EXPLORIUM

Making Sense of Deployment: Feature Engineering, Training, Testing, and Monitoring

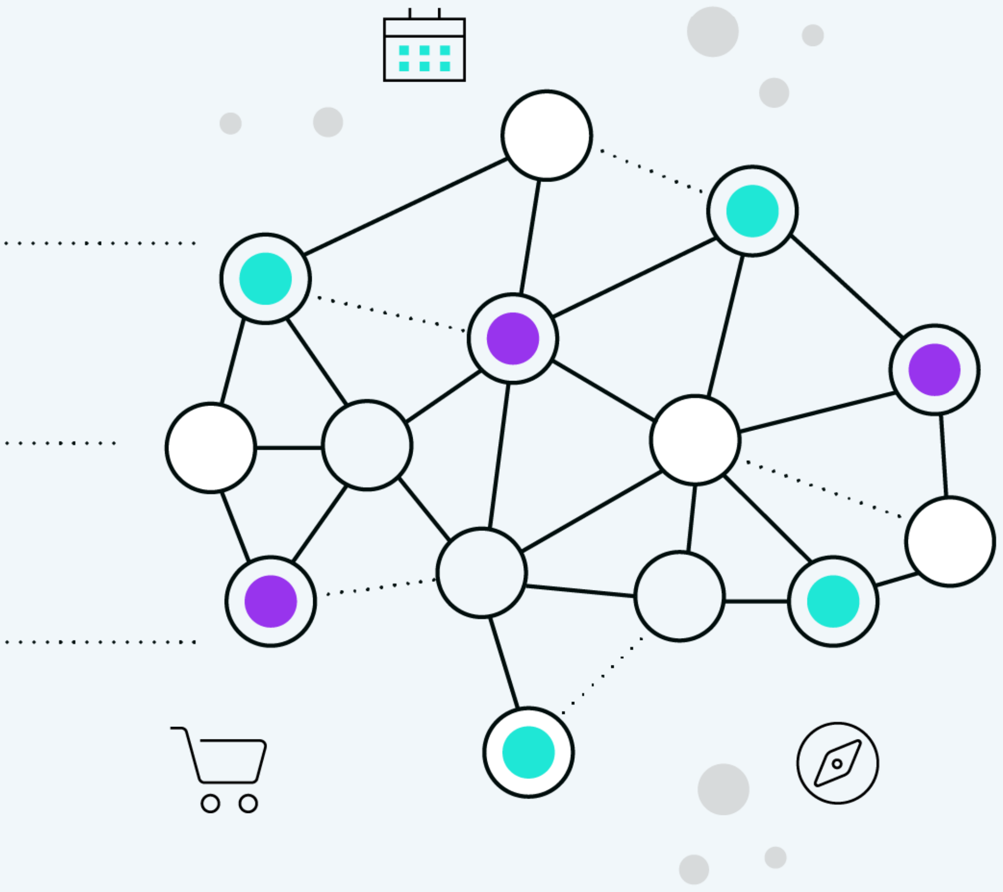


Table of contents

Deploying Your Data - Feature Engineering, Training, and Monitoring

Congratulations! You've spent a considerable amount of time working on getting your data just right and now you're ready to leave that step behind and focus on your machine learning (ML) models. Well, yes and no. You are definitely ready to start focusing on your models and getting insights, but that doesn't mean you're done with data. Far from it, actually.

"By 2022, 70% of organizations will rigorously track data quality levels via metrics, increasing data quality by 60% to significantly reduce operational risks and costs"

- Gartner Research

Building your dataset, cleaning it, and enriching it is an important step in the data workflow, and the steps that follow would be near impossible without it. However, now it's time to shift gears regarding how we approach data, and what we're doing with it. Once you've finished building your new, unified dataset, it's time to actually start mining it for insights and using it to build models that will give you the best results to your predictive questions.

It’s true that how you’re interacting with your data changes at this point, but that doesn’t mean you can ignore your datasets. It just means that while you’re done building your datasets, you’re only just started extracting value from them.

