Feature Engineering

Course Summary and Key Takeaways

Learning Objectives

- Use the Vertex AI Feature Store
- Describe how to move from raw data to features
- Perform Feature Engineering in BigQuery ML and Keras
- Preprocess features using Apache Beam and Cloud DataFlow
- Use tf.Transform

Module Breakdown

- Module 0: Introduction
- Module 1: Introduction to Vertex AI Feature Store
- Module 2: Raw Data to Features
- Module 3: Feature Engineering
- Module 4: Preprocessing and Feature Creation
- Module 5: Feature Crosses - TensorFlow Playground
- Module 6: Introduction to TensorFlow Transform

Summary

Want to know about **Vertex AI Feature Store**? Want to know how you can improve the accuracy of your ML models? What about how to find which data columns make the most useful features? Welcome to Feature Engineering, where we discuss good versus bad features and how you can preprocess and transform them for optimal use in your models. This course includes content and labs on feature engineering using BigQuery ML, Keras, and TensorFlow.

Key takeaways

Module 1: Introduction to Vertex AI Feature Store

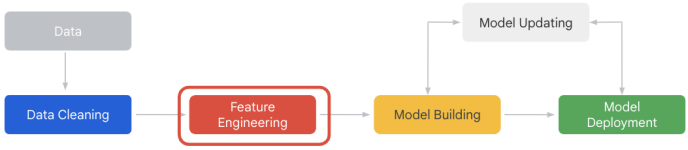
Vertex AI Feature Store solves three key challenges that come up often for people working with ML features:

- Sharing and reuse
- Reliably serving in production with low latency
- Inadvertent skew in feature values between training and serving

---

Module 2: Raw Data to Features

Predictive models are constructed using supervised learning algorithms where classification or regression models are trained on historical data to predict future outcomes.



Feature engineering is a crucial step in the process of **predictive modeling**. It involves the transformation of a given feature, with the objective of reducing the modeling error for a given target.

> **Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data.**
>
> Prof. Andrew Ng

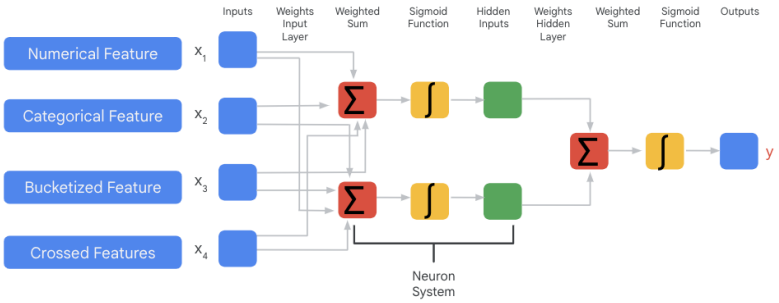
Feature engineering can be defined as a process that attempts to create additional relevant features from the existing raw features in the data and to increase the predictive power of the learning algorithm.

It can also be defined as the process of combining domain knowledge, intuition, and data science skill sets to create features that make the models train faster and provide more accurate predictions.

Machine learning models, such as neural networks, accept a **feature vector** and provide a prediction. These models learn in a supervised fashion where a set of feature vectors with expected output is provided.

From a practical perspective, many machine learning models must represent the features as real-numbered vectors because the feature values must be multiplied by the model weights. In some cases, the data is raw and must be transformed into feature vectors.

Features, the columns of your dataframe, are key in assisting machine learning models to learn. Better features result in faster training and more accurate predictions. As the diagram shows, feature columns are input into the model—not as raw data, but as feature columns.



Module 3: Feature Engineering

Engineering new features from a provided feature set is a common practice. Such engineered features will either augment or replace portions of the existing feature vector. These engineered features are essentially calculated fields based on the values of the other features. Feature vectors can be numerical, categorical, bucketized, crossed, and hashed.

BigQuery preprocessing involves two aspects:

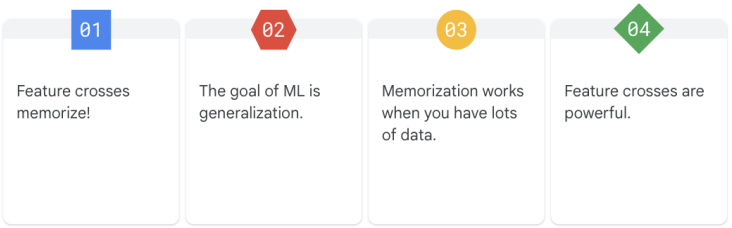
1. Representation transformation
2. Feature construction

Feature representation is converting a numeric feature to a categorical feature (through **bucketization**), and converting categorical features to a numeric representation (through **one-hot encoding**, learning with counts, **sparse feature embeddings**, and so on).

