Feature Engineering, Transformation and Selection

DeepLearning.AI

Welcome

DeepLearning.AI

Feature Engineering

Introduction to Preprocessing

"Coming up with features is difficult, time-consuming, and requires expert knowledge. Applied machine learning often requires careful engineering of the features and dataset."

— Andrew Ng

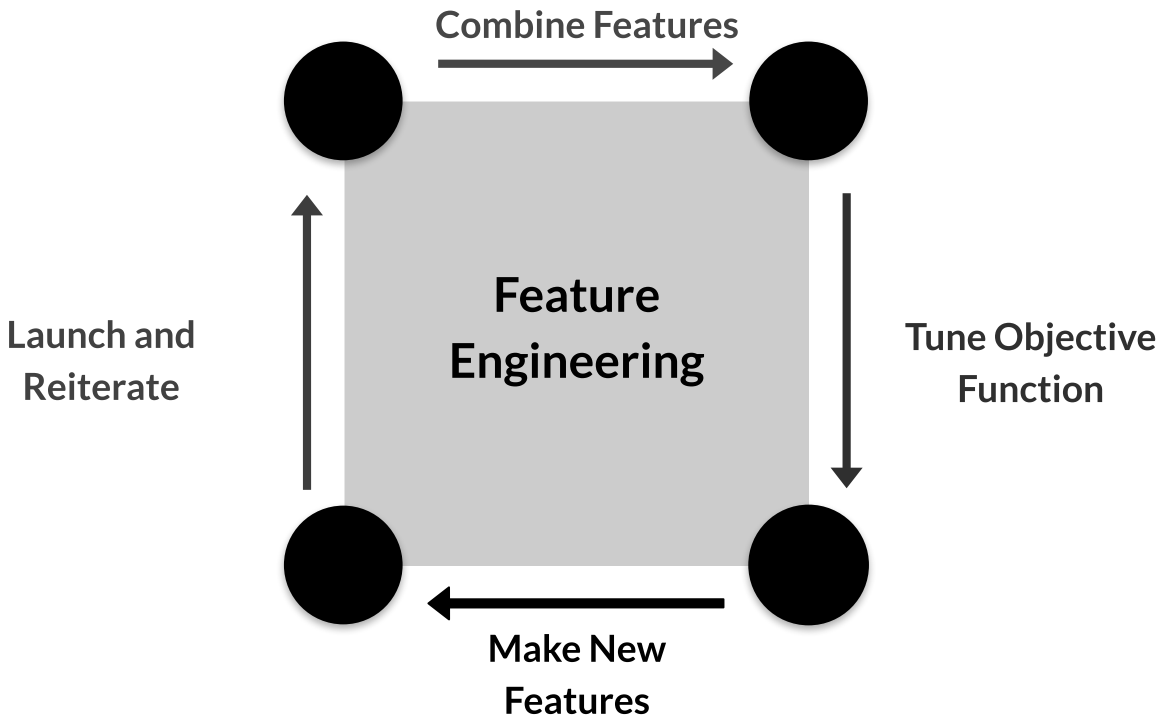
Outline

- Squeezing the most out of data
- The art of feature engineering
- Feature engineering process
- How feature engineering is done in a typical ML pipeline

