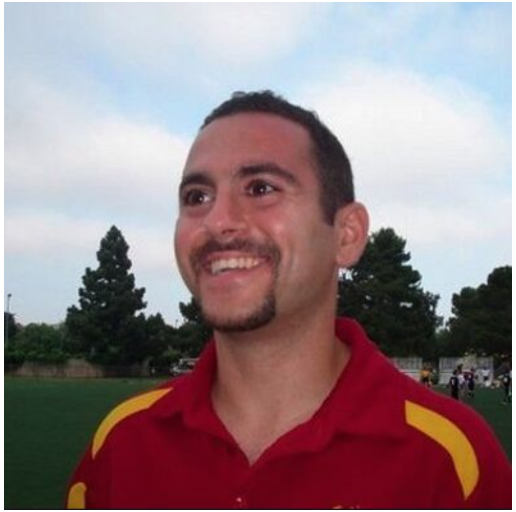
TensorFlow Extended Part 1

Data Validation & Transform

Armen Donigian

Who am I?



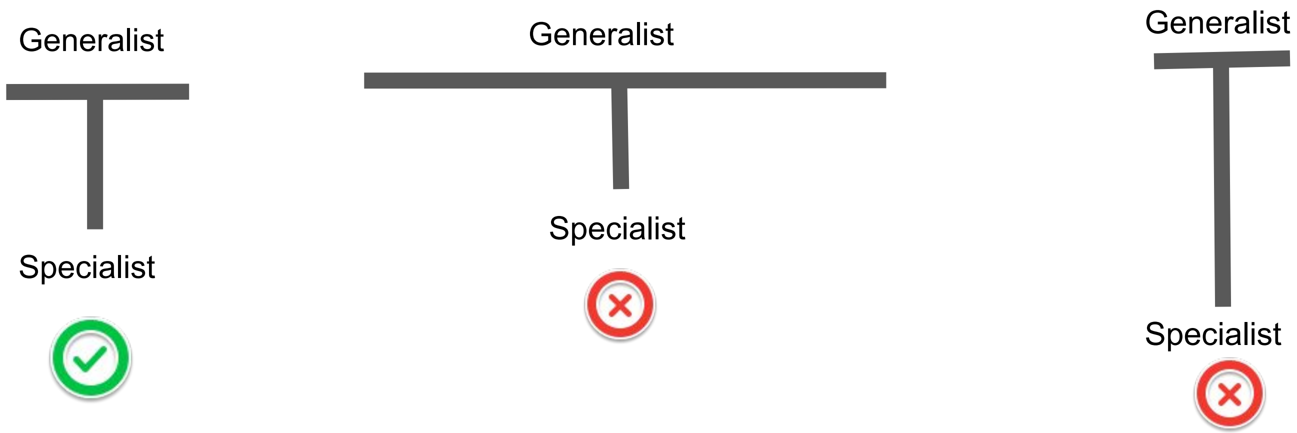
- Computer Science Undergrad degree @UCLA
- Computer Science Grad degree @USC
- 15+ years experience as Software & Data Engineer
- Computer Science Instructor
- Mentor @Udacity Deep Learning Nanodegree
- Real-time wagering algorithms @GamePlayerNetwork
- Differential GPS corrections @Jet Propulsion Laboratory, landing sequence for Mars Curiosity
- Years of experience in design, implementation & productionalization of machine learning models for several FinTech underwriting businesses
- Currently, head of personalization & recommender systems @Honey
- Available for Consulting (donigian@LevelUpInference.com)

Goals, Breadth vs Depth...

Goal: Provide context of the *requirements*, *tools* & *methodologies* involved with developing a production grade machine learning pipeline.

Slides will provide you with *breadth*.

Notebooks will provide you with *depth* (i.e. implementation details).



Lesson Roadmap

Day 1

- **Overview of TFX: What problems it can help you solve (30 mins)**
  a. What is TFX & Why Should You Care?
  b. What can you leverage? TFX Ecosystem
  c. Which problems can TFX help you solve?
  d. TFX Components

  10 min Break

- **TensorFlow Data Validation Overview (45 mins)**
  a. Review most common real world challenges!
     i. Which TFDV methods help you solve them
  b. What are common types of Skews?
  c. Dataset Overview
  d. Schema Inference & Validation
  e. How to Visualize Data at scale?
  f. How to detect Data Anomalies?

  10 min Break

- **TensorFlow Transform Overview (40 mins)**
  a. Review most common real world transformations!
  b. Apache Beam & TFT
  c. Pre-processing using TFT
     i. TFT Analyzers
  d. How to use Apache Beam effectively?
  e. Load dataset, pre-process & train model

  10 min Break

- **Example Case Study integrating TF Data Validation & Transform (35 mins)**
  a. Review End to End TFDV & TFMA Notebooks

TensorFlow Extended Overview

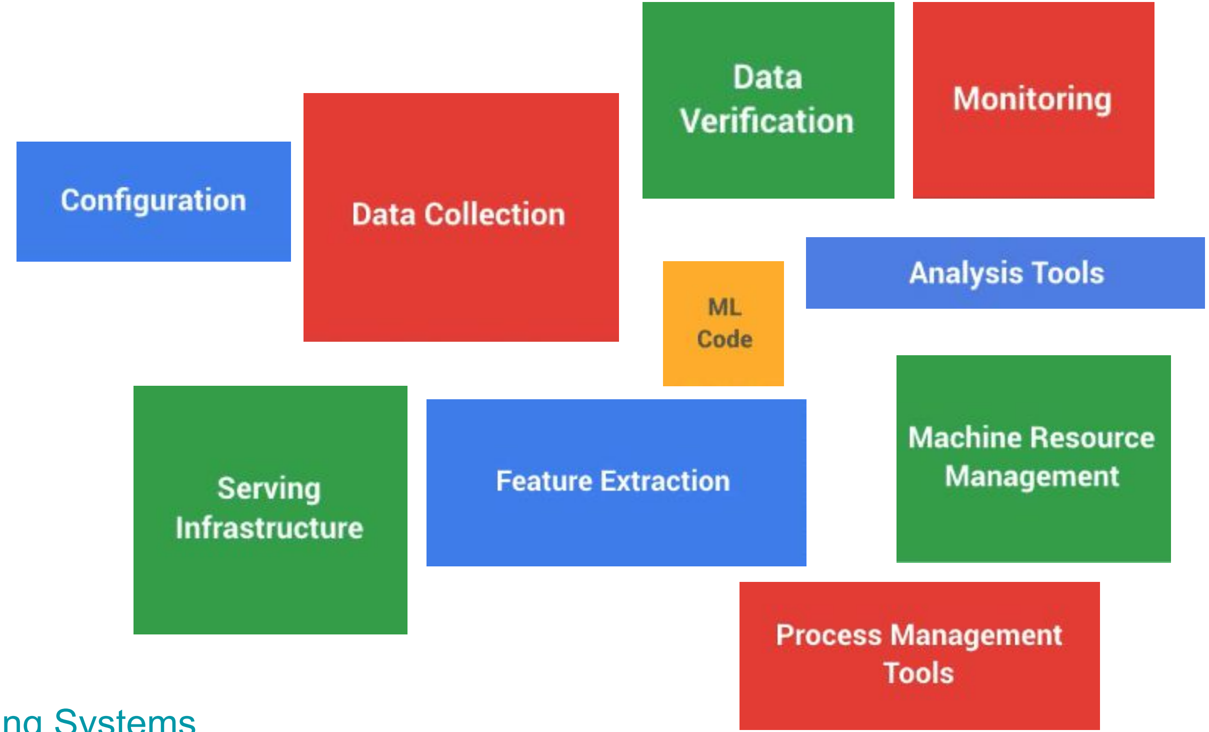
TensorFlow Extended (TFX)

TFX is…

- A general purpose machine learning platform implemented @Google
- A set of gluable components into one platform simplifying the development of end to end ML pipelines.
- An open source solution to reduce the time to production from months to weeks while minimizing custom, fragile solutions filled with tech debt.
- Used by Google to create & deploy their machine learning models.