Feature construction is creating new features either by using typical techniques, such as polynomial expansion (by using univariate mathematical functions) or feature crossing (to capture feature interactions).

Features can also be constructed by using business logic from the domain of the ML use case.

Feature crosses are about memorization. Memorization is the opposite of generalization, which is what machine learning aims to do.

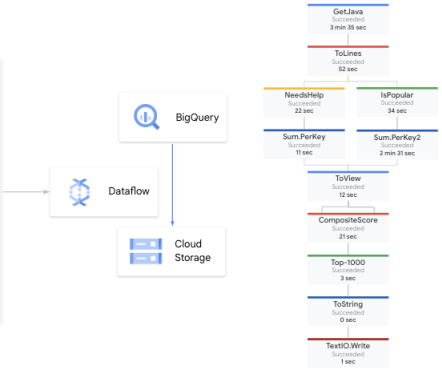


You will find feature crosses extremely useful in real-world datasets. Larger data allows you to make your boxes smaller, and you can memorize more finely. Feature crosses are a powerful feature preprocessing technique on large datasets.

Module 4: Preprocessing and Feature Creation

One way to think about feature preprocessing or even any data transformation is to think in terms of pipelines. So suppose you have some data in a data warehouse, like BigQuery, then you can use BigQuery as an input to your pipeline, do a sequence of steps to transform the data, maybe introduce some new features as part of the transformation. Finally you can save the result to an output like Cloud Storage.

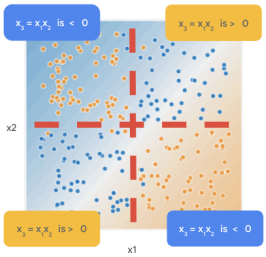
```
def packageHelp(record, keyword):
   count=0
   package_name=``
   if record is not None:
      lines=record.split(`\n`)
      for line in lines:
         if line.startswith(keyword):
            package_name=line
         if `FIXME` in line or `TODO` in line
            count+=1
      packages = (getPackages(package_name)
      for p in packages:
          yield (p, count)
```



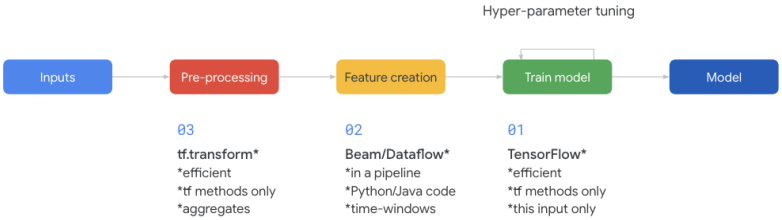
Dataflow is a platform that allows you to run these kinds of data processing pipelines. Dataflow can run pipelines written in Python and Java programming languages. Dataflow sets itself apart as a platform for data transformations because it is a serverless, fully managed offering from Google that allows you to execute data processing pipelines at scale.

Module 5: Feature Crosses - TensorFlow Playground

This module reviews feature crosses in **TensorFlow Playground**.



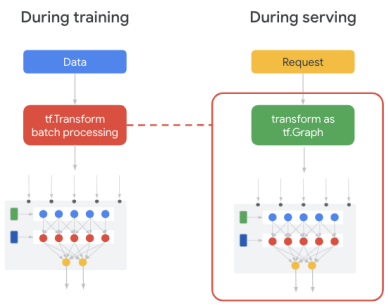
Module 6: An Introduction to TensorFlow Transform



There are three possible places to do feature engineering. Each of which has its pros and cons.

With **TensorFlow Transform**, you are limited to tf methods, but then you get the efficiency of TensorFlow. You can also use the aggregate of your entire training dataset because tf.transform uses Dataflow during training, but only TensorFlow during prediction.

Transform does batch processing, but also emits a TF graph that can be used to "repeat" these transformations in serving.



By combining this graph with the trained model graph, into a single serving graph, you can guarantee that the same operations that were done to the training data, will be done to the request during serving, before passing the transformed result to the trained model graph.