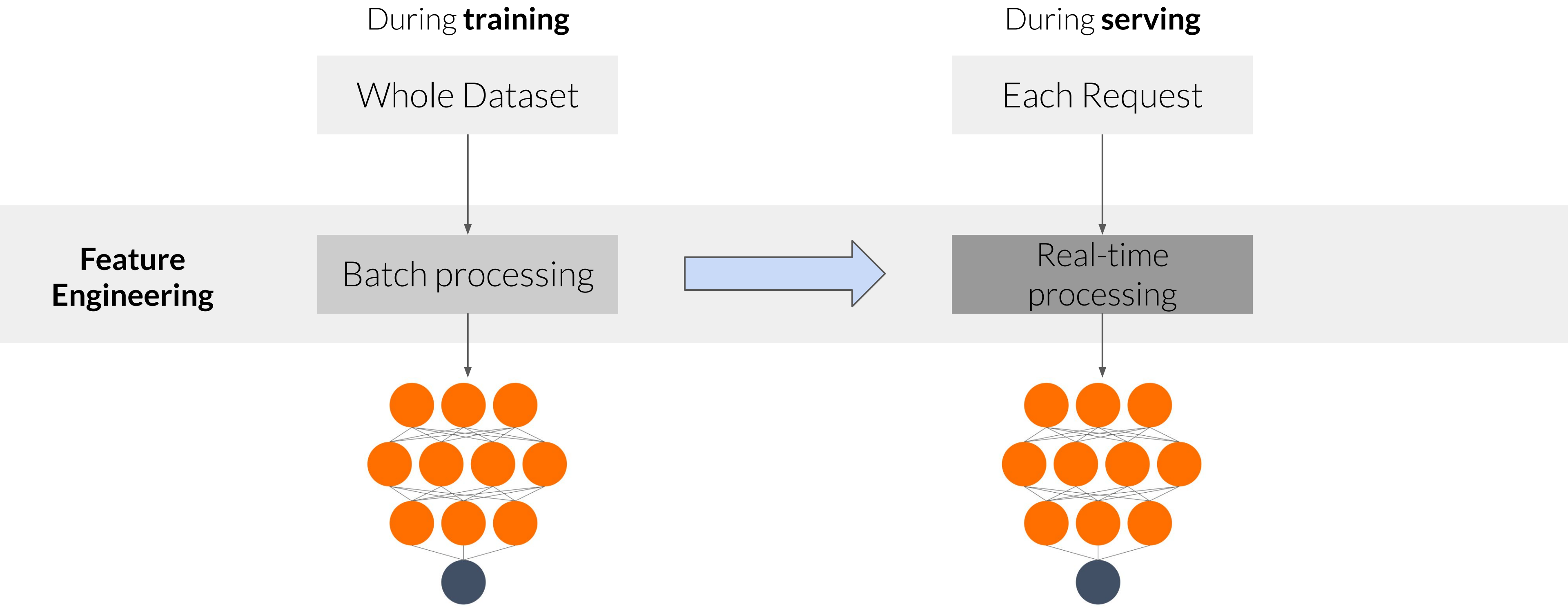
Squeezing the most out of data

- Making data useful before training a model
- Representing data in forms that help models learn
- Increasing predictive quality
- Reducing dimensionality with feature engineering

Art of feature engineering



Typical ML pipeline



Key points

- Feature engineering can be difficult and time consuming, but also very important to success
- Squeezing the most out of data through feature engineering enables models to learn better
- Concentrating predictive information in fewer features enables more efficient use of compute resources
- Feature engineering during training must also be applied correctly during serving

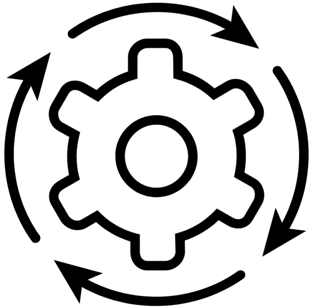
DeepLearning.AI

Feature Engineering

Preprocessing Operations

Outline

- Main preprocessing operations
- Mapping raw data into features
- Mapping numeric values
- Mapping categorical values
- Empirical knowledge of data



Main preprocessing operations

- Data cleansing
- Feature tuning
- Representation transformation
- Feature extraction
- Feature construction

Mapping raw data into features

Raw Data

```
0: {
 house_info : {
 num_rooms : 6
 num_bedrooms : 3
 street_name: “Shorebird Way”
 num_basement_rooms: -1
...
   }
}
```

Raw data doesn’t come to us as feature vectors

Feature Engineering

Feature Vector

```
[
6.0,
1.0,
0.0,
0.0,
9.321,
-2.20,
1.01,
0.0,
...,
]
```

Process of creating features from raw data is **feature engineering**

Mapping categorical values

Street names

{'Charleston Road', 'North Shoreline Boulevard', 'Shorebird Way', 'Rengstorff Avenue'}

Raw Data

```
0: {
 house_info : {
 num_rooms : 6
 num_bedrooms : 3
 street_name: “Shorebird Way”
 num_basement_rooms: -1
...
  }
}
```

String Features can be handled with one-hot encoding

Feature Engineering

Feature Vector

One-hot encoding
This has a 1 for “Shorebird way” and 0 for all others

```
street_name feature=
[0,0, ..., 0, 1, 0, ..., 0]
```

Categorical Vocabulary

```
# From a vocabulary list

vocabulary_feature_column = tf.feature_column.categorical_column_with_vocabulary_list(

                            key=feature_name,

                            vocabulary_list=["kitchenware", "electronics", "sports"])
# From a vocabulary file

vocabulary_feature_column = tf.feature_column.categorical_column_with_vocabulary_file(

                             key=feature_name,

                             vocabulary_file="product_class.txt",

                             vocabulary_size=3)
```

Empirical knowledge of data

Text - stemming, lemmatization, TF-IDF, n-grams, embedding lookup

Images - clipping, resizing, cropping, blur, Canny filters, Sobel filters, photometric distortions

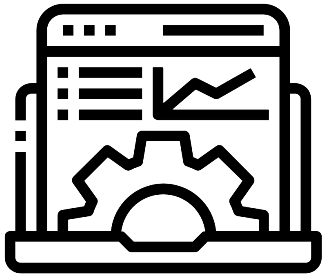
Key points

- Data preprocessing: transforms raw data into a clean and training-ready dataset
- Feature engineering maps:
  - Raw data into feature vectors
  - Integer values to floating-point values
  - Normalizes numerical values
  - Strings and categorical values to vectors of numeric values
  - Data from one space into a different space

Feature Engineering Techniques

Outline

- Feature Scaling
- Normalization and Standardization
- Bucketizing / Binning
- Other techniques