Why Should You Care?

What you first think? **vs...** Real World ML Use Cases

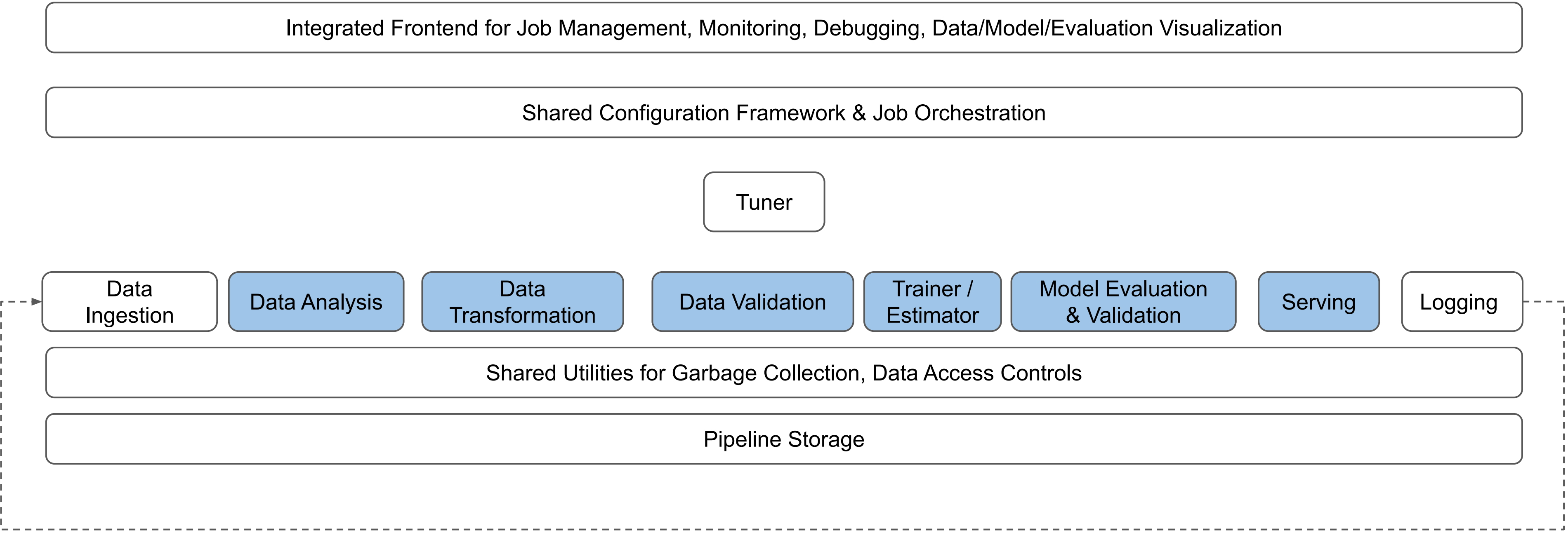


Takeaway: Doing machine learning in real world is HARD!

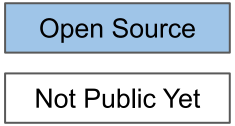
Building custom solutions is expensive, duplicative, fragile & leads to tech debt.

[Hidden Technical Debt in Machine Learning Systems]

What Can I Leverage: TFX Ecosystem



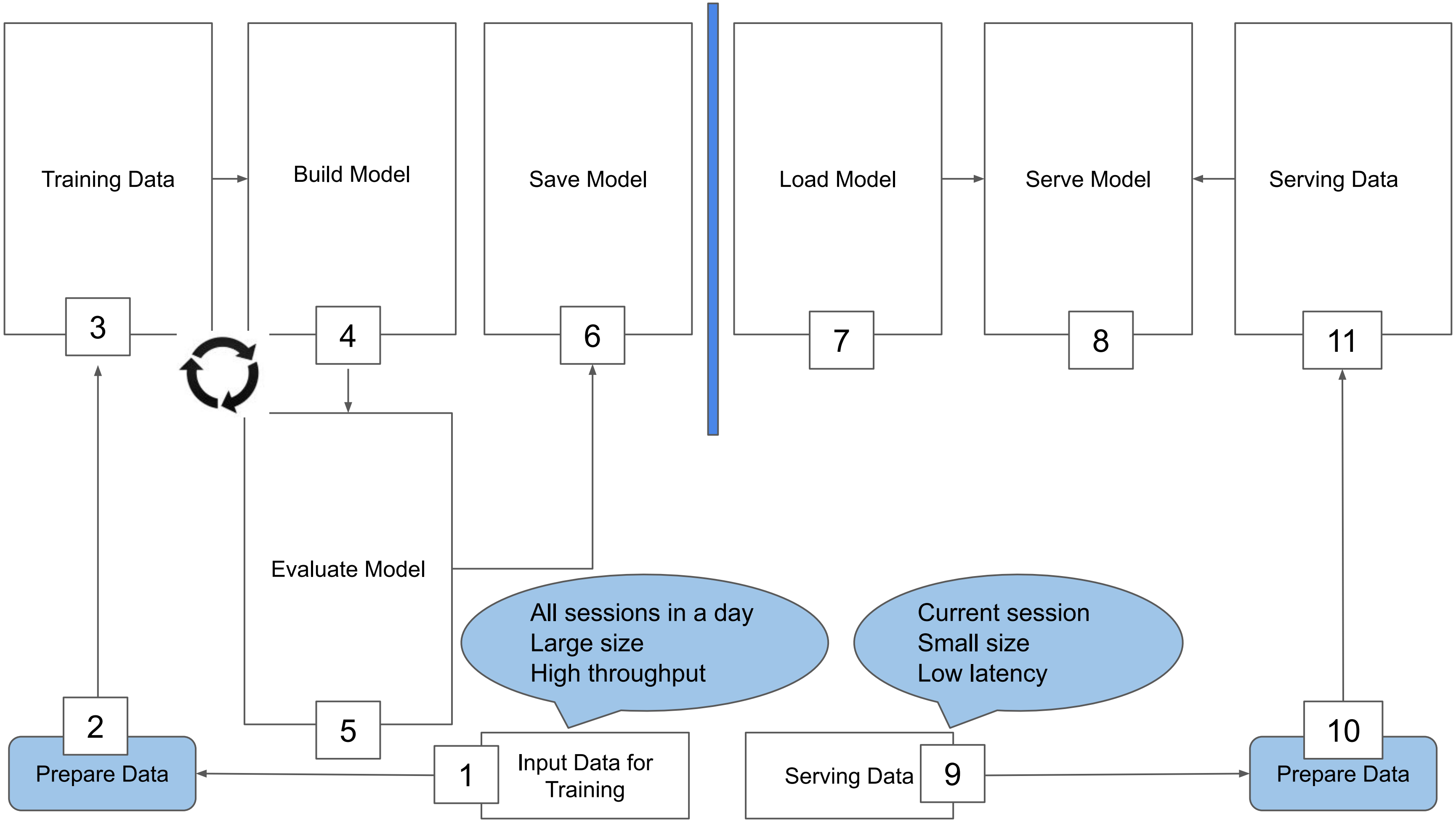
Machine Learning Platform Overview



[Link](#) to TFX paper

Train / Serving Data Flows

A Data Science Workflow, Glossary of ML terms, Diagram Reference



What Could Go Wrong...

