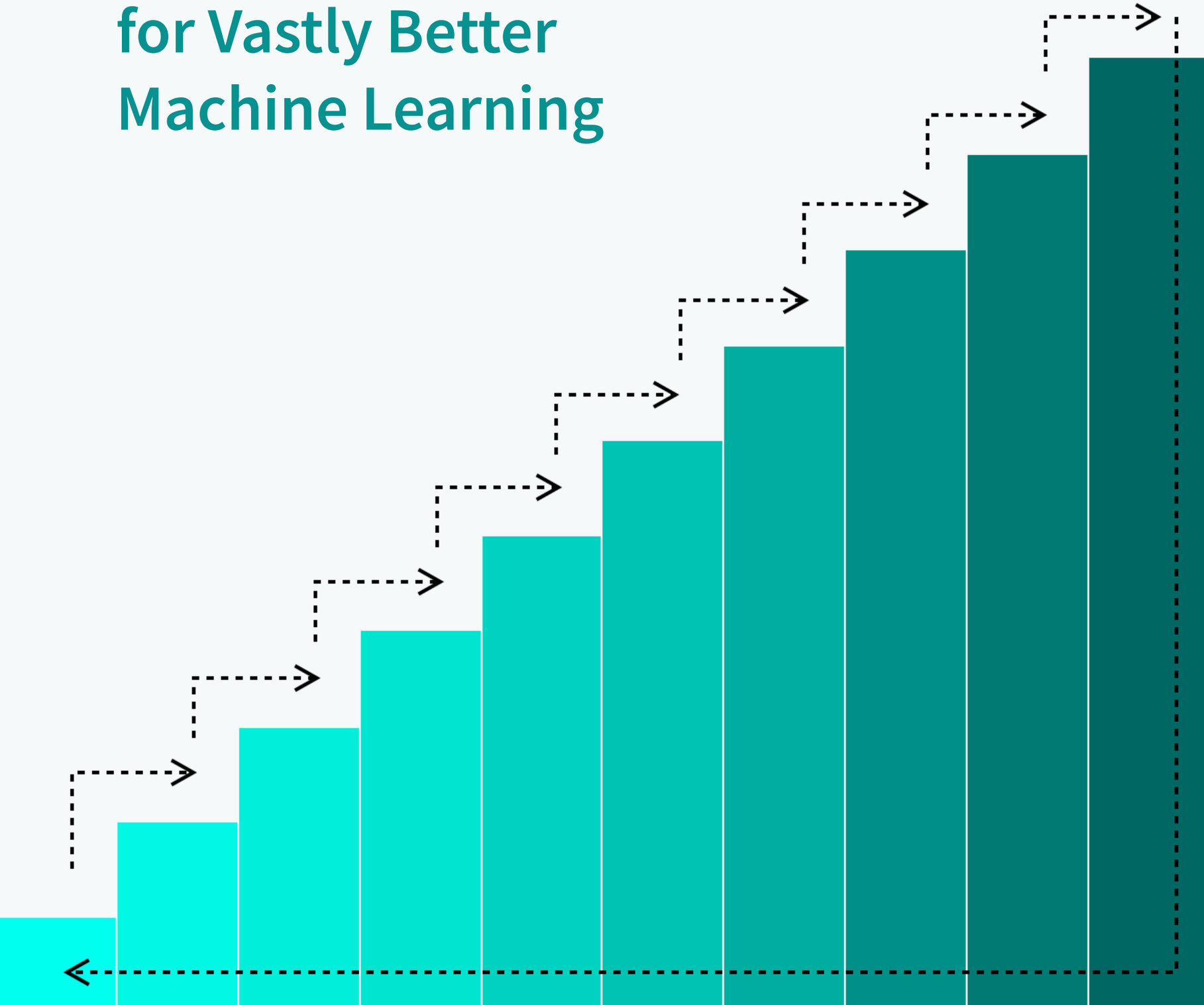


Table of contents

Imagine you're revising for an exam. You decide to prepare yourself for real-world questions by working your way through past papers, noting which problems come up again and again, and focusing your time and attention on these topics. By the time the exam comes up, you're ready to nail any question along these lines.