Feature engineering techniques

Numerical Range

- Scaling
- Normalizing
- Standardizing

Grouping

- Bucketizing
- Bag of words

Scaling

- Converts values from their natural range into a prescribed range
    - E.g. Grayscale image pixel intensity scale is [0,255] usually rescaled to [-1,1]

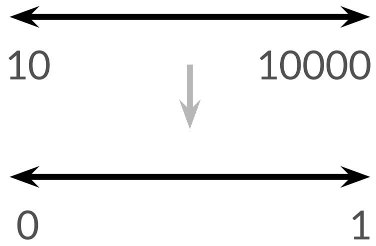
```
image = (image - 127.5) / 127.5
```

- Benefits
    - Helps neural nets converge faster
    - Do away with NaN errors during training
    - For each feature, the model learns the right weights

Normalization

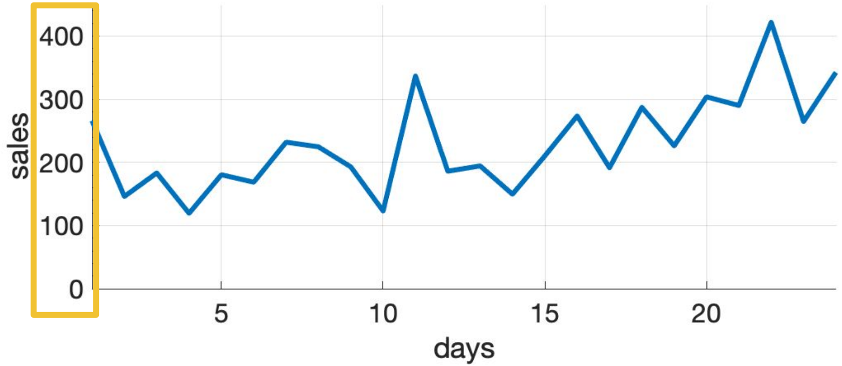
$$X_{\text{norm}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

$$X_{\text{norm}} \in [0, 1]$$

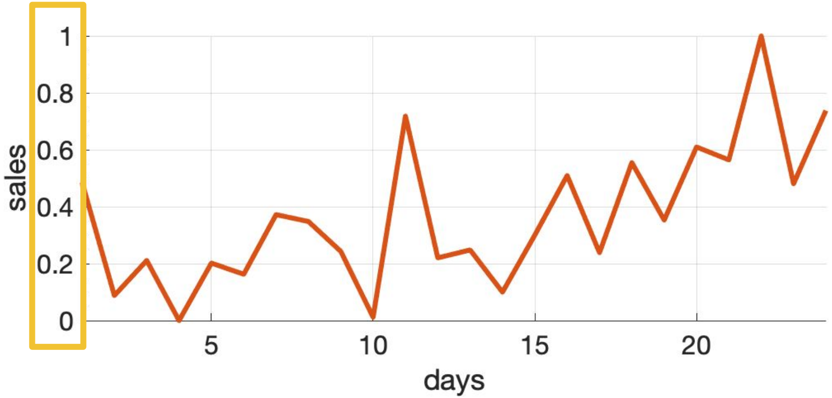


Normalization

Original



Normalized

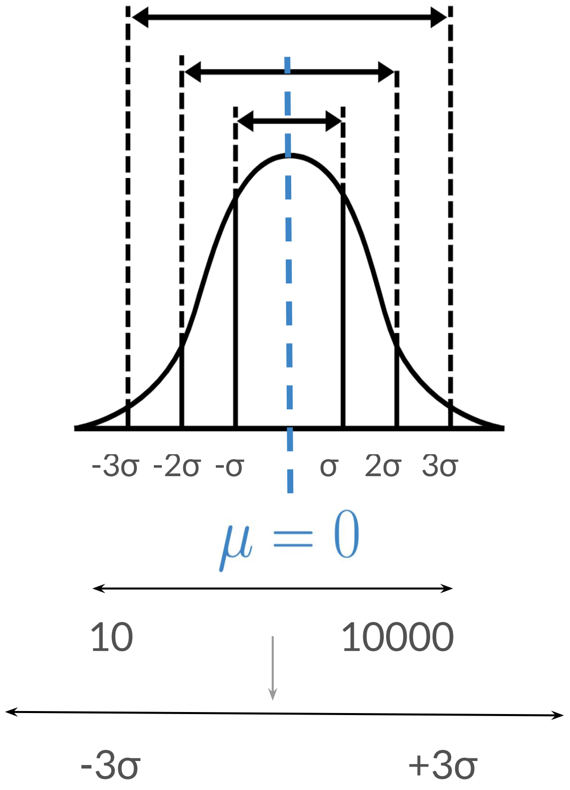


Standardization (z-score)