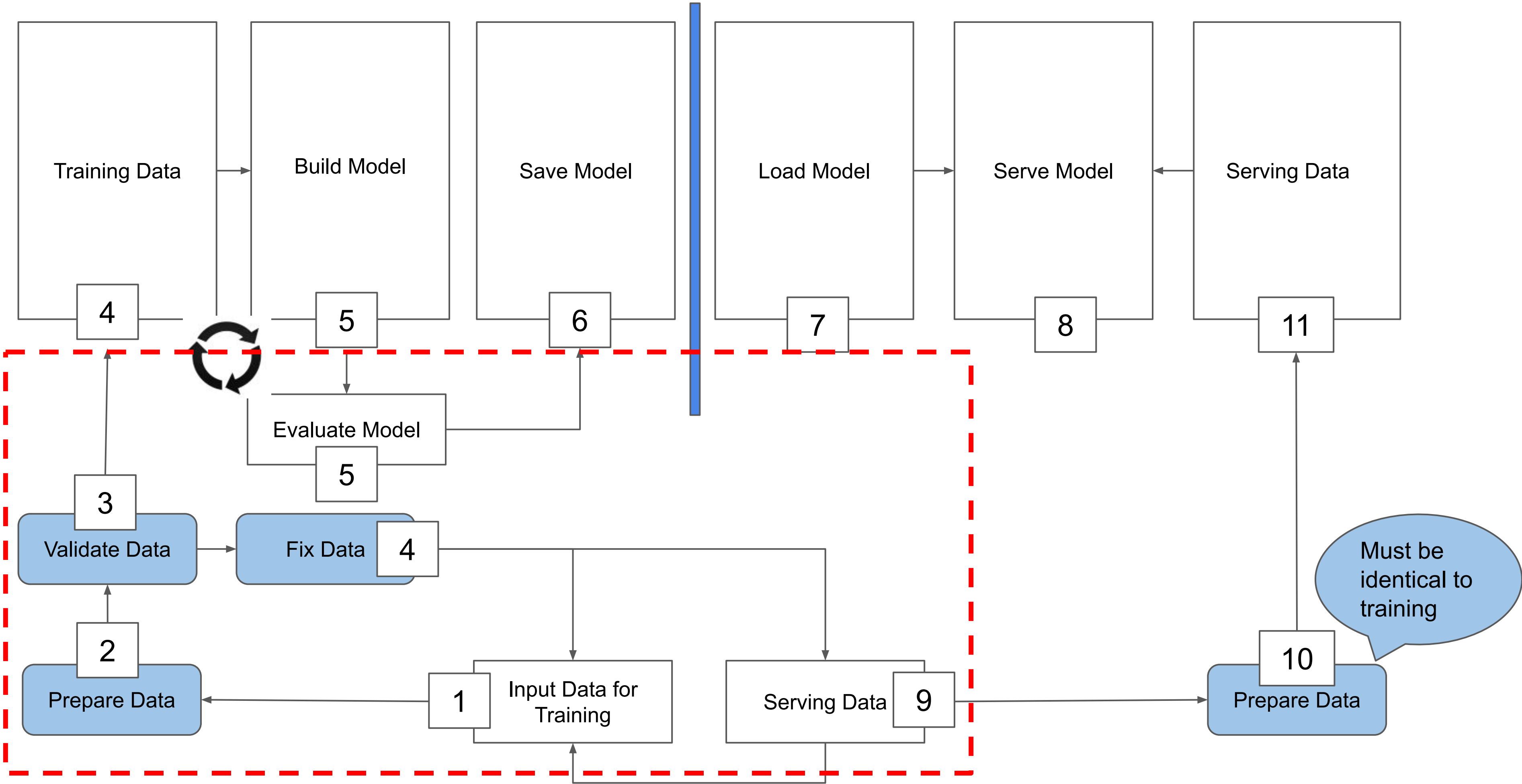
In no particular order…

- What errors are in the data? Finding errors in GBs or TBs w/ O(1000s) of features is hard!
- How do I standardize data input pipeline when there are tens of diverse data storage systems with different formats?
- How do I gain an understanding (analysis or visualization) of GBs or TBs w/ O(1000s) of features?
  - What is a reasonable data schema? How can I define a training vs serving context?
  - Does new data conform to previously inferred schema (validation)?
  - How can I detect when a signal is available in training but not in serving?
- Which data significantly affects the performance of the model?
- How different are the training vs test vs serving sets?
  - Are these differences important? How can I define constraints on distribution of values?
- Which data characteristics do we want to alert on? How sensitive should the alerts be?
- Which part of the data is problematic?
- How can I apply data transformations to GBs or TBs w/ O(1000s) of features in a scalable way?
- How to backfill data with a fix to a known issue?

Click links above to find related research papers & projects.