In this whitepaper, we’ll discuss the process of feature engineering and extraction, and why you should look for automation first. We’ll also cover training and testing your machine learning models, as well as the different methods for both. Finally, we’ll talk about the importance of setting up mechanisms to continually monitor and refresh your data to keep your models relevant.

Feature engineering

Think about all the data you've collected while getting ready to build your machine learning models. You've prepared it, cleaned it, organized it, and have it ready to use, but now is where you need to go a step further. See, data by itself doesn't give you the insights you need. It's how you combine the data to find possible predictors to your questions that really makes your data shine. How? Through feature engineering.

What is feature engineering?

Feature engineering means finding potential predictors for the questions you'll be asking when you come to build your machine learning model.

Now that you have your enriched dataset, you have more columns and data points to view in tandem, more possible connections you could be making between them, more patterns to establish, more correlations to explore. In other words, you can start extracting more features, or predictor variables, that will help you train your machine learning models. Ultimately, these features can greatly help you improve the performance of your models.

Feature engineering can be an easy task or a very difficult one, depending on what you need and if you're doing it manually.

Manual feature engineering

When you engineer features by yourself, you're going through the data with a fine-toothed comb, looking for any interesting patterns or links and using your domain knowledge to assess whether you think that these are relevant, meaningful, or worth pursuing further. That's extremely time-consuming and resource-intensive. It also opens you up to all kinds of errors and biases.

While domain knowledge is useful for interpreting what you're looking at, it can have a blinkering effect. Given this approach's subjectivity, you're more likely to overstate some connections between phenomena in the data based on what you already believe to be true or expect to see, while ignoring, disregarding, or simply not noticing other connections that may turn out to be fruitful.

Think about the way a suggestion engine works versus the way human recommendations work. If a friend tells you that they enjoyed the 2019 movie Parasite, you might recommend other movies based on your interpretation of their taste in cinema. For example, you might suggest they watch a different Bong Joon-ho movie, like Snowpiercer or Okja. You might suggest they watch another South Korean classic, like Old Boy. Or perhaps you think that this means they're interested in movies that present a social commentary, so you suggest Joker.

Without asking them what they liked about the film, though, you'd run out of suggestions pretty quickly — and anyway, you've only seen so many movies yourself. If they searched Netflix for Parasite, on the other hand, the system would suggest most of these, too, but would also suggest hundreds of other titles that people who liked Parasite also watched, or that share a feature with Parasite but also with other movies you've rated highly, or that have something else in common with Parasite that would never have occurred to you as a feature at all.

This is the trouble with manual feature engineering. It's limited by what the data scientist deems noteworthy or relevant or what they personally consider to be a predictor. Features are limited to the scope of that one person's creativity, expertise, scope, and bandwidth. Simply put, you use the things you know, but ignore your unknown unknowns, which means that you may be missing out on potentially great features simply because you never thought to look for them.

Features are limited to the scope of that one person's creativity, expertise, scope, and bandwidth. Simply put, you use the things you know, but ignore your unknown unknowns.

How does it work?

Manual feature engineering means you have to look at all the data you have and then think up all the possible combinations of columns and predictors that could give you the answers you need. You also need to test each individual feature that you come up with to see how it connects. This can take weeks on end.

This might not be too much of a problem when you're running a small project or building an individual model, but it isn't really scalable. When you try to translate this approach into a huge project, you'll quickly run into issues. Trying to do feature engineering by hand for multiple models and complex projects will quickly become overwhelming.

Automated feature engineering

On the other hand, automated feature engineering produces huge numbers of options based on every correlation the system can find. It works in a similar way to manual feature engineering, but it extends the concept to as many connections as possible.

Let's go back to the above example. Your friend loved the film Parasite, and is looking for new recommendations. You've hit a wall on the recommendation front, so you took what you know, and added a whole new set of data and variables to the equation.

You might have looked into weather forecasts and records of the day your friend watched the movie, as well as socioeconomic information about the area where they live, and maybe even looked at social media interactions around the movie and people who claimed an interest in it.

From here, automated feature engineering tools will take over to combine these data sources looking for any possible correlation that could be relevant. This isn't limited specifically to the movie itself, or to similar interests, but to any possible connection that could yield a predictive outcome.