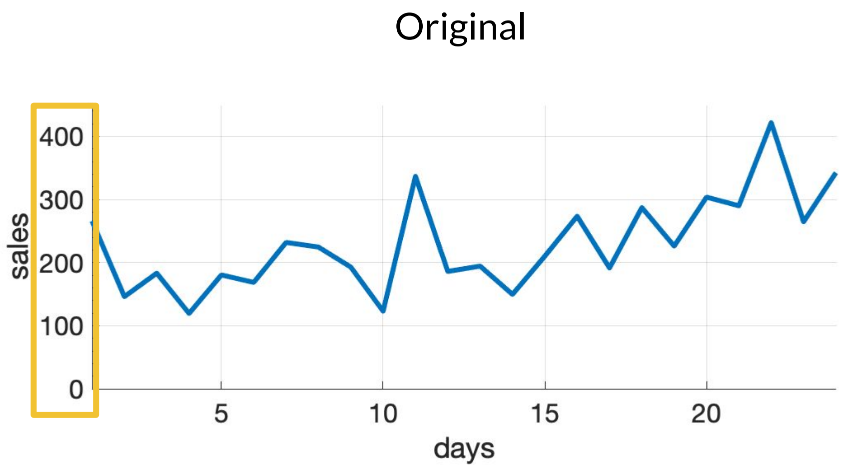
- Z-score relates the number of standard deviations away from the mean
- Example:

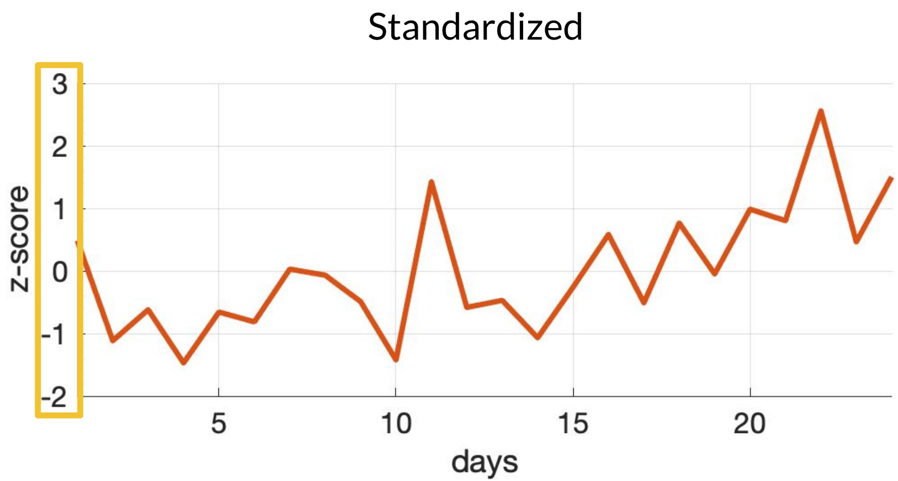
$$X_{\text{std}} = \frac{X - \mu}{\sigma} \qquad \text{(z-score)}$$

$$X_{\text{std}} \sim \mathcal{N}(0, \sigma)$$

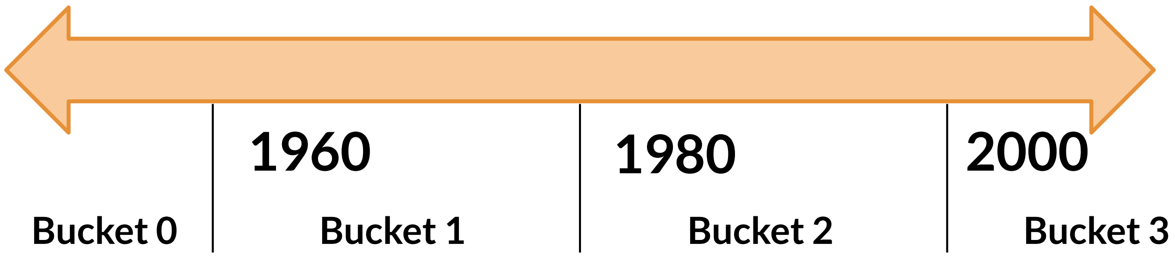


Standardization (z-score)