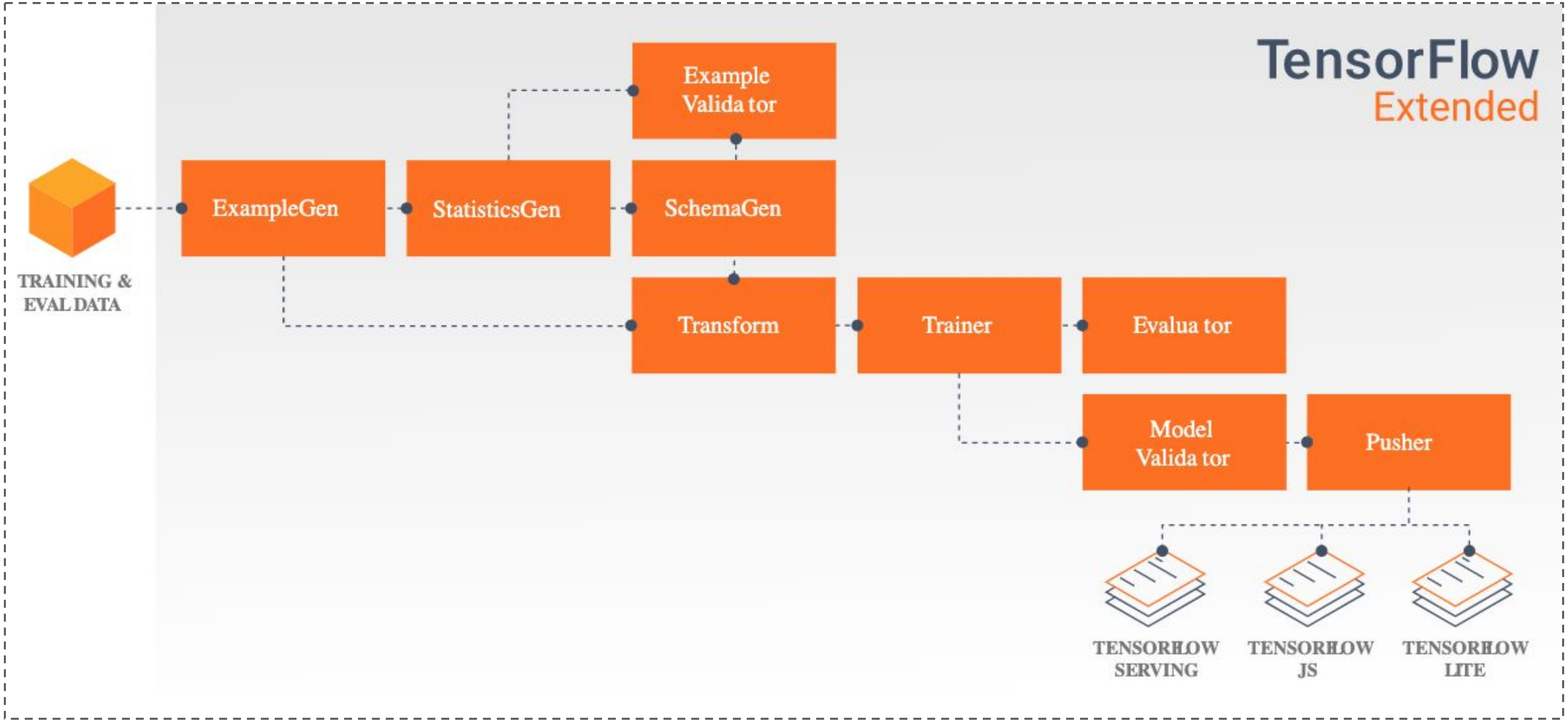
Train / Serving Data Flows

A Data Science Workflow, Glossary of ML terms, Diagram Reference



Architecture Overview

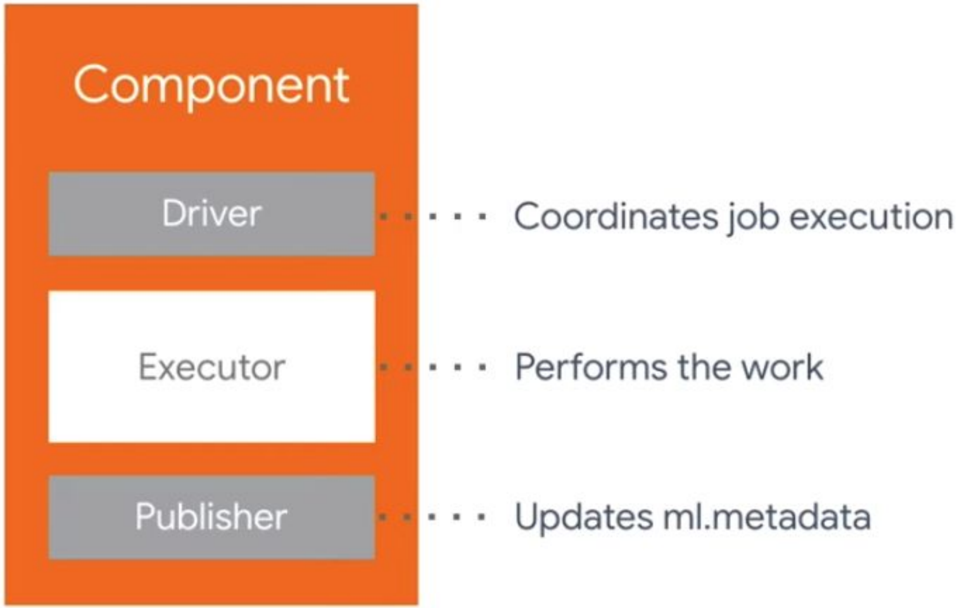
TFX pipelines can be orchestrated using Apache Airflow and Kubeflow Pipelines.
For this workshop, we will be running in interactive mode.



source

What is a TFX Component?

- TFX pipelines are a series of components
- Components are organized into DAGs
- Executor is where insert your work will be
- Driver feeds data to Executor
- Publisher writes to ml.metadata

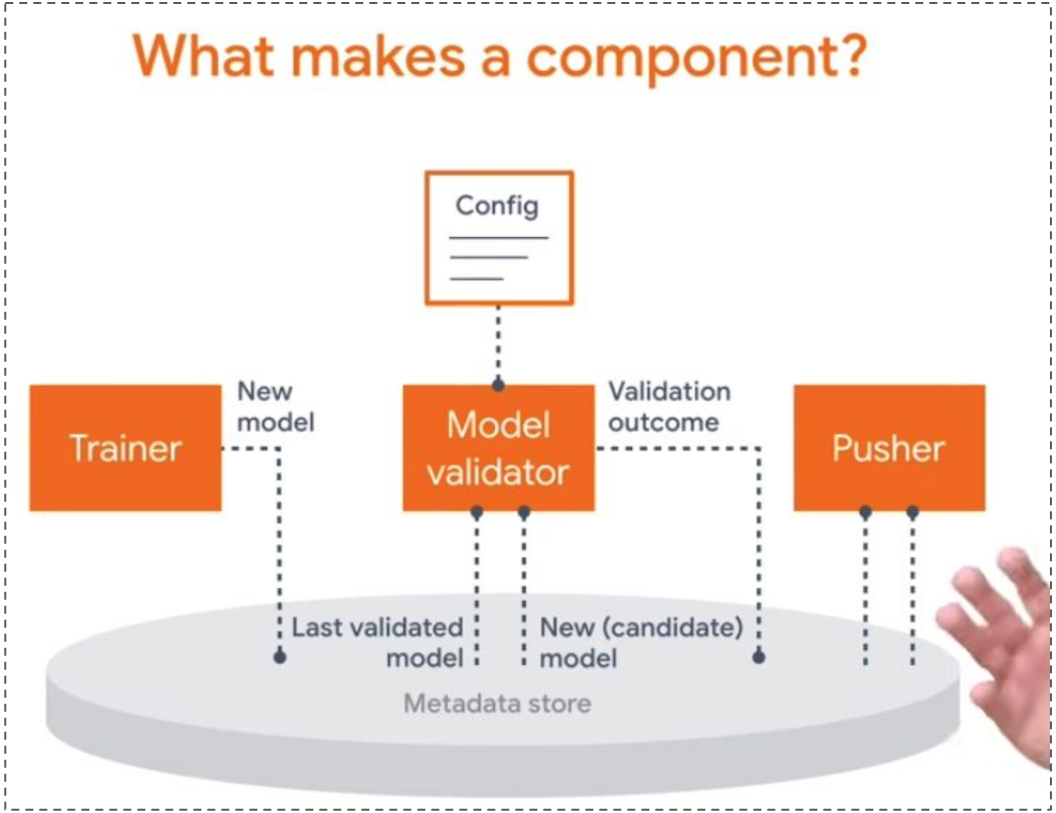


source

What is a TFX Component?

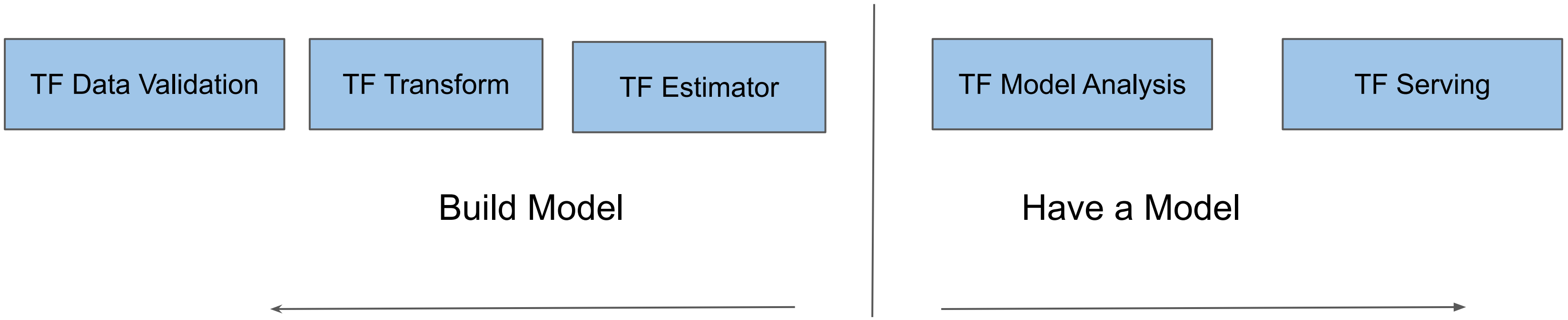
As data flows through pipelines…

- Components read data coming from Metadata store and ...
- Write data to components further in the pipeline...
- Except when at start and end
- Orchestrators like (Kubeflow or Airflow) help you manage triggering of tasks and monitor components
- ML Metadata store is a RDBMs containing...
    - Trained & re-trained Models
    - Data we trained with
    - Evaluation results
    - Location of data objects (not data)
    - Execution history records for every component
    - Data provenance of intermediate outputs



