Feature Engineering: What Powers Machine Learning

How to Extract Features from Raw Data for Machine Learning

Will Koehrsen Nov 12, 2018 · 8 min read

This is the third in a four-part series on how we approach machine learning at Feature Labs. The complete set of articles is:

1. Overview: A General-Purpose Framework for Machine Learning
2. Prediction Engineering: How to Set Up Your Machine Learning Problem
3. Feature Engineering (this article)
4. Modeling: Teaching an Algorithm to Make Predictions

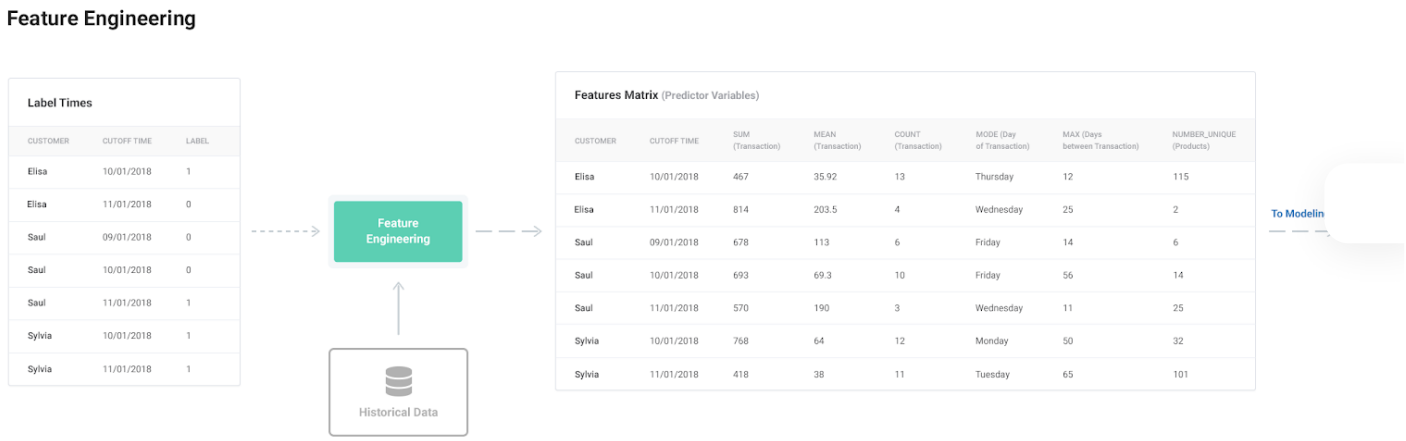
These articles cover the concepts and a full implementation as applied to predicting customer churn. The project Jupyter Notebooks are all available on GitHub. (Full disclosure: I work for Feature Labs, a startup developing tooling, including Featuretools, for solving problems with machine learning. All of the work documented here was completed with open-source tools and data.)

Feature Engineering

It's often said that "data is the fuel of machine learning." This isn't quite true: data is like the *crude oil* of machine learning which means it has to be refined into *features* — predictor variables — to be useful for training a model. Without relevant features, you can't train an accurate model, no matter how complex the machine learning algorithm. The process of extracting features from a raw dataset is called *feature engineering*.

The Feature Engineering Process

Feature engineering, the second step in the machine learning pipeline, takes in the label times from the first step — prediction engineering — and a raw dataset that needs to be refined. Feature engineering means building features for each label while *filtering the data used for the feature based on the label's cutoff time* to make valid features. These features and labels are then passed to modeling where they will be used for training a machine learning algorithm.



The process of feature engineering.

While feature engineering requires label times, in our general-purpose framework, it is *not hard-coded* for specific labels corresponding to only one prediction problem. If we wrote our feature engineering code for a single problem — as feature engineering is traditionally approached — then we would have to redo this laborious step every time the parameters change.

Instead, we use APIs like Featuretools that can build features for *any set of labels without requiring changes to the code*. This means for the customer churn dataset, we can solve multiple prediction problems — predicting churn every month, every other week, or with a lead time of two rather than one month — using the exact same feature engineering code.