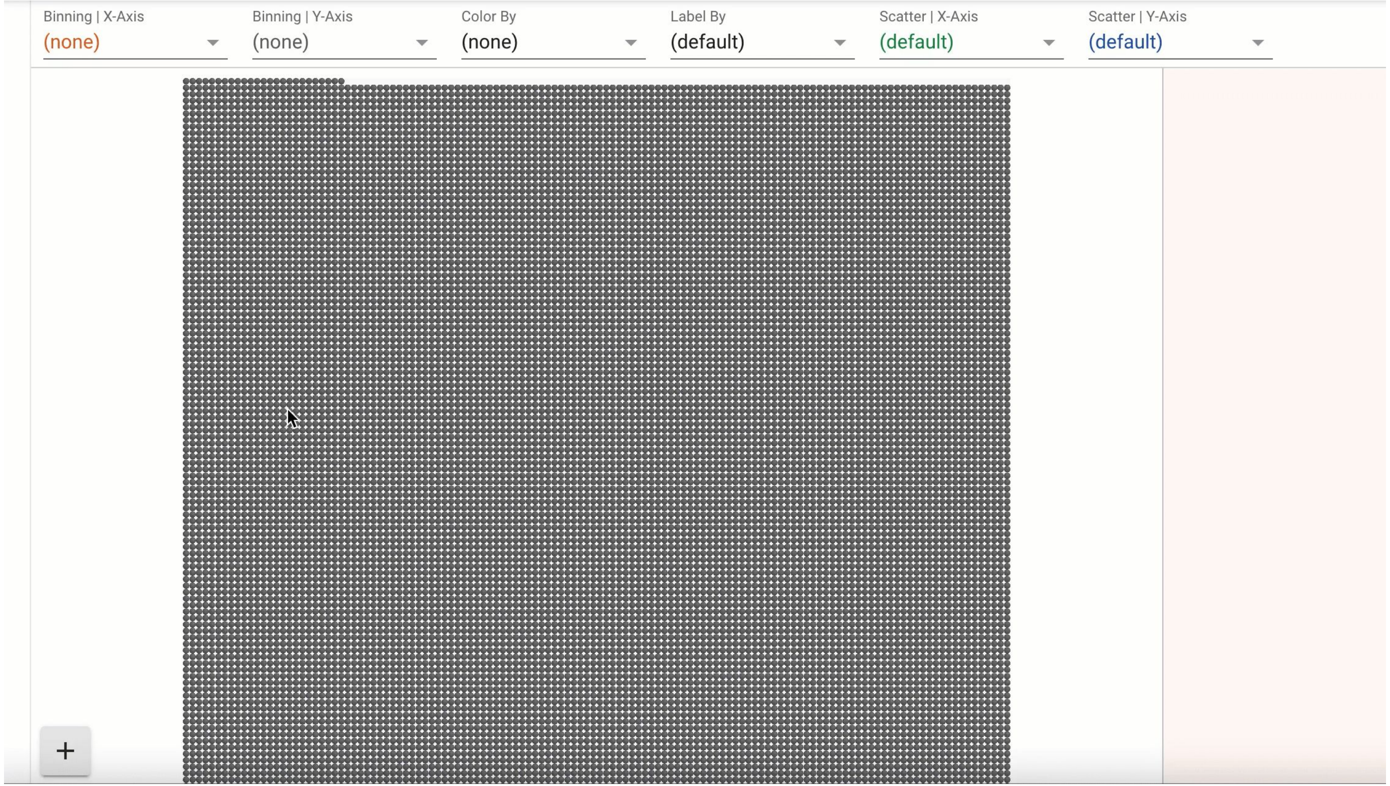


Bucketizing / Binning



Date Range	Represented as...
< 1960	[1, 0, 0, 0]
>= 1960 but < 1980	[0, 1, 0, 0]
>= 1980 but < 2000	[0, 0, 1, 0]
>= 2000	[0, 0, 0, 1]

Binning with Facets



Other techniques

Dimensionality reduction in embeddings

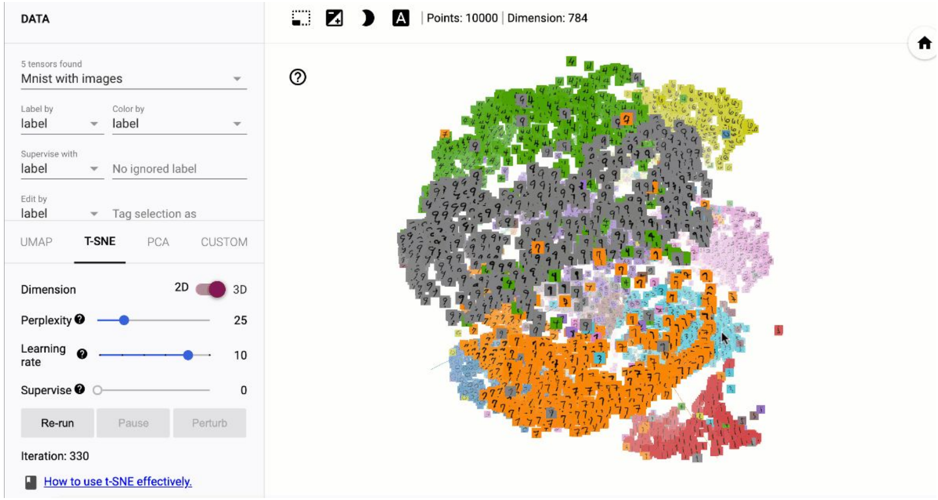
- Principal component analysis (PCA)
- t-Distributed stochastic neighbor embedding (t-SNE)
- Uniform manifold approximation and projection (UMAP)

Feature crossing

TensorFlow embedding projector

- Intuitive exploration of high-dimensional data
- Visualize & analyze
- Techniques
  - PCA
  - t-SNE
  - UMAP
  - Custom linear projections
- Ready to play

  @ projector.tensorflow.org



Key points

- Feature engineering:
    - Prepares, tunes, transforms, extracts and constructs features.
- Feature engineering is key for model refinement
- Feature engineering helps with ML analysis