source

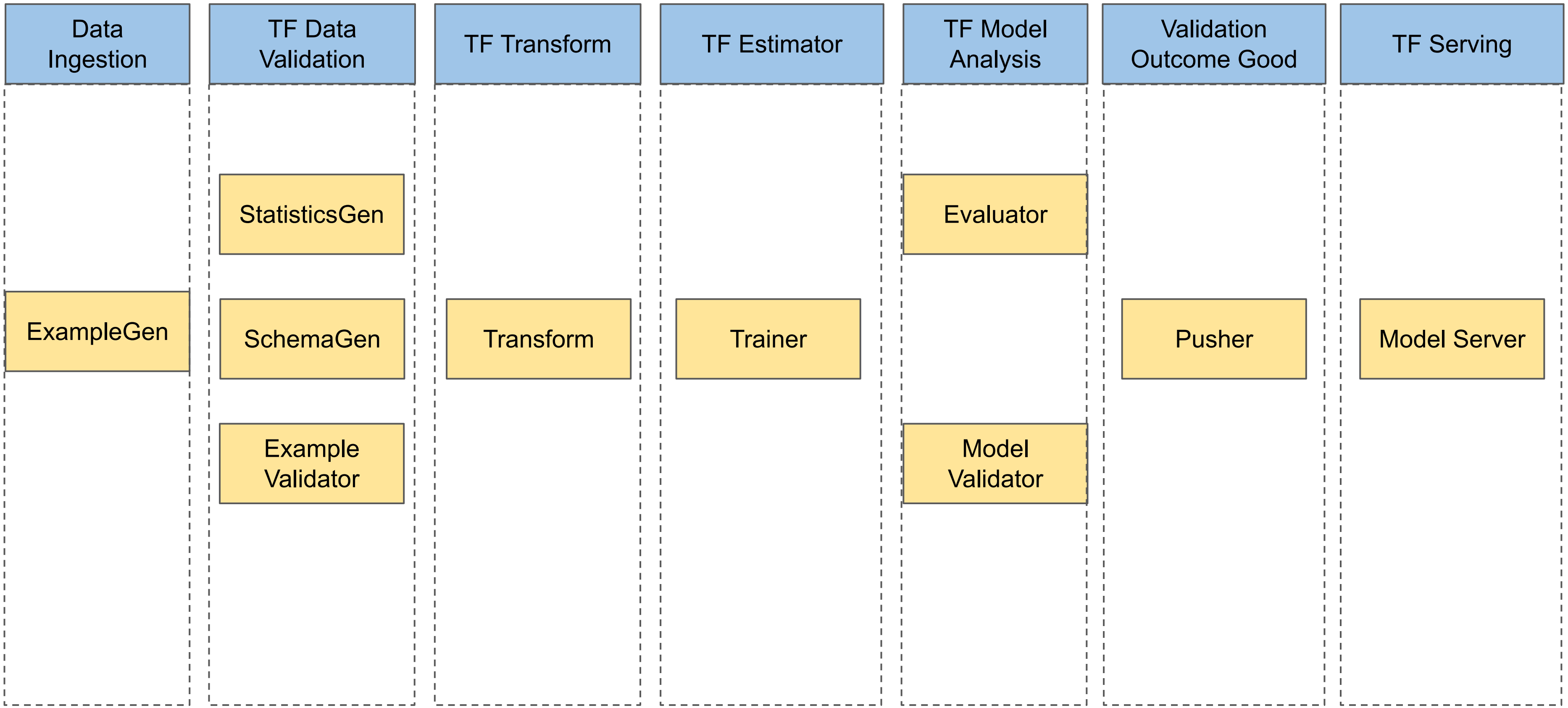
TFX Pipeline



Components API (docs)

- ExampleGen ingests and splits the input dataset.
- StatisticsGen calculates statistics for the dataset.
- SchemaGen SchemaGen examines the statistics and creates a data schema.
- ExampleValidator looks for anomalies and missing values in the dataset.
- Transform performs feature engineering on the dataset.
- Trainer trains the model using TensorFlow Estimators
- Evaluator performs deep analysis of the training results.
- ModelValidator ensures that the model is "good enough" to be pushed to production.
- Pusher deploys the model to a serving infrastructure.
- TensorFlow Serving for serving.

Components



source

Installation Notes

Python 3.x support now available for Apache Beam

Install Dependencies

```
%%bash
pip install tensorflow==1.14.0
pip install tfx==0.14.0rc1
pip install apache-beam==2.14.0
pip install tensorflow-data-validation==0.14.1
pip install tensorflow-metadata==0.14.0
pip install tensorflow-model-analysis==0.14.0
pip install tensorflow-transform==0.14.0
```

Compatible versions

The following table describes how the `tfx` package versions are compatible with its major dependency PyPI packages. This is determined by our testing framework, but other *untested* combinations may also work.

tfx	tensorflow	tensorflow-data-validation	tensorflow-model-analysis	tensorflow-metadata	tensorflow-transform	ml-metadata	apache-beam[gcp]
GitHub master	nightly (1.x)	0.14.1	0.14.0	0.14.0	0.14.0	0.14.0	2.14.0
0.14.0	1.14.0	0.14.1	0.14.0	0.14.0	0.14.0	0.14.0	2.14.0
0.13.0	1.13.1	0.13.1	0.13.2	0.13.0	0.13.0	0.13.2	2.12.0
0.12.0	1.12	0.12.0	0.12.1	0.12.1	0.12.0	0.13.2	2.10.0

source

TensorFlow Data Validation

TensorFlow Data Validation Overview

Better Data, Better Models	Automated schema generation	Integration with Facets (live demo)	Anomaly detection
Compute summary statistics for train/test data	Input feature value ranges	Identify train/test/validation set skew	Detect missing values
Drift detection	Type imputation	Unexpected feature values	Out of range values
Training-Serving skew detection	Environment specific schema	Feature by feature analysis	Wrong feature types
	Schema validation	Compare statistics across two or more data sets	Correct non-conforming data
		Supports visualization of large datasets responsively	

Real World Challenges...	What to keep in mind?	How TFDV can help?
Data contains anomalies…	Data anomalies can impact some learners more negatively than others, as well as interpretability & analysis.	*tfdv.display_anomalies()*
How to visualize high dimensional data	Visualization not only useful for storytelling but easier to detect patterns & relationships	Integration with Facets, *tfdv.visualize_statistics()*
Missing or incomplete data dictionary	Inferring schema for hundreds or thousands of features is challenging!!!	*tfdv.infer_schema()* *tfdv.display_schema()*
Schema Validation	Ensuring features have proper type, range of values, missing values etc (train, eval & serving sets)	*tfdv.validate_statistics()* *validate_instance()*
Need to determine data distribution	Computing summary statistics on at scale is challenging!	tfdv.generate_statistics_from_csv() tfdf.generate_statistics_from_tfrecord()
Train - Serving Skew	Schema skew, Feature skew, Distribution skew	Previous methods will find skew due to faulty sampling, 3pd dependencies or other causes.
Categorical feature drift over time	Monitor features during serving for feature drift	L-infinity distance supported
Common helper methods		get_categorical_numeric_features() get_categorical_features() get_multivalent_features() tfdv.write_schema_text() tfdv.load_schema_text()