But then: disaster. The real exam paper goes off in a completely different direction. It focuses on topics so new, they weren't covered in last year's paper. You've been training hard, but you've been training for the wrong questions. And now you're stuck.

This is, in a nutshell, why so many machine learning projects fail. You may have created something truly sophisticated and intelligent, but if you train it using the wrong data, to respond to the wrong queries, it will fall apart when you unleash it on real-world problems.

Over the next 29 pages, we'll cover tips and techniques for generating and applying better training data. Data that is both broad enough and specific enough to cover all the bases you need, so that it doesn't conflict with real-world data later.

Are you ready? Then let's begin.

Raw data can be a chaotic mess, but you simply can't afford to cut corners when getting it fit for purpose. You need to take your time to properly preprocess data and carefully select the best techniques, whether these are straightforward or complex.

Does your data collection need an overhaul?

Raw data can be a chaotic mess, but you simply can't afford to cut corners when getting it fit for purpose. You need to take your time to properly pre-process data and carefully select the best techniques, whether these are straightforward or complex.

Here are six tips to get you started:

1. Start by **removing outliers**, as these can confuse your results and send algorithms down the wrong path. If you're using Python for your machine learning modeling, for instance, you can use Pandas to quickly identify outliers and remove them.

   You can run a quick test for outliers by using interquartile ranges (IQR) with Pandas, with a simple script:

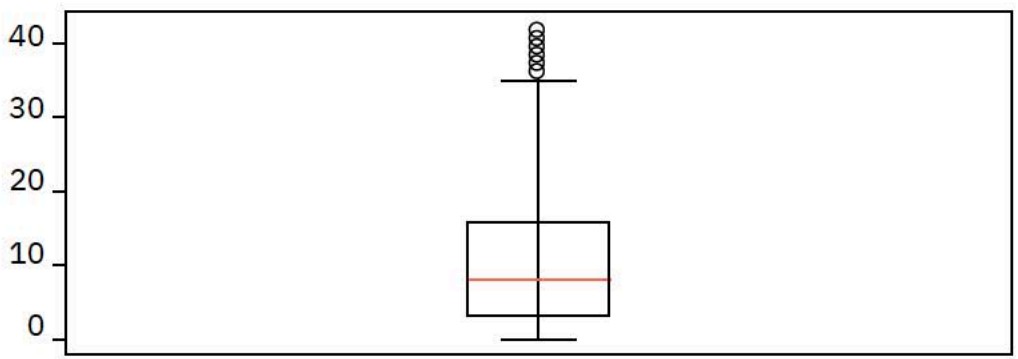
```
import pandas as pd
df = pd.read _ csv("dataset.csv")
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)

print(df < (Q1 - 1.5*IQR))| (df > (Q3 + 1.5*IQR))
```

The resulting table will populate each cell with boolean values, where "True" represents an outlier value and "False" represents a value within the normal distribution.

Alternatively, you can use visual methods with Matplotlib to put your data into box plots that will help you remove extreme values from your set:

```
df = pd.read_csv('testset.csv')
plt.boxplot(df["experience"])
plt.show()
```

Your output will look like this:



Where the dots outside the brackets are considered your outliers.

2. **Normalization** is also important to figure out meaningful perimeters and categories, and ultimately to make sense of your results. The problem with data that isn't normalized is that in some cases different columns might have such vastly different values that you may get results that make little sense, or that affect your model's overall value.