> *This fits with the principles of our machine learning approach: we* **segment** *each step of the pipeline while standardizing inputs and outputs. This independence means we can change the problem in prediction engineering without needing to alter the downstream feature engineering and machine learning code.*

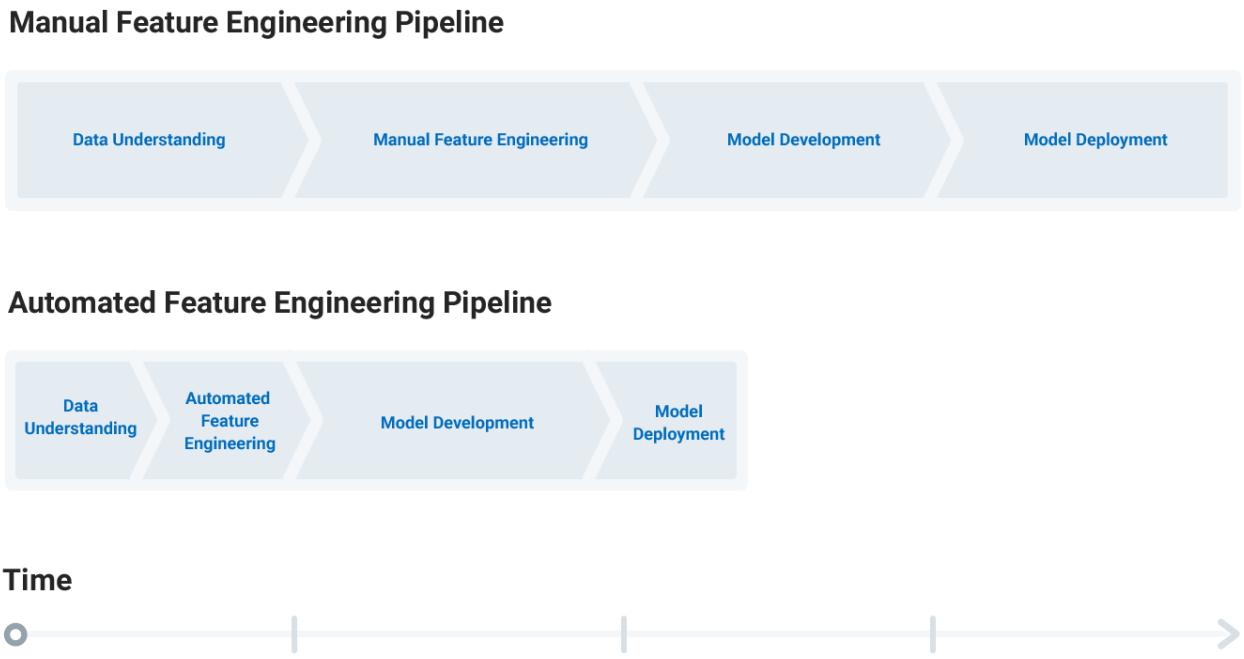
The key to making this step of the machine learning process repeatable across prediction problems is *automated feature engineering*.

Automated Feature Engineering: Build Better Predictive Models

Faster

Traditionally, feature engineering is done by hand, building features one at a time using domain knowledge. However, this manual process is *error-prone, tedious, must be started from scratch for each dataset*, and ultimately is *limited by constraints on human creativity* and *time*. Furthermore, in time-dependent problems where we have to filter every feature based on a cutoff time, it's hard to avoid errors that can invalidate an entire machine learning solution.

Automated feature engineering overcomes these problems through a reusable approach to automatically building hundreds of relevant features from a relational dataset. Moreover, this method filters the features for each label based on the cutoff time, creating a rich set of valid features. In short, automated feature engineering enables data scientists to **build better predictive models in a fraction of the time.**



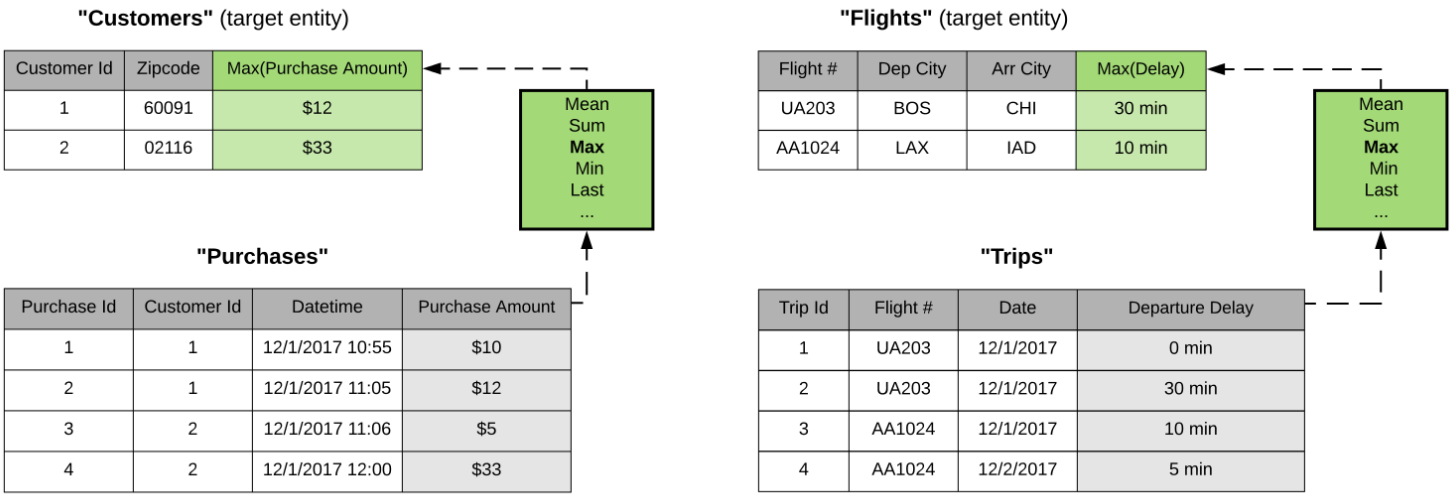
Manual vs Automated Feature Engineering Pipelines.