For instance, the feature engineering platform could discover that more people claimed to like the movie on days when the sky was cloudy, or when there was heavy rain. Similarly, it could discover that people in lower socioeconomic areas found the movie interesting. It could find that social media chatter about the movie coincided with large scale protests, or even with seemingly unrelated events such as the Olympics or the World Cup. No matter how seemingly unrelated, automated feature engineering will explore a nearly endless number of combinations to find possible predictive features.

Inevitably, some of these will just be coincidences rather than useful predictors. That's where your domain knowledge comes in. You can review the list, select the most interesting and relevant ones, and scrap the ones that seem like nonsense. In the process, you'll come across hundreds of unexpected combinations and connections in your data columns that simply wouldn't have occurred to you otherwise.

As a result, you end up with significantly more feature options than you could possibly have created manually. You can also choose to rank features not only by what works as a predictor but by what works the best and is most relevant to your questions. This gives you scope to experiment with different predictors and features without piling on more workload, helping you get better answers quickly, and reducing your time to deployment.

Training and testing data

The good news is that by this stage in the process, you now have all the data you need to build and train your models. You're no longer looking to find new data or to change your datasets in any radical way.

However, the journey isn't over yet! The next step involves splitting your existing datasets in sophisticated ways in order to train and test those models effectively. Put simply, you now need to separate the data you have into sets that you'll use to train your model and sets you'll use to test it. How you go about this will have a huge impact on the quality of your future results.

Training and testing datasets: what's the difference?

Generally speaking, you want to use the same dataset to test and to train your models. However, you will use different parts of this dataset for these two functions.

Training datasets explained

Training your model is the process of teaching it how to predict an outcome – and the data you feed into it will dictate how it "learns" what features are predictors of which outcomes.

That means your training data is instructing the model to link certain data points, so that it will be able to predict the same connections in similar data, later on.

Testing datasets explained

When your machine learning model has finished training, you need to test that it really has got the hang of this and can make accurate predictions based on what it has learned. This is why your testing data should come from the same dataset, but not actually be the same data used to train the model. Rather, you'll use some leftover data from that training data set to feed into the model and check if it comes up with the kind of predictions you expect.