   To avoid this issue, you can use scikit-learn's preprocessing package to fix your values in a few lines of code (in this case, we'll be using a sample dataset of California housing prices):

```
import pandas as pd
from sklearn import preprocessing

df = pd.read_csv('california_housing_train.csv')
min_max_scaler = preprocessing.MinMaxScaler()
df_scaled = min_max_scaler.fit_transform(df)
df = pd.DataFrame(df_scaled)

print(df)
```

The script will turn this table:

	longitude	latitude	...	median_income	median_house_value
0	-114.31	34.19	...	1.4936	66900.0
1	-114.47	34.40	...	1.8200	80100.0
2	-114.56	33.69	...	1.6509	85700.0
3	-114.57	33.64	...	3.1917	73400.0
4	-114.57	33.57	...	1.9250	65500.0
...	...	...	...	...	...
16995	-124.26	40.58	...	2.3571	111400.0
16996	-124.27	40.69	...	2.5179	79000.0
16997	-124.30	41.84	...	3.0313	103600.0
16998	-124.30	41.80	...	1.9797	85800.0
16999	-124.35	40.54	...	3.0147	94600.0

Into this:

	0	1	2	...	6	7	8
0	1.000000	0.175345	0.274510	...	0.077454	0.068530	0.107012
1	0.984064	0.197662	0.352941	...	0.075974	0.091040	0.134228
2	0.975100	0.122210	0.313725	...	0.019076	0.079378	0.145775
3	0.974104	0.116897	0.254902	...	0.037000	0.185639	0.120414
4	0.974104	0.109458	0.372549	...	0.042921	0.098281	0.104125
...	...	...	...	...	...	...	...
16995	0.008964	0.854410	1.000000	...	0.060516	0.128081	0.198764
16996	0.007968	0.866100	0.686275	...	0.076303	0.139170	0.131960
16997	0.004980	0.988310	0.313725	...	0.074823	0.174577	0.182682
16998	0.004980	0.984060	0.352941	...	0.078441	0.102054	0.145981
16999	0.000000	0.850159	1.000000	...	0.044236	0.173432	0.164125

Giving you a standardized set of values that you can use with a linear regression without affecting your accuracy.

3. Next, turn your attention to variables. Make sure you **address any missing variables**, whether that means inferring the missing value from other data attributes, and inputting this into your dataset, or by deleting the whole record if necessary.

   In some cases, you'll be able to simply replace any null value with a 0, in which case you can use Pandas to simply fill them (we'll build a small table to exemplify):

```
import pandas as pd
import numpy as np

df = pd.DataFrame([[np.nan, 2, np.nan, 0],
                   [3, 4, np.nan, 1],
                   [np.nan, np.nan, np.nan, 5],
                   [np.nan, 3, np.nan, 4]],
                  columns = list('ABCD'))
```

Our table will look like this:

	A	B	C	D
0	NaN	2.0	NaN	0
1	3.0	4.0	NaN	1
2	NaN	NaN	NaN	5
3	NaN	3.0	NaN	4

To fill the values, we can simply use Pandas' .fillna() function

```
df = df.fillna(0)
```

For an output that looks like this:

	A	B	C	D
0	0.0	2.0	0.0	0
1	3.0	4.0	0.0	1
2	0.0	0.0	0.0	5
3	0.0	3.0	0.0	4

4. If things are overcomplicated, you may also end up having to **reduce the number of variables** you use altogether, using approaches such as clustering, principal component analysis, or generative adversarial networks (GANs). Focusing on the most important ones will help you refine your model.

   Let's see how we could consolidate our variables using a basic K-means clustering algorithm. We'll use the "iris" dataset, which is widely available, for an example. Let's use a 5-fold clustering:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

df = pd.read_csv('IRIS.csv')
df.head(10)

x = df.iloc[:, [0,1,2,3]].values

kmeans5 = KMeans(n_clusters=5)
y_kmeans5 = kmeans5.fit_predict(x)
print(y_kmeans5)