Motivation for Automated Feature Engineering

After solving a few problems with machine learning, it becomes clear that many of the operations used to build features are repeated across datasets. For instance, we often find the weekday of an event — be it a transaction or a flight— and then find the

average transaction amount or flight delay by day of the week for each customer or airline. Once we realize that these operations *don't depend on the underlying data*, why not **abstract** this process into a framework that can build features for any relational dataset?

This is the idea behind automated feature engineering. We can apply the same basic building blocks — called feature primitives — to different relational datasets to build predictor variables. As a concrete example, the "max" feature primitive applied to customer transactions can also be applied to flight delays. In the former case, this will find the *largest transaction for each customer*, and in the latter, the *longest flight delay for a given flight number*.



To calculate a customer's most expensive purchase, we apply the **`Max`** primitive to the purchase amount field in all related purchases. When we perform the same steps to a dataset of airplane flights, we calculate "the longest flight delay".

Source: How Deep Feature Synthesis Works

This is an embodiment of the idea of abstraction: remove the need to deal with the details — writing specific code for each dataset — by building higher level tools that take advantage of operations common to many problems.

> *Ultimately, automated feature engineering makes us more efficient as data scientists by removing the need to repeat tedious operations across problems.*

Implementation of Feature Engineering

Currently, the only open-source Python library for automated feature engineering using multiple tables is Featuretools, developed and maintained by Feature Labs. For the customer churn problem, we can use Featuretools to quickly build features for the label times that we created in prediction engineering. (Full code available in this Jupyter Notebook).

We have three tables of data: customer background info, transactions, and user listening logs. If we were using manual feature engineering, we'd brainstorm and build features by hand, such as *the average value of a customer's transactions*, or her *total spending on weekends in the previous year*. For each feature, we'd first have to filter the data to *before the cutoff time* for the label. In contrast, in our framework, we make use of Featuretools to automatically build hundreds of relevant features in a few lines of code.

We won't go through the details of Featuretools, but the heart of the library is an algorithm called Deep Feature Synthesis which stacks the feature engineering building blocks known as primitives (simple operations like "max" or finding the "weekday" of a transaction) to build "deep features". The library also automatically filters data for features based on the cutoff time.

For more on automated feature engineering in Featuretools see:

- Automated Feature Engineering in Python
- Featuretools documentation and GitHub

Featuretools requires some background code to link together the tables through relationships, but then we can automatically make features for customer churn using the following code (see notebook for complete details):

