ENGINEERING

Deep Feature Synthesis: How Automated Feature Engineering Works

Max Kanter

Jan 18, 2018 • 7 min read

The artificial intelligence market is fueled by the potential to use data to change the world. While many organizations have already successfully adapted to this paradigm, applying machine learning to new problems is still challenging.

The single biggest technical hurdle that machine learning algorithms must overcome is their need for processed data in order to work — they can only make predictions from numeric data. This data is composed of relevant variables, known as "features." If the calculated features don't clearly expose the predictive signals, no amount of tuning can

take a model to the next level. The process for extracting these numeric features is called "feature engineering."

Automating feature engineering optimizes the process of building and deploying accurate machine learning models by handling necessary but tedious tasks so data scientists can focus more on other important steps. Below are the basic concepts behind an automated feature engineering method called Deep Feature Synthesis (DFS), which generates many of the same features that a human data scientist would create.

Invented at MIT

DFS was conceived in MIT's Computer Science and Artificial Intelligence Lab in 2014. Our founders, Kalyan Veeramachaneni and Max Kanter, used DFS to create the "Data Science Machine" to automatically build predictive models for complex, multi-table datasets. They used this system to compete in online data science competitions and beat 615 out of 906 human teams.

They first shared this work in a peer-reviewed paper during IEEE's International Conference on Data Science and Advanced Analytics in 2015. Since then, it has now matured to not only power Feature Labs' products, but also motivate & enable researchers around the world, including those at Berkeley and IBM.

Understanding Deep Feature Synthesis

There are three key concepts in understanding Deep Feature Synthesis:

1. Features are derived from relationships between the data points in a dataset.

DFS performs feature engineering for multi-table and transactional datasets commonly found in databases or log files. We focus on this type of data because it is the most common type of enterprise data used today: a survey of 16,000 data scientists on Kaggle found that they spent 65% of their time using relational datasets.

2. Across datasets, many features are derived by using similar

mathematical operations.

To understand this, let's consider a dataset of customers and all of their purchases. For each customer, we may want to calculate a feature representing their most expensive purchase. To do this, we would collect all the transactions related to a customer and find the Maximum of the purchase amount field. However, imagine that we did this for a dataset comprised of airplane flights. If we applied Maximum to a numerical column in this scenario, it could calculate "the longest flight delay," which could predict the potential for delays in the future.

Even though the natural language descriptions differ completely, the underlying math remains the same. In both of these cases, we applied the same operation to a list of numeric values to produce a new numeric feature that was specific to the dataset. These dataset-agnostic operations are called "primitives."

3. New features are often composed from utilizing previously derived features.

Primitives are the building blocks of DFS. Because primitives define their input and output types, we can stack them to construct complex features that mimic the ones that humans create today.

DFS can apply primitives across relationships between entities, so features can be created from datasets with many tables. We can control the complexity of the features we create by setting a maximum depth for our search.

Consider a feature which is often calculated by data scientists for transactional or event log data: "the average time between events." This feature is valuable in predicting either fraudulent behavior or future customer engagement. DFS achieves the same feature by stacking two primitives, Time Since and Mean.

This example highlights a second advantage of primitives, which is that they can be used to quickly enumerate many interesting features in a parameterized fashion. So instead of Mean, we could use Maximum, Minimum , Standard Deviation, or Median to automatically generate several different ways of summarizing the time since the previous event. If we were to add a new primitive to DFS — like the distance between two locations — it would automatically combine with the existing primitives without any effort needed from the user.

Constantly Improving

Back in September, we announced that we were open-sourcing an implementation of DFS for both veteran and aspiring data scientists to try out. In the three months since then, Featuretools has become the most popular library for feature engineering on Github.

This means that a community of people can join together to contribute primitives from which everyone can benefit. Since primitives are defined independently of a specific dataset, any new primitive added to Featuretools can be incorporated into any other dataset that contains the same variable data types. In some cases, this might be a dataset in the same domain, but it could also be for a completely different use case. As an example, here is a contribution of 2 primitives to handle free text fields.

