Explore feature engineering

In this reading, you will learn more about what happens in the Analyze stage of PACE—namely, feature engineering. The meaning of the term "feature engineering" can vary broadly, but in this course it includes feature selection, feature transformation, and feature extraction. You will come to understand more about the considerations and process of adjusting your predictor variables to improve model performance.

When building machine learning models, your model is only ever going to be as good as your data. Sometimes, the data you have will not be predictive of your target variable. For example, it's unlikely that you can build a good model that predicts rainfall if you train it on historical stock market data. In this case, it might seem obvious, but when you're building a model, you'll often have features that plausibly could be predictive of your target, but in fact are not. Other times, your model's features might contain a predictive signal for your model, but this signal can be strengthened if you manipulate the feature in a way that makes it more detectable by the model.

Feature engineering is the process of using practical, statistical, and data science knowledge to select, transform, or extract characteristics, properties, and attributes from raw

data. In this reading, you will learn more about these processes, when and why to use them, and what good feature engineering can do for your model.

Feature Selection

Feature selection is the process of picking variables from a dataset that will be used as predictor variables for your model. With very large datasets, there are dozens if not hundreds of features for each observation in the data. Using all of the features in a dataset often doesn't give any performance boost. In fact, it may actually hurt performance by adding complexity and noise to the model. Therefore, choosing the features to use for the model is an important part of the model development process.

Generally, there are three types of features:

1. Predictive: Features that by themselves contain information useful to predict the target

2. Interactive: Features that are not useful by themselves to predict the target variable, but become predictive in conjunction with other features

3. Irrelevant: Features that don't contain any useful information to predict the target

You want predictive features, but a predictive feature can

also be a redundant feature. Redundant features are highly correlated with other features and therefore do not provide the model with any new information—for example, the steps you took in a day, may be highly correlated with the calories you burned. The goal of feature selection is to find the predictive and interactive features and exclude redundant and irrelevant features.

The feature selection process typically occurs at multiple stages of the PACE workflow. The first place it occurs is during the Plan phase. Once you have defined your problem and decided on a target variable to predict, you need to find features. Keep in mind that datasets are not always prepackaged in nice little tables ready to model. Data professionals can spend days, weeks, or even months acquiring and assembling features from many different sources.

Feature selection can happen once more during the Analyze phase. Once you do an exploratory data analysis, it might become clear that some of the features you included are not suitable for modeling. This could be for a number of reasons. For example, you might find that a feature has too many missing or clearly erroneous values, or perhaps it's highly correlated with another feature and must be dropped so as not to violate the assumptions of your model. It's also common that the feature is some kind of metadata, such as

an ID number with no inherent meaning. Whatever the case may be, you might want to drop these features.

During the Construct phase, when you are building models, the process of improving your model might include more feature selection. At this point, the objective usually is to find the smallest set of predictive features that still results in good overall model performance. In fact, data professionals will often base final model selection not solely on score, but also on model simplicity and explainability. A model with an R2 of 0.92 and 10 features might get selected over a model with an R2 of 0.94 and 60 features. Models with fewer features are simpler, and simpler models are generally more stable and easier to understand.

When data professionals perform feature selection during the Construct phase, they typically use statistical methodologies to determine which features to keep and which to drop. It could be as simple as ranking the model's feature importances and keeping only the top *a*% of them. Another way of doing it is to keep the top features that account for ≥ b% of the model's predictive signal. There are many different ways of performing feature selection, but they all seek to keep the predictive features and exclude the non-predictive features.