kmeans5.cluster_centers_
```

This will give us an initial division of our dataset into clusters, but we can visualize it using the elbow method:

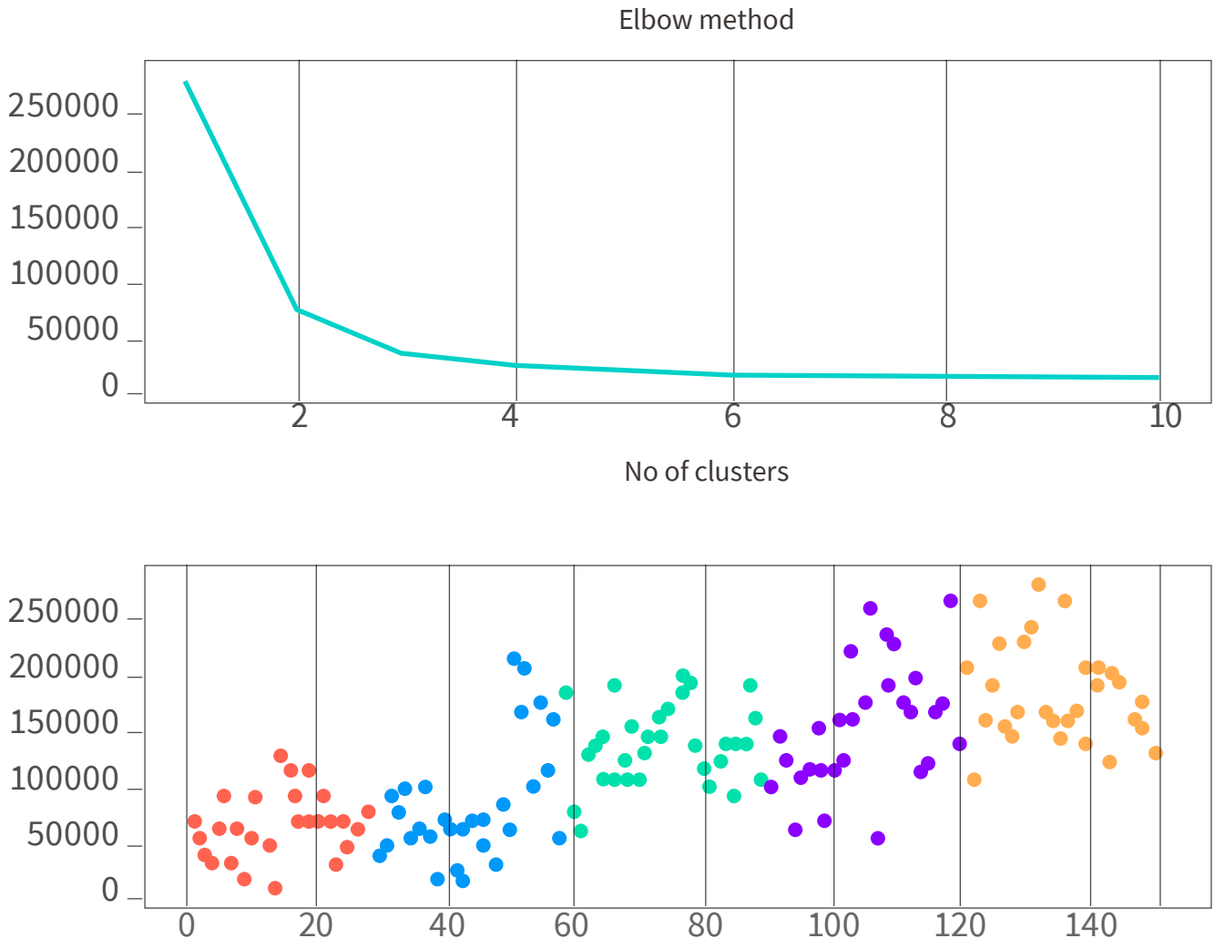
```
Error =[]
for i in range(1, 11):
    kmeans = KMeans(n _ clusters = i).fit(x)
    kmeans.fit(x)
    Error.append(kmeans.inertia _ )
plt.plot(range(1, 11), Error)
plt.title('Elbow method')
plt.xlabel('No of clusters')
plt.ylabel('Error')
plt.show()
```

As well as in a scatter plot:

```
plt.scatter(x[:, 0], x[:, 1], c=y _ kmeans5, cmap='rainbow')
```

In the end, we'll get this:

```
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3]
```



K-means clustering will help determine the best number of clusters (or variables) to separate your datasets into. This will help you start organizing your data better.

5. Machine learning also works better with continuous data attributes, so consider **feature engineering** or vectorization of categorical variables.

   If you're simply looking for a quick fix to convert your categorical variables into numerical or binary values, you can use Pandas' .get_dummies() function. Again, using our 'iris' dataset, let's see how it would work:

```
import pandas as pd
df = pd.read _ csv('IRIS.csv')
pd.get _ dummies(df)
```

   The resulting table will look like this:

	Unnamed: 0	Sepal.Length	...	Species_versicolor	Species_virginica
0	1	5.1	...	0	0
1	2	4.9	...	0	0
2	3	4.7	...	0	0
3	4	4.6	...	0	0
4	5	5.0	...	0	0
..	...	...	...	...	...
145	146	6.7	...	0	1
146	147	6.3	...	0	1
147	148	6.5	...	0	1
148	149	6.2	...	0	1
149	150	5.9	...	0	1

6. Machine learning models tend to veer towards over-rationalization, assuming that patterns are normal and downplaying anomalies or other rare events. If your datasets are unbalanced, you may need to **upsample.** Upsampling is a way of creating synthetic data by emphasizing and duplicating rare data, highlighting unusual events.

   Using our 'iris' dataset one last time, let's quickly create an imbalance to demonstrate how upsampling works:

```
import numpy as np
from sklearn.datasets import load _ iris

iris = load _ iris()
X = iris.data
y = iris.target

X = X[40:,:]
y = y[40:]
```

   The script will turn this table:

```
[0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

Since this will affect our results and our model's accuracy, our goal in upsampling isn't to simply randomly create new data, but to use our existing information to duplicate data points and balance our datasets:

```
i_class0 = np.where(y==0)[0]
i_class1 = np.where(y==1)[0]

n_class0 = len(i_class0)
n_class1 = len(i_class1)

i_class0_upsample = np.random.choice(i_class0, size=n_
class1, replace = True)
print(np.concatenate((y[i_class0_upsample], y[i_class1])))
```

The resulting array won't have unexpected data points, but it will balance out our variables to our targets, meaning that while our accuracy may decrease (due to the lack of imbalance), we'll have better models in the long run:

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[1111111111111111111111111111111111111111111111111[11111111111111111
1]
```

These are just some of the many techniques you might consider using to pre-process your data. If you're using a data science platform, this will likely have appropriate tools built-in. Explore and apply as appropriate, depending on the specific challenges presented by your training data.

Acquiring more data

One of the biggest hurdles for data scientists trying to build effective machine learning models is accessing enough high-quality training data. It's vital that this is varied, accurate, extensive, and at the same time, nuanced enough to reflect the kind of information the model will encounter in real life.

Augmented data discovery

Building robust models means you will likely need to incorporate external datasets that add value to your existing datasets. This third-party information may include census data, weather updates, traffic reporting, marketing campaign responses, or even unstructured, natural-language text collected from blogs and social media accounts. Bringing in external datasets like these is particularly beneficial for demand forecasting and customer behavior prediction models. That said, without the right tools and platforms in place, it can be highly resource-heavy and time-intensive to acquire external data.

One of the biggest hurdles for data scientists trying to build effective machine learning models is accessing enough high-quality training data.

With augmented data discovery, data scientists can connect automatically to thousands of external datasets. This automates the process of data profiling, cleaning, modeling, enrichment, metadata labeling and so on, which is otherwise an enormous task. You simply connect to ready-cleaned and normalized external data and explore what impact it has on your model.

Within augmented data discovery, data enrichment is particularly important when you're looking to boost your training data. This involves picking out the most relevant datasets and data points from the external data. You can then combine these details with your existing datasets to greatly enhance their value, helping you tease out the answers you need. That might be something as simple as adding zip codes or other location data into existing datasets you hold on customers (or potential customers). The process of enrichment is built into augmented data discovery, saving you a lot of time and hassle.

Crowdsourcing

Crowdsourcing means accumulating useful information from a specified pool of respondents rather than simply buying a complete dataset from a professional provider.

Broadly speaking, crowdsourcing refers to any situation where you get lots of different people to chip in ideas, information, or content for a particular project. In the context of generating training data, crowdsourcing

usually means you recruit a hive of data-harvesting or organizing worker bees who all contribute to cleaning up or improving the quality of your data. Typically, you would do this through a dedicated platform rather than amassing that hive yourself.

Crowdsourcing is a useful way to amass data (e.g. from open/publicly available sources) and metadata (e.g. labels and content classifications). You could also go down this route to commission transcriptions of PDFs, captioning of audio and video files, product categorization, image annotation, and cropping. Subjective tasks like moderating content, assessing ads, and validating translations performed by other machine learning tools are all good fits for crowdsourcing.

Feature engineering

While we're on the subject of getting more out of your data, feature engineering is a clever way to mine more insights out of internal or external datasets by pushing the basic information further.

Essentially, you create new input features from the ones you already have. This allows you to isolate or highlight the most important information, guiding your algorithms towards the really crucial stuff. It also means you can bring in your own — or others' — domain expertise, adding value to the model.

For example, let's say you have a list of purchase dates in your sales

information. These dates actually tell you a lot more than you might think at first glance. Beyond the sales quarter figures, from this simple string of numbers, you can extract what month the sale was made on. You know which day of the week you made a sale. You can tell how far through the month the sale occurred — whether, for example, it was likely to be right after payday. By extracting this information and using it to generate new tabular data for you to model and analyze, you get to create valuable new data points to enrich your machine learning models.

Feature engineering is also used to encode images in a way that can be used for machine learning analysis, to mathematically derive new features from numerical data, to handle and input missing data, and to convert text into representative numerical values.

Synthesizing data

Sometimes, it's not possible to get hold of sufficient data from the real world to achieve what you need. Instead, you can synthesize this, using domain insights, data augmentation, or even advanced simulation.

For instance, by generating synthetic speech data, you create more training data for natural language processing and conversational systems. By altering 2D images or creating 3D models of objects, you can produce

multiple perspectives, colors, types of lighting, and other variations on an image which can be used to train recognition systems. Similar techniques with human facial expressions are used to generate more nuanced training data for facial recognition algorithms.

These techniques are becoming increasingly popular across a broad range of industries and sectors. For example, in the transportation sector, simulated street views and driving conditions help to train driverless vehicles more effectively. This is also beneficial in the security and defense sector, when training satellites and drones to respond to a range of nuanced attacks and threats. Meanwhile, in robotics, you can generate a broader range of navigation scenarios and poses, testing how the technology reacts.

There are a number of approaches you could take to synthesizing data. Two of the most common (and effective) are data augmentation and building simulation models.

Data augmentation