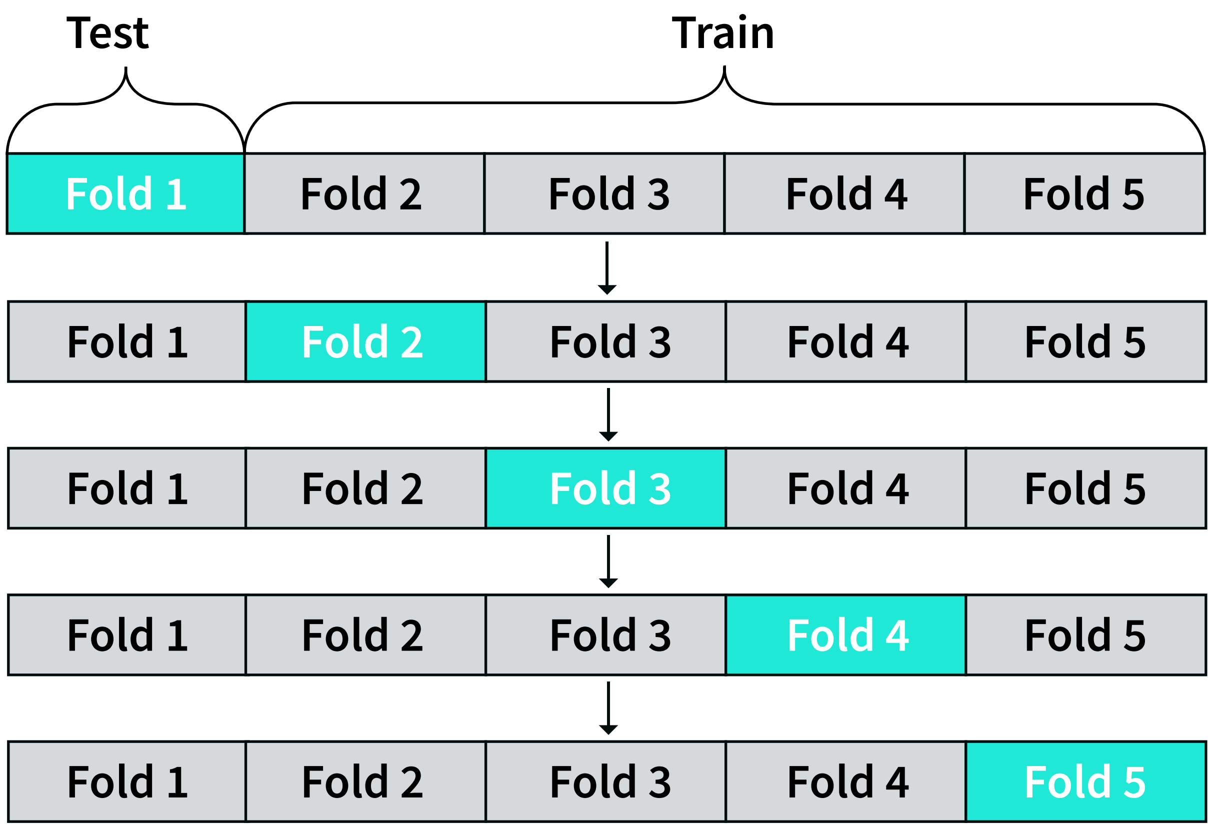
How to split your data

Before you start training and testing, you need to figure out the best way to split the data in the first place. Depending on your dataset's size, you will probably opt for either K-folds or LOOCV, two of the most common and popular methods.

K-folds

This is an approach to splitting data that involves shuffling the dataset randomly and splitting it into subsets or "folds" using all but one of these folds to train the model. "K" here refers to the number of folds/subsets you divide the dataset into. The idea of k-folds cross-validation is to test the skill of an ML model on data it hasn't seen.

The goal of splitting the data like this is to create enough sets to test the model several times and find an average score. Perhaps the biggest advantage of k-folds over other popular methods, though, is that it's fairly simple to deploy and its results are often less biased than other more complex methods.



This would leave us with five separate model scores. Now, all that would be left to do is to calculate the mean accuracy score, as well as the variance score (such as the standard deviation and standard error) to determine how accurate a model is.

Usually, it's best to use five or 10 folds, which have been empirically shown to have lower rates of error and variance, as well as lower bias scores. However, the number of folds you choose also depends on the size of your dataset to ensure that each fold is split evenly.

The good thing about this system is that you can keep reshuffling and splitting up the data as many times as you like, so that the model has plenty of similar but unique training sets and test sets to learn from, always drawn from the same dataset.

When to use k-folds

This is one of the most common methods used to split up data and is best for large datasets, where shuffling the data at random really will give you a completely different fold each time. K-folds remains the most popular and simplest method for cross-validating your models. It also scales well with your datasets, so you don't have to divert computing resources to use it, and it can be accomplished relatively quickly.

Leave One Out Cross Validation (LOOCV)

Unlike k-folds cross-validation, which splits datasets into distinct subsets and uses one as the testing set, LOOCV focuses on the entire dataset, using single points as tests. So instead of having five, 10, or more folds, you'll use the entire dataset to train your model and use an individual observation to test its skill and accuracy.

For example, if you had a dataset of 100 data points, you'd pick a single data point (initially) as a test, and train your model with the remaining 99. You would then repeat this process another 99 times and then calculate the mean skill score for your model.

This makes LOOCV great at some things (such as dealing with smaller datasets), but needlessly resource-heavy and highly unscalable for large datasets. More importantly, perhaps, LOOCV can be subject to high levels of variance and overfitting issues because you're often using too much data to train your model and testing on just a single observation.

When to use LOOCV

This works really well when you're using a small dataset, as taking an average isn't going to overload your system. If you're working with a really big dataset, though, calculating an average across every observation is a huge job and computationally very expensive.