Handling Time

It's easy to accidentally leak information about what you're trying to predict into a model. One of our previous retail enterprise customer's applications is a great example: production models didn't match the company's development results. They were trying to predict who would become a customer, and the most important feature in their model was the number of emails that their prospects had opened. It showed high accuracy during training but unfortunately didn't work when it was deployed into production.

In retrospect, the reason for this is readily apparent — these prospects only started reading emails after becoming customers. The company's **manual feature engineering** step

wasn't properly filtering out the data they had received after the outcome they were predicting had already come true.

DFS in Featuretools can automatically calculate the features for each training example at the specific time associated with the example by using "cutoff times." It accomplishes this by simulating what the raw data looked like at a past point in time in order to perform feature engineering on the valid data. This leads to fewer label leakage problems, which helps data scientists become more confident about the results they are deploying into production.

Augmenting the Human Touch with Automation

DFS can be used to develop baseline models with little human intervention. We have shown how this is possible in public demos using Featuretools. However, the automation of feature engineering should be thought of as a complement to critical human expertise — it enables data scientists to be more precise and productive.

For many problems, a baseline score is enough for a human to decide if an approach is valid. In one case, we ran an experiment against 1,257 human competitors on Kaggle. We produced feature matrices using DFS and then utilized a regressor in order to create a machine learning model.

The machine learning score (RSME) vs. the percentile on the leaderboard. As the score goes down, the place on the leaderboard goes up. The colored vertical lines represent the leaderboard position of different experiments using Featuretools.

We found that with almost no human input, DFS outperforms both baseline models in this prediction problem. In a real-world setting, this is valuable supporting evidence for leveraging machine learning in this use case. Next, we showed how adding custom primitives can be used to outperform more than 80% of our competitors and get close to the best overall score.

Applying Deep Feature Synthesis

We recently wrote about how automated feature engineering was used to increase revenue

for a global bank's fraud detection model. In that case, we were predicting if an individual transaction was fraudulent, but we created features based on historical behaviors of the customer who made the transaction. DFS created features such as "the time since the last transaction," "the average time between transactions," and "the last country in which this card was used." All of these features depend on the relationships between the various data points and required using cutoff time to make sure only behavior from before the fraudulent event was used.

As a result, the number of false positives dropped by 54% compared to the bank's existing software solution, thereby shrinking the number of customers affected by incorrectly blocked transactions. The financial impact of the new model was estimated to be €190,000 per 2 million transactions evaluated.

Deep Feature Synthesis vs. Deep Learning

Deep Learning automates feature engineering for images, text, and audio where a large training set is typically required, whereas DFS targets **the structured transactional and relational datasets** that companies work with.

The features that DFS generates are more explainable to humans because they are based on combinations of primitives that are easily described in natural language. The transformations in deep learning must be possible through matrix multiplication, while the primitives in DFS can be mapped to any function that a domain expert can describe. This increases the accessibility of the technology and offers more opportunities for those who are not experienced machine learning professionals to contribute their own expertise.

Additionally, while deep learning often requires many training examples to train the complex architectures it needs to work, DFS can start creating potential features based only on the schema of a dataset. For many enterprise use cases, enough training examples for deep learning are not available. DFS offers a way to begin creating interpretable features for smaller datasets that humans can manually validate.

A Better Future with Feature Engineering

Automating feature engineering offers the potential to accelerate the process of applying

machine learning to the valuable datasets collected by data science teams today. It will help data scientists to quickly address new problems as they arise and, more importantly, make it easier for those new to data science to develop the skills necessary to apply their own domain expertise.

Renowned machine learning professor Pedros Domingos once said, “One of the holy grails of machine learning is to automate more and more of the feature engineering process.” We agree wholeheartedly, and we couldn’t be more excited to work at the forefront of this field!

Learn more about EvalML in this podcast

Recently, Angela Lin and Jeremy Shah participated in a Podcast.__init__ episode where they talked about EvalML, an automated machine learning library written in Python. They describe what automated machine learning really is and talk about the use cases EvalML solves, as well ...

Tyler Heinl
Sep 8, 2021 • 1 min read