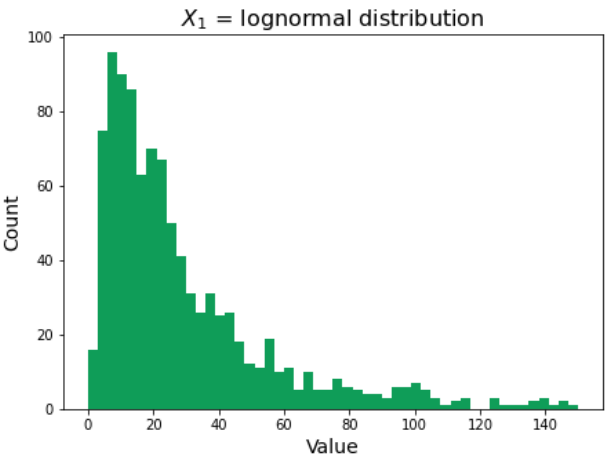
Feature Transformation

Feature transformation is a process where you take features that already exist in the dataset, and alter them so that they're better suited to be used for training the model. Data professionals usually perform feature transformation during the Construct phase, after they've analyzed the data and made decisions about how to transform it based on what they've learned.

Log normalization

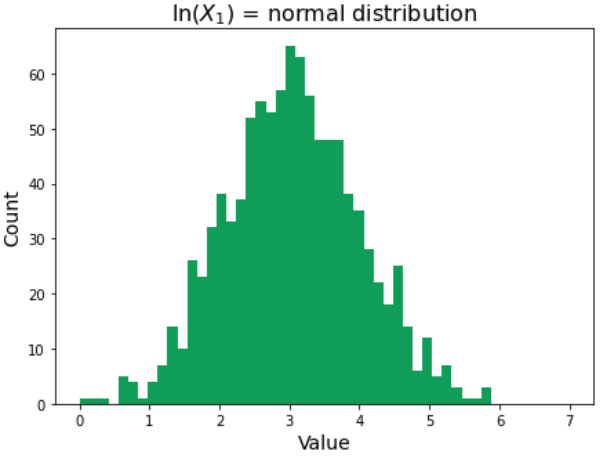
There are various types of transformations that might be required for any given model. For example, some models do not handle continuous variables with skewed distributions very well. As a solution, you can take the log of a skewed feature, reducing the skew and making the data better for modeling. This is known as **log normalization.**

For instance, suppose you had a feature *X*1 whose histogram demonstrated the following distribution:



This is known as a **log-normal distribution.** A log-normal

distribution is a continuous distribution whose logarithm is normally distributed. In this case, the distribution skews right, but if you transform the feature by taking its natural log, it normalizes the distribution:



Normalizing a feature's distribution is often better for training a model, and you can later verify whether or not taking the log has helped by analyzing the model's performance.

Scaling

Another kind of feature transformation is **scaling**. Scaling is when you adjust the range of a feature's values by applying a normalization function to them. Scaling helps prevent features with very large values from having undue influence over a model compared to features with smaller values, but which may be equally important as predictors.

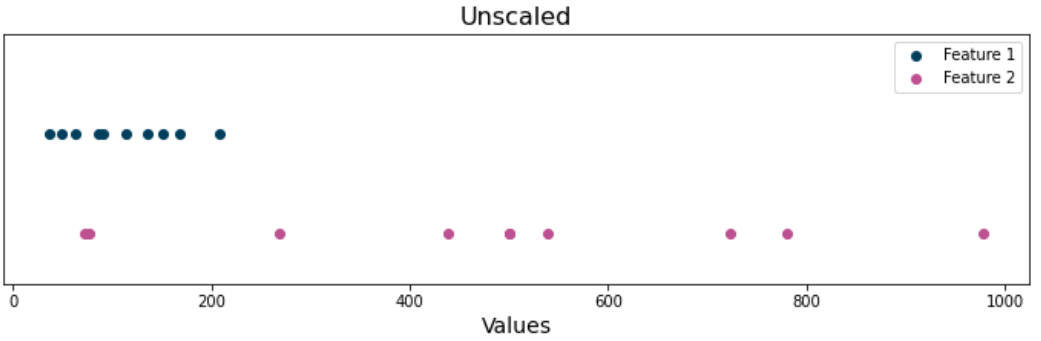
There are many scaling methodologies available. Some of the most common include:

Normalization

Normalization (e.g., **MinMaxScaler** in scikit-learn) transforms data to reassign each value to fall within the range [0, 1]. When applied to a feature, the feature's minimum value becomes zero and its maximum value becomes one. All other values scale to somewhere between them. The formula for this transformation is:

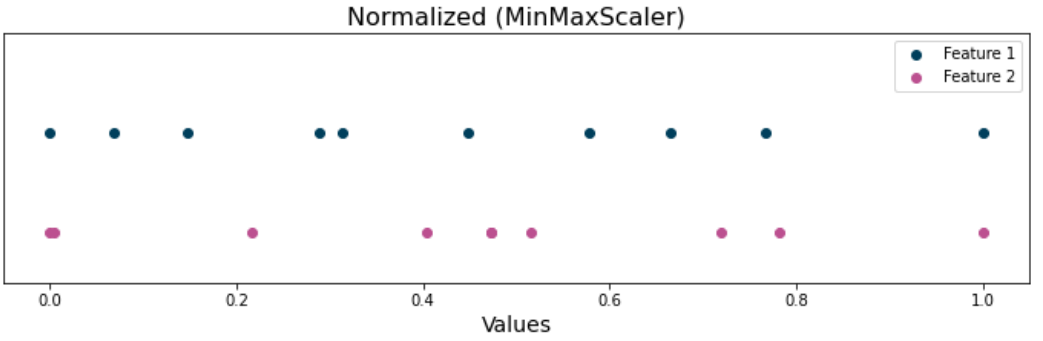
$$x_{i,normalized} = \frac{x_{max} - x_{min}}{x_i - x_{min}}$$

For example, suppose you have feature 1, whose values range from 36 to 209; and feature 2, whose values range from 72 to 978:



It is apparent that these features are on different scales from one another. Features with higher magnitudes of scale will be more influential in some machine learning algorithms, like K-means, where Euclidean distances between data points are calculated with the absolute value of the features (so large feature values have major effects, compared to small

feature values). By min-max scaling (normalizing) each feature, they are both reduced to the same range:

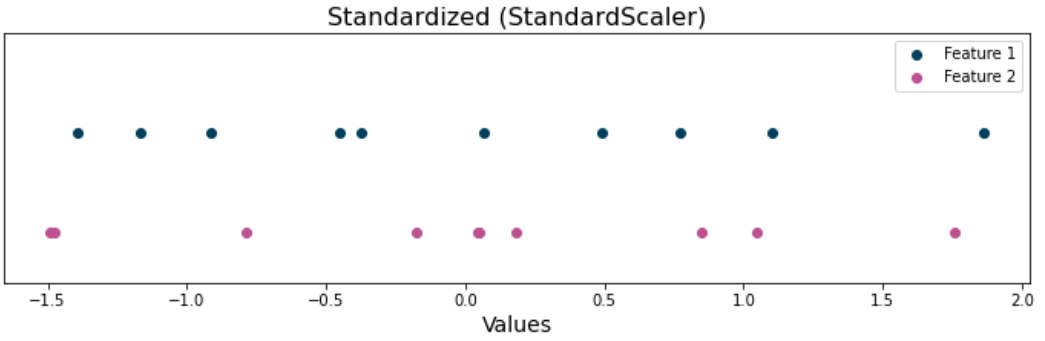


Standardization

Another type of scaling is called **standardization** (e.g., **StandardScaler** in scikit-learn). Standardization transforms each value within a feature so they collectively have a mean of zero and a standard deviation of one. To do this, for each value, subtract the mean of the feature and divide by the feature's standard deviation:

$$xi,standardized = xstand.dev.xi - xmean$$

This method is useful because it centers the feature's values on zero, which is useful for some machine learning algorithms. It also preserves outliers, since it does not place a hard cap on the range of possible values. Here is the same data from above after applying standardization:



Notice that the points are spatially distributed in a way that is very similar to the result of normalizing, but the values and scales are different. In this case, the values now range from -1.49 to 1.76.