Data training methodologies

It's vital that you choose methods and data techniques that will help you train your models effectively. The methodology you choose depends on what you want your model to do — what problems it solves or questions it answers in order to make a prediction about the data it's fed. Here are some common methodologies you might choose from.

The methodology you choose depends on what you want your model to do — what problems it solves or questions it answers in order to make a prediction about the data it's fed.

Regression

This is a technique used to analyze and model relationships between variables. At its core, regression is used to forecast connections between dependent (what we want to find out) and independent (what we know) variables, as well as predicting causality. There are several types of regression models, and they generally form the basis of most machine learning algorithms.

On a broad level, regressions vary based on the number of independent variables and the relationships between them and dependent variables. Let's take a look at some of the more common types of regression algorithms:

- Linear regressions are based on the standard slope-intercept formula ($y = Mx + b$) and are limited to one independent variable and one dependent variable. More importantly, this model assumes there is a linear relationship between the two.

  However, there are different types of linear regression, including those that can have multiple variables. In this case, our formula will look more like this:

  $f(x,y,z) = w1x + w2y + w3z$

  Where x, y, and z are the different pieces of information we have (our data), while w represents our weights, the data we're trying to learn. Linear regressions are great for simple categorizations since we can set a threshold to measure whether variables meet a relationship or not. This poses problems, though, when we run into more complex relationships than yes or no.

- Logistic regressions are used when our dependent variable is categorical (i.e., it can be more than one thing, such as 1 or 0), giving us a better understanding of these types of relationships. Again, there are multiple types of logistic regressions, including binary (where there are only two possible outcomes): multinomial (where we measure three or more categories without ordering them) and ordinal (where we're looking for the ordering, such as when we're rating things).

Classification

This involves predicting whether a particular data point falls into one class/target/label/category or another. For example, a machine learning algorithm designed to identify nudity in photographs might be trained to classify photographs as "containing nudity" or "not containing nudity".

This type of supervised machine learning uses training data to understand the relationship between input variables and their outcomes to map functions that can predict the desired outputs later on. For example, a spam detector in an email client will scan through a trove of emails that are both spam and not spam to identify those variables that could lead to a more confident classification as spam once the model is in production. There are two general types of classification "learners":

- Lazy learners simply store training data until testing data is presented. At that point, the model will simply scan for the most related data stored to find a working classification. Some common "lazy learners" include k-nearest neighbor models and case-based reasoning algorithms. Generally, these types of algorithms are easier to train, but take a significantly longer time making predictions.
- Eager learners build classification models based on training data before they ever see testing data. This means that they must build a single hypothesis for each instance based on theavailable data. Therefore, they take a much longer time to train, but are much faster when it comes to making confident predictions. Some of the most common eager learners include decision trees, Naive Bayes algorithms, and even artificial neural networks.

Clustering

Clustering involves organizing objects into groups based on shared traits. The model figures out which data points contain relevant similarities and clusters them accordingly. Unlike classification, clustering uses unsupervised learning to identify and group similar data points into larger groups (or "clusters") that are easier to manipulate and understand.

This method is a great introductory and surface-level training method since it can start giving you insights about your data and make it easier for your models to classify data later on. Generally speaking, clustering techniques are more effective if you're starting from a dataset that is unstructured or not as organized. Additionally, it's useful to look for anomalies in your data, which will often pop up as outliers in most clustering algorithms.

Dimensionality reduction

The "curse of dimensionality" kicks in when a dataset has a huge number of features. The more features, the more complex the model will be – and so it makes sense to strip back the features you don't need will be – and so it makes sense to strip back the features you don't need to keep things streamlined. Dimensionality reduction does just that. For example, it might reduce a thousand columns down to a hundred in your dataset.

The term "dimensionality reduction" refers to a variety of techniques rather than a monolithic method. However, they all have the same goal — to give you a feature set that is free of redundancies and white noise that could give you less accurate results or result in overfitting issues.

Generally, dimensionality reduction focuses on the features you have, using both feature engineering and feature extraction to find those that are too similar, too imprecise, or simply not relevant to the predictive question. However, there are other methods to reduce dimensionality, including matrix factorization, which uses algebraic methods to cut down dataset matrices into their constituent parts.