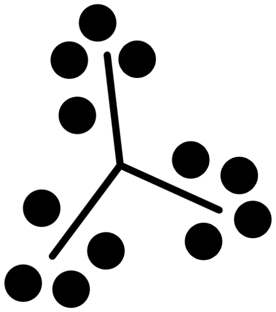
DeepLearning.AI

Feature Engineering

Feature Crosses

Outline

- Feature crosses
- Encoding features

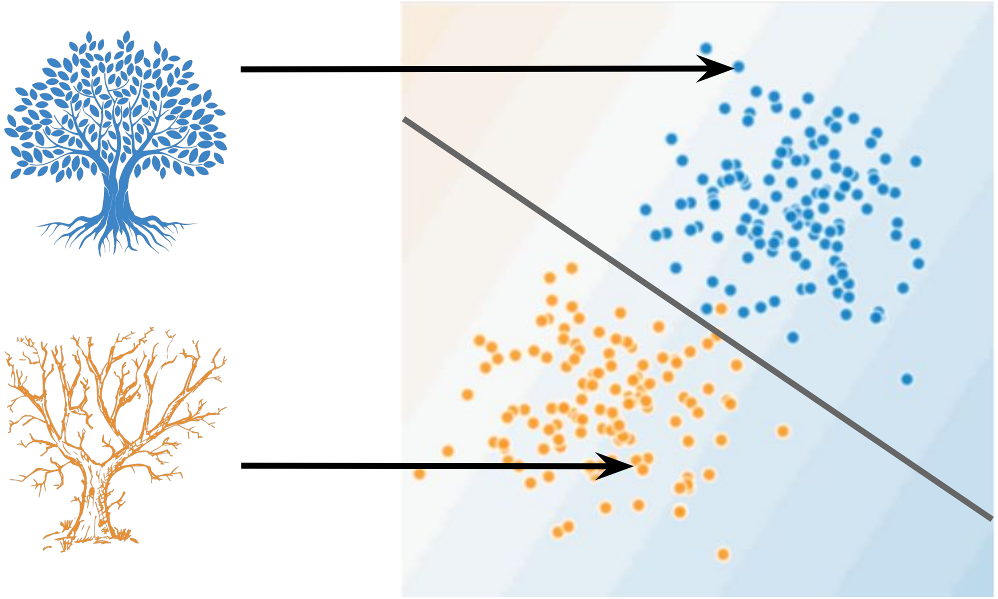


Feature crosses

- Combines multiple features together into a new feature
- Encodes nonlinearity in the feature space, or encodes the same information in fewer features
- [A X B]: multiplying the values of two features
- [A x B x C x D x E]: multiplying the values of 5 features
- [Day of week, Hour] => [Hour of week]

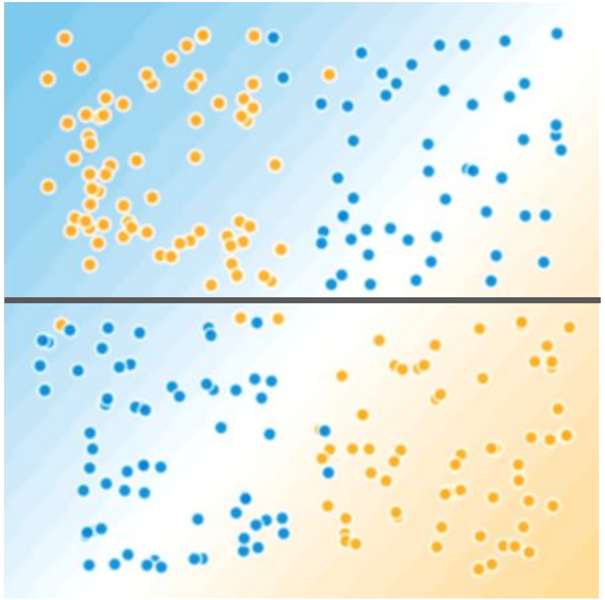
We can create many different kinds of feature crosses