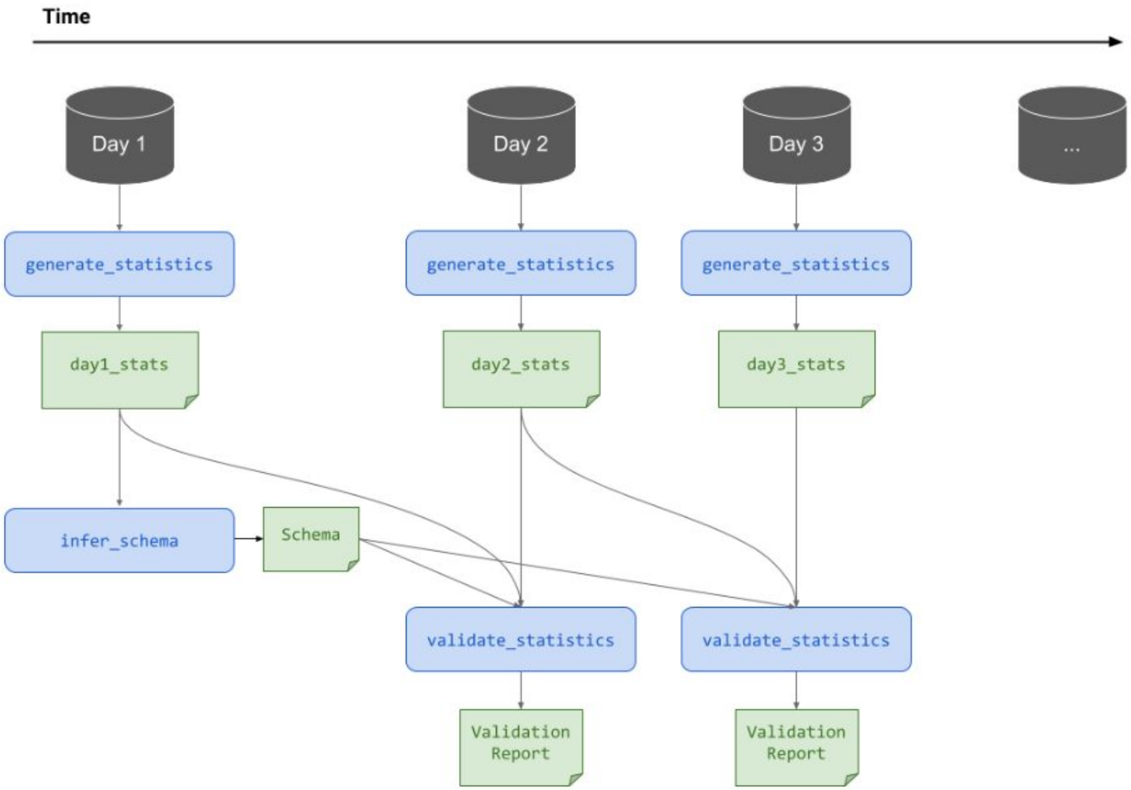
Real World Challenges...	**What to keep in mind?**	**How TFDV can help?**
Labels with invalid data	Be skeptical of labels as you are of features.	*tfdv.display_anomalies()*
Features with different order of magnitudes	Some learners are sensitive to these differences.	Compare min/max values across features (norm)
Bugs causing uniformly distributed data (ex 1)	Row numbers, globally incrementing, many unique values which occur w/ same frequency.	Observe output of *visualize_statistics*()
Bugs causing uniformly distributed data (ex 2)	Row numbers, globally incrementing, many unique values which occur w/ same frequency.	Observe output of *visualize_statistics*()
Unbalanced Feature	Unbalanced features could be expected, but if a feature always has the same value you may have a data bug.	In a Facets Overview, choose "Non-uniformity" from the "Sort by" dropdown.

TensorFlow Data Validation Visualized

Two of the most common use cases for TFDV:

1. Validation of continuously arriving data

2. Training/serving skew detection



source

Dataset

Bucket Features	Dense Float Features	Vocab Features	Categorical Features
pickup_hour	trip_distance		
pickup_month	passenger_count		
pickup_day_of_week	tip_amount		
dropoff_month			
dropoff_hour			
dropoff_day_of_week			

bucketize	scale_to_z_score

Transformations

Target to predict: fare_amount

ExampleGen TFX Pipeline Component

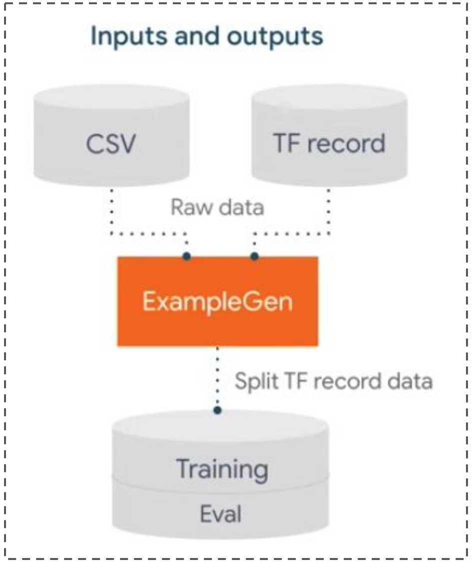
Notes:

- To ingest data into your ML pipeline
  - Input: data formatted in CSV, TFRecord & BigQuery
  - Output: tf.Example records
  - We'll be using CsvExampleGen executor to convert a CSV into TF examples.

```
context = InteractiveContext()

examples = csv_input(TRAIN_DATA_DIR)

# ingest data into pipeline
example_gen = CsvExampleGen(input_base=examples)
context.run(example_gen)
```



- BigQuery based ExampleGen, see this for more details

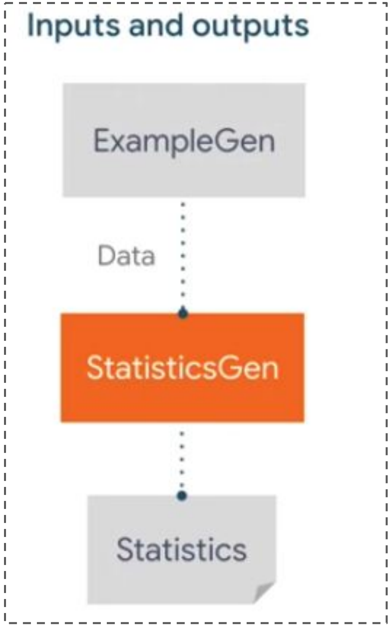
source

StatisticsGen TFX Pipeline Component

Notes:

- Generates feature statistics over training & serving data
    - Makes full pass over data to calculate descriptive statistics
- Scales to large datasets using Apache Beam
    - Input: Datasets produced by ExampleGen component
    - Output: Dataset stats
    - Here's how you can use StatisticsGen

```
statistics_gen = StatisticsGen(
      input_data=example_gen.outputs['examples'])
context.run(statistics_gen)
```

