Introduction to Feature Engineering

Learn about feature engineering and its role in machine learning.

Introduction

Feature engineering is an integral part of building and implementing machine learning models. In this article, you'll learn about what feature engineering is, why we need it and where it fits in within the machine learning workflow. Additionally, you'll get a brief overview of the many data science tools involved in feature engineering and how they assist data scientists in model diagnostics.

What is feature engineering?

Before we can talk about feature engineering, we have to define what we mean by 'features'. A feature is a measurable property in a dataset, and a feature can be used as an input to a machine learning model. One way to think about features is as predictor variables that go into a model to predict the outcome variable. For example, if we want a model that predicts precipitation at a given place and time, we might use temperature, humidity, month, altitude, etc. as inputs. These are our features.

Often, when presented with a dataset, it might not be clear what features we should use for a specific model (i.e., should we use 'tree density' as an input to our precipitation model?). Similarly, in large datasets, there might be too many features to manually decide which features to use. Some features might be highly correlated with one another, some might not vary much with the outcome variable, and some might be in the wrong form to be a model input and so on. It is not uncommon that a data scientist might not realize any of this until they begin diagnosing a model that is performing poorly.

Feature engineering is a way to address these challenges. It is an umbrella term for the techniques we use to help us make decisions about features in machine learning model implementations.

Why do we need feature engineering?

To paraphrase Tolstoy, "All functional model implementations resemble one another but every dysfunctional model implementation is dysfunctional in its own way."

There are a lot of different ways that a model implementation can be dysfunctional. So the question is: What would make a model implementation functional or dysfunctional? Consider the following four attributes:

1. Performance:

We would like our machine learning model to perform "well" on our data. If it is not able to predict the outcome variable (to a reasonable degree of accuracy) on known data, it would be unwise to use it to predict outcomes on unknown data.

2. Runtime:

Suppose a model has excellent performance but takes a really long time to run. For a data scientist, depending on their available computational resources, such a model might be impractical for production.

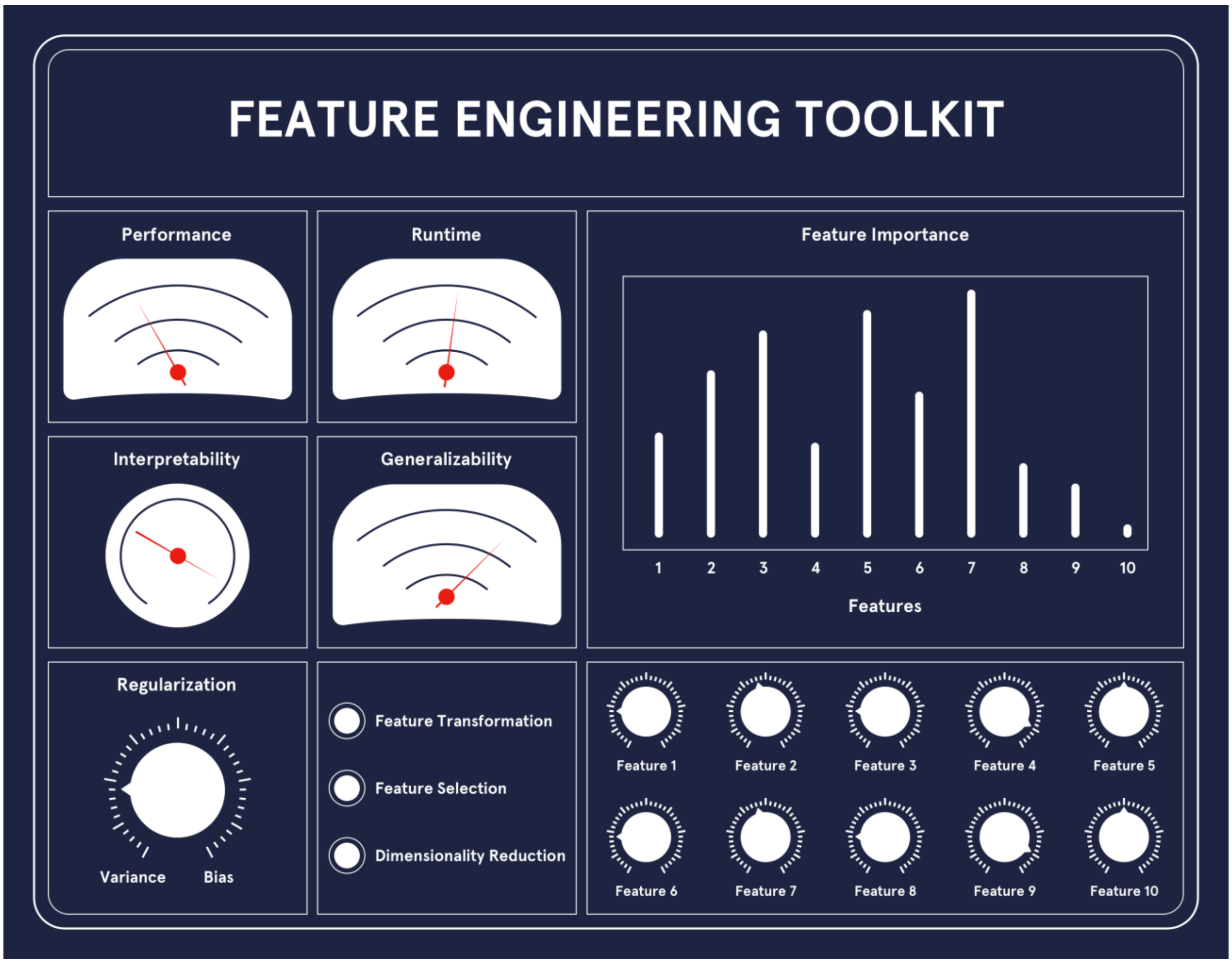
3. Interpretability:

A model is only as good as the insights it helps us glean from the data. Data scientists are often tasked with finding out what factors drive different outcomes. A well-performing model would not be of much help if it's opaque and uninterpretable.

4. Generalizability:

We would like our model to generalize well to unseen data. Often data scientists work with streaming data and need their model to be flexible with new and unknown data..

Feature engineering can be thought of as a toolkit of techniques to use when a model is missing one or more of the above attributes. If we imagine a model diagnostic machine (kind of like a modern version of this one) with meters representing the attributes, feature engineering would be akin to turning knobs and pushing buttons until we arrive upon a model that meets all of our attributes satisfactorily.



Feature Engineering and the Machine Learning Workflow

So where does feature engineering fall within the machine learning workflow? Does it go before modeling, after modeling or alongside modeling? The answer is: all of the above! Feature engineering is often introduced as an intermediate step between exploratory data analysis and implementing a machine learning algorithm. In reality, these distinctions are fuzzy and the process is not exactly linear.

Broadly, we can divide feature engineering techniques into three categories:

1. Feature Transformation methods:

These involve numerical transformations methods and ways to encode non-numerical variables. These techniques are applied *before* implementing a machine learning model. They include and are not limited to: scaling, binning, logarithmic transformations, hashing and one hot encoding. These methods typically improve performance, runtime and interpretability of a model.

2. Dimensionality Reduction methods:

Dimensionality of a dataset refers to the number of features within a dataset and reducing dimensionality allows for faster runtimes and (often) better performance. This is an extremely powerful tool in working with datasets with “high dimensionality”. For instance, a hundred-feature problem can be reduced to less than ten modified features, saving a lot of computational time and resources while maintaining or even improving performance. Typically, dimensionality reduction methods are machine learning algorithms themselves, such as Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), etc.

These techniques transform the existing feature space into a new subset of features that are ordered by decreasing importance. Since they “extract” new features from high dimensional data they’re also referred to as **Feature Extraction** methods. The transformed features do not directly relate to anything in the real world anymore. Rather, they are mathematical objects that are related to the original features. However, these mathematical objects are often difficult to interpret. The lack of interpretability is one of the drawbacks of dimensionality reduction.

3. Feature Selection methods

Feature selection methods are a set of techniques that allow us to choose among the pool of features available. Unlike dimensionality reduction, these retain the features *as they are* which makes them highly interpretable. They usually belong to one of these three categories:

i. Filter methods:

These are statistical techniques used to “filter” out useful features. Filter methods are completely model agnostic (meaning they can be used with any model) and are useful sanity checks for a data scientist to carry out before deciding on a model. They include and are not limited to: correlation coefficients (Pearson, Spearman, etc) , chi^2, ANOVA, and Mutual Information calculations.

ii. Wrapper methods:

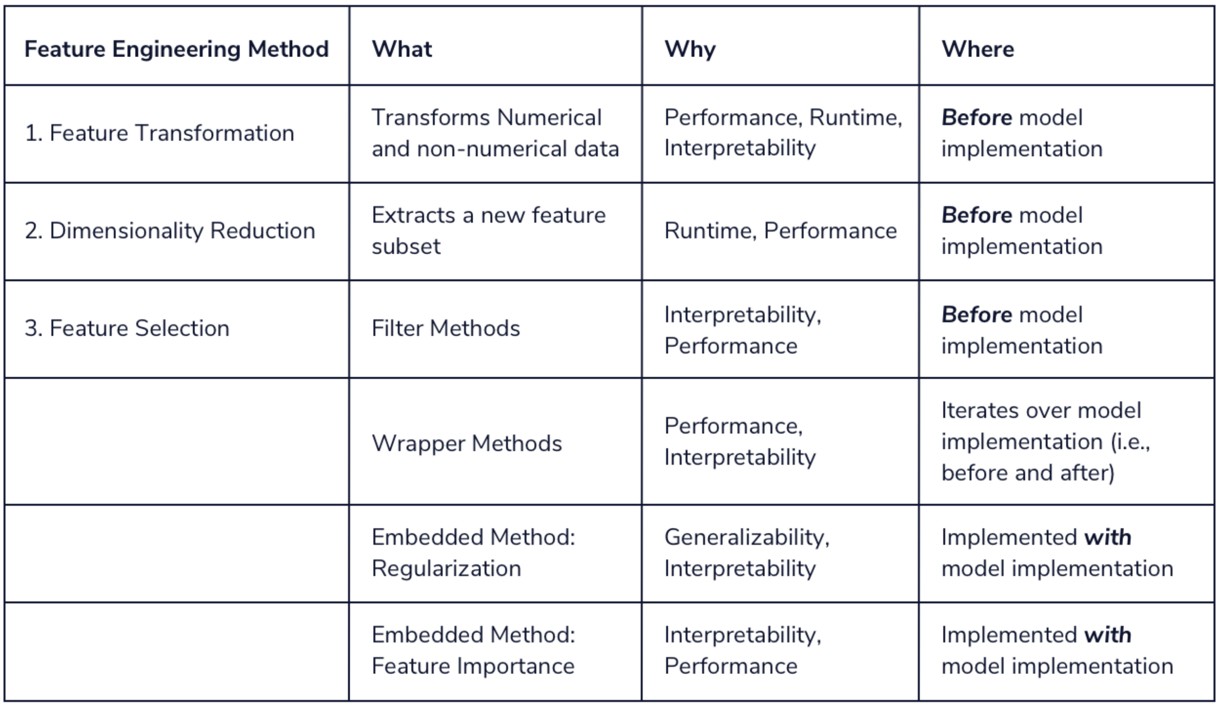
Wrapper methods search for the best set of features by using a “greedy search strategy”. They pick a subset of features, train a model, evaluate it, try a different subset of features, train a model again, and so on until the best possible set of features and most optimal performance is reached. As this could potentially go on forever, a stopping criterion based on number of features or a model performance metric is typically used. Forward Feature Selection, Backward Feature Elimination and Sequential Floating are some examples of wrapper method algorithms.

iii. Embedded methods:

Embedded methods are implemented *during* the model implementation step. Regularization techniques such as Lasso or Ridge tweak the model to get it to generalize better to new and unknown data. Tree-based feature importance is another widely used embedded method. This method provides insight into the features that are most relevant to the outcome variable while fitting decision trees to the data.

Summary

Here’s a brief summary of the feature engineering methods we’ve covered in this article, the attributes they seek to improve, and where they fit in within the machine learning workflow:



Feature Engineering Method	**What**	**Why**	**Where**
1. Feature Transformation	Transforms Numerical and non-numerical data	Performance, Runtime, Interpretability	***Before*** model implementation
2. Dimensionality Reduction	Extracts a new feature subset	Runtime, Performance	***Before*** model implementation
3. Feature Selection	Filter Methods	Interpretability, Performance	***Before*** model implementation
	Wrapper Methods	Performance, Interpretability	Iterates over model implementation (i.e., before and after)
	Embedded Method: Regularization	Generalizability, Interpretability	Implemented ***with*** model implementation
	Embedded Method: Feature Importance	Interpretability, Performance	Implemented ***with*** model implementation

We're now ready to learn about these in-depth!