Encoding features



- healthy trees
- sick trees
- Classification boundary

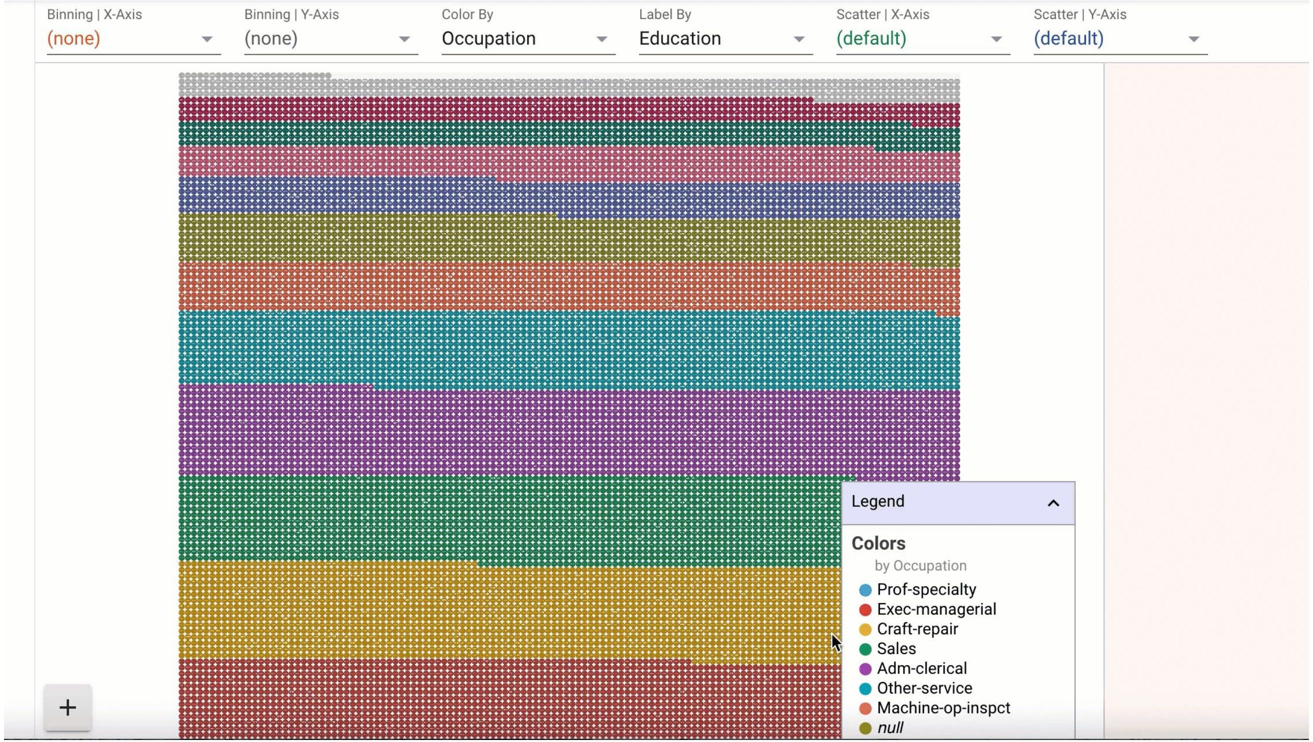
Need for encoding non-linearity



- healthy trees
- sick trees

— Classification boundary

Census dataset



Key points

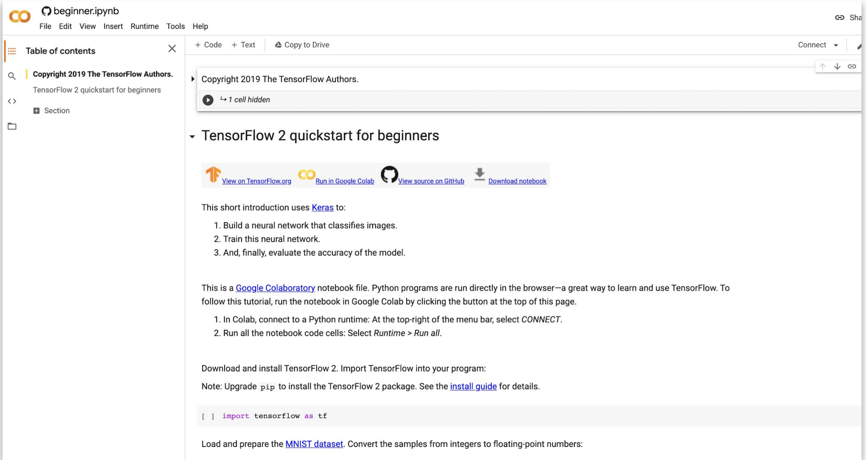
- Feature crossing: synthetic feature encoding nonlinearity in feature space.
- Feature coding: transforming categorical to a continuous variable.