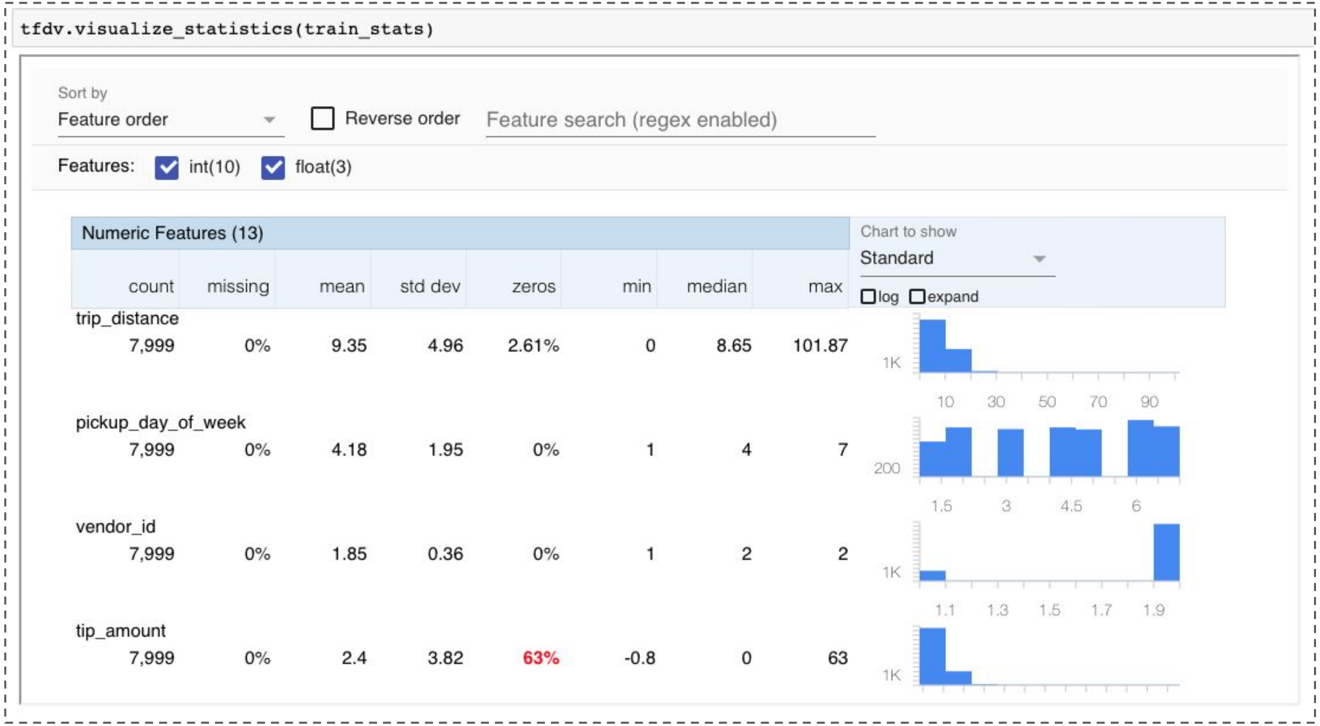


source

Visualize Data

Sanity checks…

- Feature min, max, mean, mode, median
- Randomly assigned values
- Feature correlations
- Class Imbalance
- Variance within each feature, avoid rarely occurring categoricals
- Sanity check data w/ domain knowledge
- Feature contains enough non-missing values (>80% of rows populated)
- Histograms of features (numerical & categorical)
- Feature Cardinality
- Plot of feature moving average



```
tfdv.visualize_statistics(train_stats)
```

Sort by: Feature order | Reverse order | Feature search (regex enabled)

Features: int(10) float(3)

Numeric Features (13)

	count	missing	mean	std dev	zeros	min	median	max
trip_distance	7,999	0%	9.35	4.96	2.61%	0	8.65	101.87
pickup_day_of_week	7,999	0%	4.18	1.95	0%	1	4	7
vendor_id	7,999	0%	1.85	0.36	0%	1	2	2
tip_amount	7,999	0%	2.4	3.82	63%	-0.8	0	63

Visualize Data

Sanity checks…

- How big of a difference is there between training vs evaluation sets?
- Does this difference matter?
- Which values are available in training but not in evaluation?
- Can you think of a column which would always be missing in training vs serving?

```
# compute stats over evaluation dataset
eval_stats = tfdv.generate_statistics_from_csv(data_location = EVAL_DATA)
```

```
# compare stats of train vs eval data
tfdv.visualize_statistics(lhs_statistics=eval_stats, rhs_statistics=train_stats,
                          lhs_name='EVAL_DATA.SET', rhs_name='TRAIN_DATASET')
```

Sort by: Feature order | ☐ Reverse order | Feature search (regex enabled)

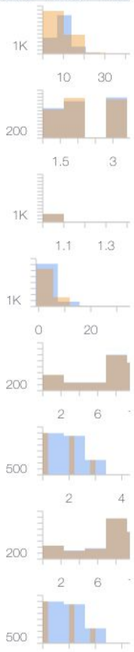
Features: ☑ int(10) ☑ float(3)

EVAL_DATA.SET | TRAIN_DATASET

Numeric Features (13)

	count	missing	mean	std dev	zeros	min	median	max
trip_distance								
EVAL_DATA.SET	8,000	0%	9.41	4.95	2.49%	0	8.7	69.42
TRAIN_DATASET	7,999	0%	9.35	4.96	2.61%	0	8.65	101.87
pickup_day_of_week								
EVAL_DATA.SET	8,000	0%	4.18	1.95	0%	1	4	7
TRAIN_DATASET	7,999	0%	4.18	1.95	0%	1	4	7
vendor_id								
EVAL_DATA.SET	8,000	0%	1.85	0.36	0%	1	2	2
TRAIN_DATASET	7,999	0%	1.85	0.36	0%	1	2	2
tip_amount								
EVAL_DATA.SET	8,000	0%	2.47	3.98	62.39%	-0.99	0	82.5
TRAIN_DATASET	7,999	0%	2.4	3.82	63%	-0.8	0	63
dropoff_hour								
EVAL_DATA.SET	8,000	0%	12.93	5.86	3.09%	0	13	23
TRAIN_DATASET	7,999	0%	13.04	5.9	2.83%	0	13	23
dropoff_month								
EVAL_DATA.SET	8,000	0%	1.72	0.73	0%	1	2	9
TRAIN_DATASET	7,999	0%	1.72	0.72	0%	1	2	3
pickup_hour								
EVAL_DATA.SET	8,000	0%	12.67	5.75	2.63%	0	13	23
TRAIN_DATASET	7,999	0%	12.83	5.78	2.18%	0	13	23
pickup_month								
EVAL_DATA.SET	8,000	0%	1.72	0.73	0%	1	2	9
TRAIN_DATASET	7,999	0%	1.72	0.72	0%	1	2	3

Chart to show: Standard | ☐ log ☐ expand ☐ per

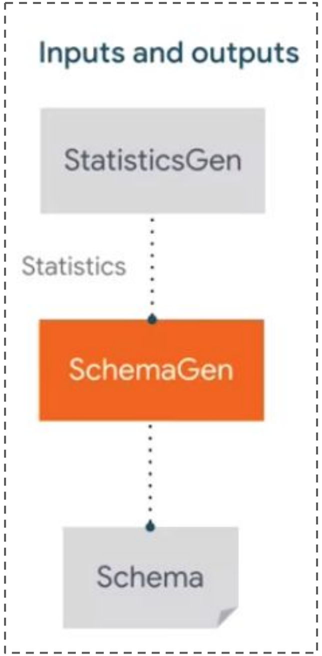


SchemaGen TFX Pipeline Component

Notes:

- Looks at statistics to infer types for each feature
- Range values, whether feature is available etc…
- Auto generated schema
    - Input: Statistics from an StatisticsGen component
    - Output: Data schema proto
    - Here’s how you can call it...

```
infer_schema = SchemaGen(
    stats=statistics_gen.outputs['output'],
    infer_feature_shape=False)
context.run(infer_schema)
```



source

Schema Inference

Works well, especially when…

- Large number of features
- Little or no knowledge of data dictionary
- File formats (CSV for instance), don't contain type info
- Non-conforming rows
- Poor semantics
  - Think "0001" vs 0001

Data type imputation is as important as missing value imputation

```
[35] schema

feature {
  name: "payment_type"
  value_count {
    min: 1
    max: 1
  }
  type: INT
  presence {
    min_fraction: 1.0
    min_count: 1
  }
}
feature {
  name: "trip_distance"
  value_count {
    min: 1
    max: 1
  }
  type: FLOAT
  presence {
    min_fraction: 1.0
    min_count: 1
  }
}
feature {
  name: "pickup_day_of_week"
  value_count {
    min: 1
    max: 1
  }
  type: INT
  presence {
    min_fraction: 1.0
    min_count: 1
  }
}
```

```
schema_dir = infer_schema.outputs['output'].get()[0].uri
schema_path = os.path.join(schema_dir, 'schema.pbtxt')

# Load and visualize the generated schema
schema = tfdv.load_schema_text(schema_path)
tfdv.display_schema(schema)
```

Feature name	Type	Presence	Valency	Domain
'payment_type'	INT	required	single	-
'trip_distance'	FLOAT	required	single	-
'pickup_day_of_week'	INT	required	single	-
'fare_amount'	FLOAT	required	single	-
'trip_type'	INT	required	single	-
'dropoff_day_of_week'	INT	required	single	-
'dropoff_hour'	INT	required	single	-
'pickup_hour'	INT	required	single	-
'vendor_id'	INT	required	single	-
'dropoff_month'	INT	required	single	-
'pickup_month'	INT	required	single	-
'tip_amount'	FLOAT	required	single	-
'passenger_count'	INT	required	single	-

ExampleValidator TFX Pipeline Component

Identifies anomalies (skew, drift, schema validation) in training & serving data.