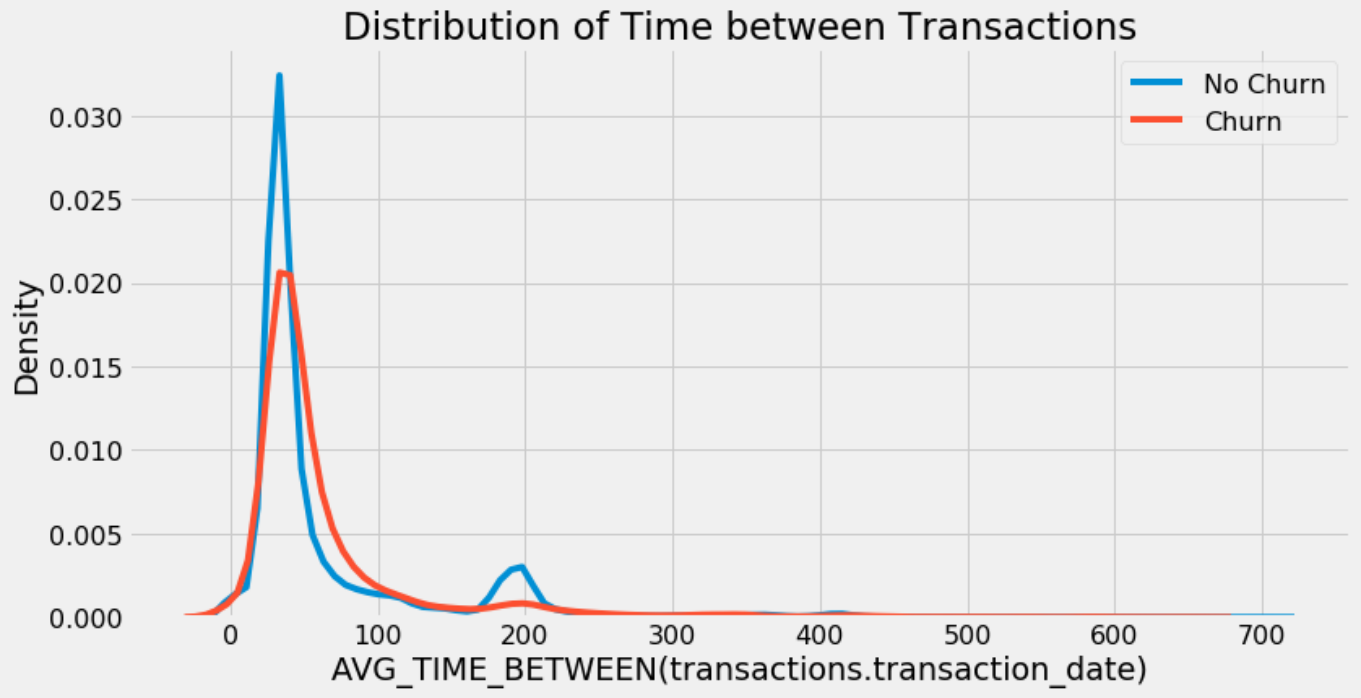
This one line of code gives us over 200 features for each label in `cutoff_times`. Each feature is a combination of feature primitives and is built with *only data from before the associated cutoff time*.

customer_id	cutoff_time	SUM(transactions.planned_daily_price WHERE is_auto_renew = 0)	AVG_TIME_BETWEEN(transactions.transaction_date)	MAX(logs.total_secs)	label
0	2016-02-01	9.933333	39.000000	11655.381	1.0
0	2016-06-01	9.933333	39.000000	11655.381	0.0
1	2015-03-01	9.933333	28.000000	14500.406	0.0
1	2015-04-01	14.900000	32.000000	14500.406	0.0
1	2015-05-01	19.866667	30.333333	41133.769	0.0

Sample of features from Featuretools automated feature engineering.

The features built by Featuretools are explainable in *natural language* because they are

built up from basic operations. For example, we see the feature `AVG_TIME_BETWEEN(transactions.transaction_date)`. This represents the average time between transactions for each customer. When we plot this colored by the label we see that customers who churned appear to have a slightly longer average time between transactions.



Distribution of time between transactions colored by the label.

In addition to getting hundreds of valid, relevant features, developing an automated feature engineering pipeline in Featuretools means we can *use the same code for different prediction problems* with our dataset. We just need to pass in the correct label times to the `cutoff_times` parameter and we'll be able to build features for a different prediction problem.

> *Automated feature engineering means we can solve multiple problems in the time it would normally take to complete just one. A change in parameters means tweaking a few lines of code instead of implementing an entirely new solution.*

To solve a different problem, rather than rewrite the entire pipeline, we:

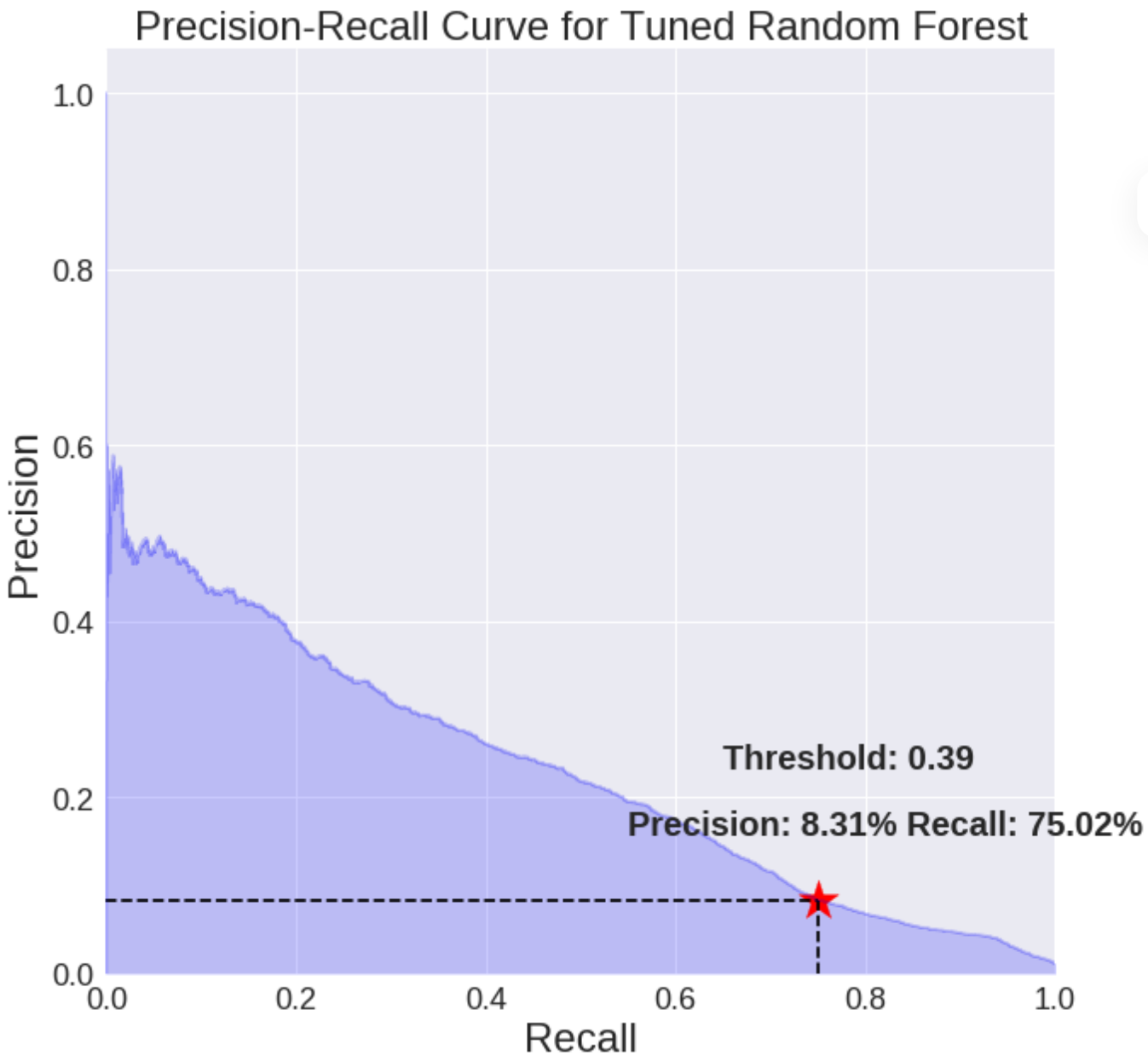
1. Tweak the prediction engineering code to create new label times
2. Input the label times to feature engineering and output features
3. Use the features to train and a supervised machine learning model

(As a brief note: the feature engineering code can be run in parallel using either Dask or Spark with PySpark. For the latter approach, see this notebook or this article on the Feature Labs engineering blog.)

Next Steps

Just as the label times from prediction engineering flowed into feature engineering, the features serve as inputs to the next stage, modeling: training an algorithm to predict the label from the features. In the final article in this series, we'll look at how to train, tune, validate, and predict with a machine learning model to solve the customer churn problem.

As a preview, pictured is the tuned precision-recall curve from machine learning. (Full notebook available on GitHub.)



Precision Recall Curve for Machine Learning

Conclusions

Feature engineering has tended to be a tedious aspect of solving problems with machine learning and a source of errors preventing solutions from being successfully implemented. By using automated feature engineering in a general-purpose machine learning framework we:

- **Automatically build hundreds of features for any relational dataset**

- **Create only valid features by filtering data on cutoff times**

Furthermore, the *feature engineering code is not hard-coded for the inputs from prediction engineering* which means *we can use the same exact code to make features for multiple prediction problems*. Applying automated feature engineering in a structured framework we are able to turn feature engineering from a painful process into a quick, reusable procedure allowing us to solve many valuable machine learning problems.

If building meaningful, high-performance predictive models is something you care about, then get in touch with us at Feature Labs. While this project was completed with the open-source Featuretools, the commercial product offers additional tools and support for creating machine learning solutions.

Thanks to Max Kanter.

Machine Learning Data Science Education Business Predictive Analytics