DeepLearning.AI

Feature Transformation At Scale

Preprocessing Data At Scale

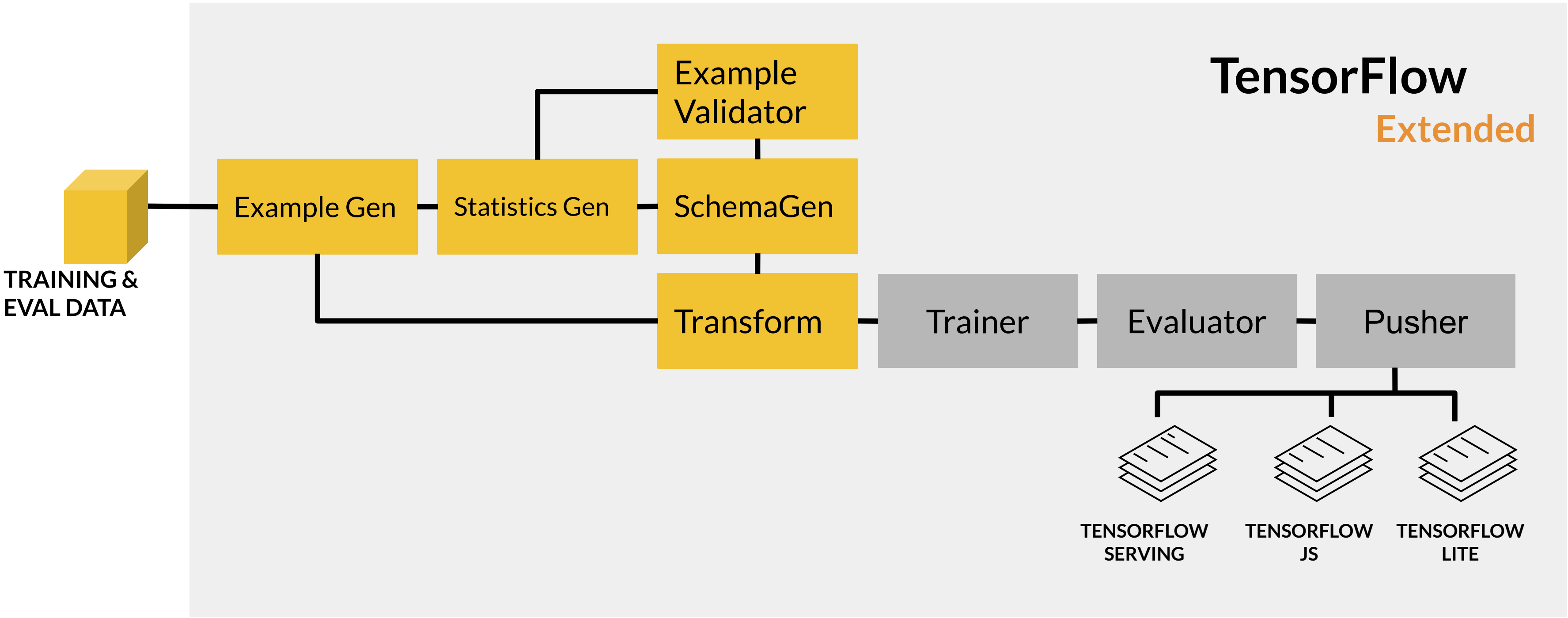
Probably not ideal



APACHE STORM™

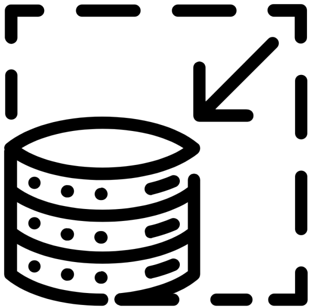
Python → Java

ML Pipeline


TensorFlow
Extended
TRAINING &
EVAL DATA
Example Gen
Statistics Gen
Example Validator
SchemaGen
Transform
Trainer
Evaluator
Pusher
TENSORFLOW SERVING
TENSORFLOW JS
TENSORFLOW LITE

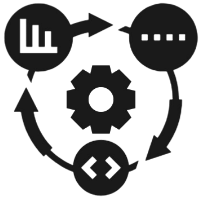
Outline

- Inconsistencies in feature engineering
- Preprocessing granularity
- Pre-processing training dataset
- Optimizing instance-level transformations
- Summarizing the challenges

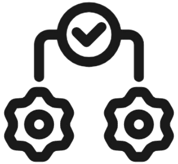


Preprocessing data at scale

Real-world models: terabytes of data



Large-scale data processing frameworks



Consistent transforms between training & serving

Inconsistencies in feature engineering

Training & serving code paths are different

Diverse deployments scenarios

- Mobile (TensorFlow Lite)
- Server (TensorFlow Serving)
- Web (TensorFlow JS)

Risks of introducing training-serving skews

Skews will lower the performance of your serving model

Preprocessing granularity

Transformations	
Instance-level	**Full-pass**
Clipping	Minimax
Multiplying	Standard scaling
Expanding features	Bucketizing
etc.	etc.

When do you transform?

Pre-processing training dataset

Pros	Cons
Run-once	Transformations reproduced at serving
Compute on entire dataset	Slower iterations

How about ‘within’ a model?

Transforming within the model

Pros	Cons
Easy iterations	Expensive transforms
Transformation guarantees	Long model latency
	Transformations per batch: skew

Why transform per batch?

- For example, normalizing features by their average
- Access to a single batch of data, not the full dataset
- Ways to normalize per batch
  - Normalize by average within a batch
  - Precompute average and reuse it during normalization

Optimizing instance-level transformations

- Indirectly affect training efficiency
- Typically accelerators sit idle while the CPUs transform
- Solution:
    - Prefetching transforms for better accelerator efficiency

Summarizing the challenges

- Balancing predictive performance
- Full-pass transformations on training data
- Optimizing instance-level transformations for better training efficiency (GPUs, TPUs, …)

Key points

- Inconsistent data affects the accuracy of the results
- Need for scaled data processing frameworks to process large datasets in an efficient and distributed manner