Encoding

Another form of feature transformation is known as **encoding**. Variable encoding is the process of converting categorical data to numerical data. Consider the bank churn dataset. The original data has a feature called “Geography”, whose values represent each customer’s country of residence—France, Germany, or Spain. Most machine learning methodologies cannot extract meaning from strings. Encoding transforms the strings to numbers that can be interpreted mathematically.

The “Geography” column contains nominal values, or values that don’t have an inherent order or ranking. As such, the feature would typically be encoded into binary. This process requires that a column be added to represent each possible

class contained within the feature.

Geography	Is France	Is Germany	Is Spain
France	1	0	0
Germany	0	1	0
Spain	0	0	1
France	1	0	0

Tools commonly used to do this include *pandas.get_dummies()* and *OneHotEncoder()*. Often methods drop one of the columns to avoid having redundant information in the dataset . Note that information isn't lost by doing this. If you have this...

Customer	Is France	Is Germany
Antonio García	0	0

... then you know this customer is from Spain!

Keep in mind that some features may be inferred to be numerical by Python or other frameworks but still represent a category. For example, suppose you had a dataset with people assigned to different arbitrary groups: 1, 2, and 3:

Name	Group
Rachel Stein	2
Ahmed Abadi	2

Sid Avery	3
Ha-rin Choi	1

The “Group” column might be encoded as type *int*, but the number is really only representative of a category. Group 3 isn’t two units “greater than” group 1. The groups could just as easily be labeled with colors. In this case, you could first convert the column to a string, and then encode the strings as binary columns. This is a problem that can be solved upstream at the stage of data generation: categorical features (like a group) should not be recorded using a number.

A different kind of encoding can be used for features that contain discrete or ordinal values. This is called ordinal encoding. It is used when the values *do* contain inherent order or ranking. For instance, consider a “Temperature” column that has values of cold, warm, and hot. In this case, ordinal encoding could reassign these classes to 0, 1, and 2.

Temperature	**Temperature (Ordinal encoding)**
cold	0
warm	1
hot	2

This method retains the order or ranking of the classes

relative to one another.

Feature extraction

Feature extraction involves producing new features from existing ones, with the goal of having features that deliver more predictive power to your model. While there is some overlap between extraction and transformation colloquially, the main difference is that a new feature is created from one or more other features rather than simply changing one that already exists.

Consider a feature called “Date of Last Purchase,” which contains information about when a customer last purchased something from the company. Instead of giving the model raw dates, a new feature can be extracted called “Days Since Last Purchase.” This could tell the model how long it has been since a customer has bought something from the company, giving insight into the likelihood that they’ll buy something again in the future. Suppose that today’s date is May 30th, extracting a new feature could look something like this:

Date of Last Purchase	Days Since Last Purchase
May 17th	13
May 29th	1
May 10th	20

May 21st	9	

Features can also be extracted from multiple variables. For example, consider modeling if a customer will return to buy something else. In the data, there are two variables: “Days Since Last Purchase” and “Price of Last Purchase.” A new variable could be created from these by dividing the price by the number of days since the last purchase, creating a new variable altogether.

Days Since Last Purchase	Price of Last Purchase	Dollars Per Day Since Last Purchase
13	$85	$6.54
1	$15	$15.00
20	$8	$0.40
9	$43	$4.78

Sometimes, the features that you are able to generate through extraction can offer the greatest performance boosts to your model. It can be a trial and error process, but finding good features from the raw data is what will make a model stand out in industry.

Key takeaways

- Analyzing the features in a dataset is essential to creating a model that will produce valuable results

- Feature Selection is the process of dropping any and all unnecessary or unwanted features from the dataset
- Feature Transformation is the process of editing features into a form where they're better for training the model
- Feature Extraction is the process of creating brand new features from other features that already exist in the dataset

Resources for more information

- MinMaxScaler documentation: scikit-learn implementation of MinMaxScaler normalization
- StandardScaler documentation: scikit-learn implementation of StandardScaler standardization