Similarly, you can use manifold learning, which creates a low-dimensional projection of a high-dimensional problem to give you a more pared down and easily visualized dataset. The goal is to create a simpler data set without sacrificing the salient structure or the relevant relationships in the data.

Monitoring and Refreshing

By this point in the process, you may be thinking you can afford to rest on your laurels. You've located the data you need. You've cleaned it up and got it fit for purpose. You've built your models, tested and trained them, and put them into production. Your data tasks are done, right?

Well, unfortunately, the answer is no. You're not done. The fact is, your models aren't static and your business landscape is constantly evolving. That means you need to keep sprucing up your models to reflect this changing context. You need to keep finding good, up-to-date, accurate data to feed your models, too. Otherwise, they'll become diminishingly relevant as time goes by.

The fact is, your models aren't static and your business landscape is constantly evolving. That means you need to keep sprucing up your models to reflect this changing context.

It's important to understand that data is a perishable good. Data that gives you incredibly useful insights now or that helps you make extremely accurate predictions about what's around the corner probably won't retain that superpower in six months' time. From user preferences to the prices of raw materials, everything you base your predictions on is changing all the time.

This is especially true during tumultuous times when, for example, markets are volatile, customer behaviors are more erratic, new regulations have come into force or a game-changing technology has shaken up the status quo. You may find that you're not even asking the right questions for the challenges of the present.

In these situations, data on the way your business or industry behaved last year won't be a great indicator of what you can expect this year. You need to feed in newer, better data.

Auditing your data in the long haul

In practice, this means that you constantly need to audit your data and your models to ensure they're still giving you accurate results. You should be testing your models on a regular basis, ensuring the data you're using gives you relevant, accurate answers.

"While automation success is possible through either traditional top-down (waterfall) deployment or more flexible agile methods, a systematic approach is key. Only 5 percent of respondents at successful companies say their deployment methods have been ad hoc, compared wit 19 percent of peers not reporting success"

- Mckinsey Research

Automated testing

Testing and retesting your models

manually is inefficient and tricky. You can speed it up dramatically by automating the process, but you will still need to take a close look at the data you feed in to ensure it's still the most accurate, useful, relevant data for that predictive model or set of questions.

For example, let's say your model has been built, trained, and tested using customer data from one particular geographic location. In a year's time, things might have shifted so that this is no longer your primary market. Instead, you now need data on customer behavior or footfall for consumers in a different country or region. It's up to you to stay on top of these issues, ensuring that you're always seeking the most important data sources and feeding them into your model to keep it fresh.

Monitoring your sources

One part of this is continually monitoring your existing data sources. Can you still depend on these? Has the data they collect drifted in a different direction? Is it still as relevant as it once was? Again, automating your connections to external sources will make it far easier for you to monitor the data sources you use and to switch to alternatives if they're no longer what you need.

Building models that last with relevant data

There you have it. Machine learning is only as good as the data you feed your models, but it goes even further than that. You don't just need to find the right data, you need to also understand how to get the best insights out of it, and how to use it to best train your models. Instead of a one-off step, your data workflow continues on even after you've built your models, and even after you've put them into production.

Ignoring data means neglecting the reason your models work, and is a recipe for getting yourself into technical debt. Make sure you're always thinking about how your data impacts your models, whether it's during your feature engineering, during training, or even once you've deployed your models into the real world. By putting data first, you'll ensure that your ML gives you relevant, powerful, and actionable insights.

Did you miss steps one and two? Get the full picture of data for ML by reading **Making Sense of Data: Auditing, Discovery, and Acquisition** and **Making Sense of Data Prep: ETL, Wrangling, Data Enrichment.**

EXPLORIUM

About Explorium

Explorium offers a first of its kind data science platform powered by augmented data discovery and feature engineering. By automatically connecting to thousands of external data sources and leveraging machine learning to distill the most impactful signals, the Explorium platform empowers data scientists and business leaders to drive decision-making by eliminating the barrier to acquire the right data and enabling superior predictive power.

For more information, visit www.explorium.ai