The ever-increasing quantity and diversity of data that's gathered every year is a major driver in the development of deep learning models. However, not every organization is in a position to collect this volume of new data. In other situations, when you're dealing with a new or emerging concept or

phenomenon, it can be too difficult (or even impossible) to gather enough different examples and variations of data in order to train a model.

In these cases, data augmentation provides a useful strategy that allows data scientists to gather a much greater diversity of data used to train models — without collecting any more actual data.

This is a relatively simple way to synthesize data: you transform a genuine data item to create a synthetic variation. For example, you might crop, rotate, mirror, blur, or adjust the color of an image or video clip. You might introduce another object into an image, or filter certain noises from audio.

Let's say you want your machine learning model to recognize a lamp. From a single image of a lamp, you could apply a number of simple transformations, showing the lamp in different colors, positioned at different angles, stretched, or squashed into different shapes, the lightbulb illuminated or dark.

In this way, a single instance of real data becomes dozens or potentially hundreds of instances, all of which you use to train the model's classifications. In the example above, your model now recognizes all of these variations as representing a lamp, improving its ability to recognize one in real-world data later on.

Simulation models

Simulation models are a much more complicated alternative to data augmentation. Rather than simply altering an image, video, text, or audio file, you build an entire virtual environment in which to test the behavior of your model.

This could mean that you directly replicate a specific environment that your machine learning-based technology will be used in, but within which you can alter the conditions at will. Or it could mean that you craft a number of entirely new environments, allowing you to test the model in unfamiliar scenarios to see how it performs.

This is particularly useful when you need to train a model to respond to dangerous, high-risk situations, or to perform tasks with serious real-world consequences, without actually putting anyone in harm's way or causing genuine physical destruction. For example, if you were training a self-driving vehicle to stop when a pedestrian steps out into the road, you obviously can't use real pedestrians until the vehicle figures out what it's supposed to do through a process of trial and error.

Developing simulation models is, of course, a complex, big-budget, sophisticated strategy. It's not something every organization will simply decide to dabble in. However, it's well worth thinking about ways you can

simulate your machine learning model's responses in situations where appropriate training data is tricky to come by.

Finding the right platform

The prominent and pressing challenge for many organizations isn't so much getting enough data as getting enough recent, relevant data. Your business context changes so quickly that data collected a year ago might be entirely useless to you as training data for a new machine learning model.

That's because your training data is only helpful if it reflects your current business operations. If they've changed since you collected the data, or they are about to change significantly, you may end up training your models with data that's entirely at odds with the kind of data you plan to feed it once you go live.

If that's the case for your organization, you really need to find alternative data sources that are up to date and reflect the kind of data you ultimately plan to use.

There are all kinds of public and commercial datasets on offer for organizations serious about enriching their training data. This creates a new challenge, though: how do you navigate the labyrinth? How do you cut

straight to what you need? How do you ensure that the steps you take to improve your machine learning models don't stall development and drive up costs to an unfeasible degree?

for you. One that facilitates augmented data discovery and feature generation, right off the bat. One that opens up access to enough new datasets to get you past the dimensionality curse that limits to many data science teams at the training data stage. One that strips out ambiguities and other headaches by harmonizing data for you.

Final thoughts

Boosting your training data means staying focused on exactly what it is you're training for. You aren't designing machine learning models to help you make predictions based on how your company operated five years ago. You aren't creating algorithms that help you make sense of the incomplete datasets you collected in the past. You're trying to build a model that will help you tackle key, emerging problems in your industry, or to identify patterns that help you predict where your business is headed next.

The prominent and pressing challenge for many organizations isn't so much getting enough data as getting enough recent, relevant data. Your business context changes so quickly that data collected a year ago might be entirely useless to you as training data for a new machine learning model.

That means you need to be proactive about seeking out the very best, most valuable, thorough, complete, and nuanced data to help you develop these models. If you don't have this data in-house, which is more than likely, you need to look further afield for it.

The right technology cuts out the middlemen, keeping down costs while granting you access to the exact datasets that truly add value. This not only helps you figure out exactly what you need, it also allows you to treat this data as a single source of truth. This kind of peace of mind means you can stop worrying about your training data – and focus on building machine learning models that knock your competitors out of the park.

EXPLORIUM

About Explorium

Our automated data discovery and feature generation platform automatically connects a company's data to thousands of relevant premium, partner, and open data sources to extract an optimal feature set based on model impact. We're creating a new category as the first company to empower and service business leaders and data scientists with end-to-end automation of data discovery and feature generation —

fueling superior decision-making and driving real business impact.