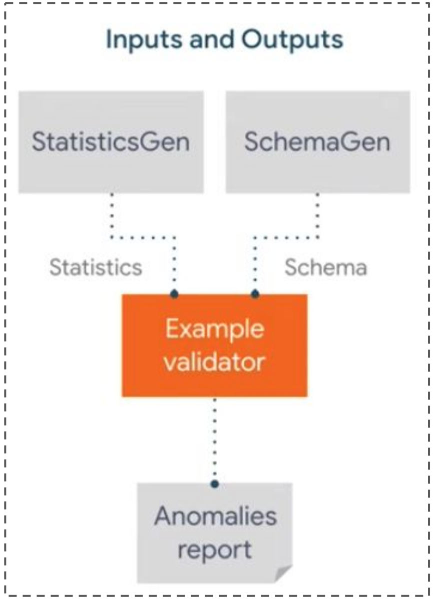
- Input: A schema from a SchemaGen component, and statistics from a StatisticsGen component
- Output: Validation results
- Here's how you can call it...

```
validate_stats = ExampleValidator(
    stats=statistics_gen.outputs['output'],
    schema=infer_schema.outputs['output'])
context.run(validate_stats)

validation_dir = validate_stats.outputs['output'].get()[0].uri
anomalies_path = os.path.join(validation_dir, 'anomalies.pbtxt')


anomalies = tfdv.load_anomalies_text(anomalies_path)
tfdv.display_anomalies(anomalies)
```

No anomalies found.



source

Schema Skew

Training schema != Serving data schema

Expected deviations (like target) should be specified via *Environments* field in schema.

- *default_environment*, *in_environment* or *not_in_environment*

Scenario:

- Suppose you found a new feature which improved offline model performance
- But after model was deployed to production, online model performance was poor
- After debugging, you discovered that the new feature added wasn’t available during serving time

Feature Skew

Feature values during *__Training__* differ compared to feature values during *__Serving__*.

Example:

Scenario 1: 3rd Party Dependency

- Data coming from 3rd party systems may & will likely differ between time you train model compared to time model is in production for serving.

Scenario 2: Different code paths between Training vs Serving

- During model training, you're experimenting with various feature engineering methods & algo's
- Unless you have a reproducible & repeatable pipeline which is identical, feature skew will be present.

Distribution Skew

Feature value distribution during *Training* differ compared to *Serving*.

Example:

Scenario 1: Often when you're starting a new product, there is a lack of data available for ML.

- You acquire (purchase, crawl etc) data to build initial mode
- After you launch product, the data you collect is unlikely to match distribution of initial training corpus

Scenario 2: You're dealing with a large dataset (does not fit into RAM)

- You choose a faulty sampling technique to sub-sample the data

Freeze Schema

Useful for later usage, serving model via TF Serving.
Human readable, useful for inspection.
Easy to compare & validate against changes in the future.

Freeze Schema

We want to persist our schema so that it can be used by other team members as well as the rest of the TensorFlow Transform & Serving pipeline.

In [17]:

```
from tensorflow.python.lib.io import file_io
from google.protobuf import text_format

file_io.recursive_create_dir(OUTPUT_DIR)
schema_file = os.path.join(OUTPUT_DIR, 'schema.pbtxt')
tfdv.write_schema_text(schema, schema_file)

!cat {schema_file}
```

```
feature {
  name: "trip_distance"
  value_count {
    min: 1
    max: 1
  }
  type: FLOAT
  presence {
    min_fraction: 1.0
    min_count: 1
  }
}
feature {
  name: "pickup_day_of_week"
  value_count {
    min: 1
    max: 1
  }
  type: INT
  presence {
```

Environments

Environments allow you to define slightly different schemas for each use case

- Label will not exist in for serving set

```
# all features are by default in both TRAINING, EVAL and SERVING environments
schema.default_environment.append('TRAINING')
schema.default_environment.append('EVAL')
schema.default_environment.append('SERVING')

# indicate that 'fare_amount' feature is not in SERVING environment.
tfdv.get_feature(schema, 'fare_amount').not_in_environment.append('SERVING')

serving_anomalies_with_env = tfdv.validate_statistics(
    serving_stats, schema, environment='SERVING')

tfdv.display_anomalies(serving_anomalies_with_env)
```

No anomalies found.

TFDV Knowledge Check

Q: Suppose your model begins to perform poorly in a production environment, due to lack of availability of a feature coming from a third party provider. Which type of drift best explains the root cause?

a) Schema Drift
b) Distribution Drift
c) Feature Drift
d) a & b
e) b & c

TensorFlow Data Transform

Performs Feature Engineering

TensorFlow Transform

Apache Beam and TFX

- Framework for running batch & streaming data processing jobs
- TFX libraries use Beam for running jobs locally or on compute clusters
  - Direct runner (single node dev)
  - Runners in large deployments orchestrated via Kubernetes or Apache Mesos
- TFX uses Beam Python API
  - Python 2.x & 3.x available
- Review Transform API
- TFX example on Kubeflow Pipelines
- Orchestration via Airflow, Kubeflow

Feature Engineering @ Scale	Transformations
Transform data before it goes into a model. Output is exported as a TensorFlow graph, used for both training & serving.	tft.scale_by_min_max(), tft.scale_to_0_1(), tft.scale_to_z_score()
Create feature embeddings & enriching text features	tft.tfidf(), tft.ngrams(), tft.hash_strings()
Builds transformations into TF graph for your model, so same transformations are applied for train & serving.	Convert strings to integers by generating a vocabulary over all input values
Vocabulary generation	tft.compute_and_apply_vocabulary(), tft.string_to_int()
Normalize values & Bucketization	tft.bucketize()

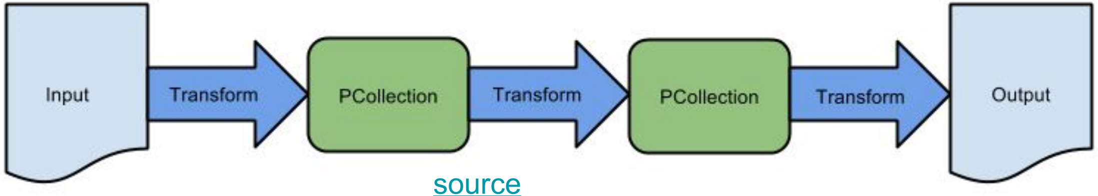
Apache Beam (Intro)

- Open source SDK to help you build data pipelines (batch, streaming)
- Not dependent on a specific compute engine (Spark, Cloud Dataflow)
- You can run via *Direct* or *Distributed* Runner
- Python 2.7 for now, Python 3 coming soon

3 central concepts:

- ***Pipeline***: end to end workflow of your data pipeline (DAG)
- ***PCollection***: distributed dataset (think RDD), abstraction which Beam uses to transfer data between ***PTransforms***
- ***PTransform:*** process which operates on input ***PCollection*** & produces output ***PCollection***

Programming Guide



source

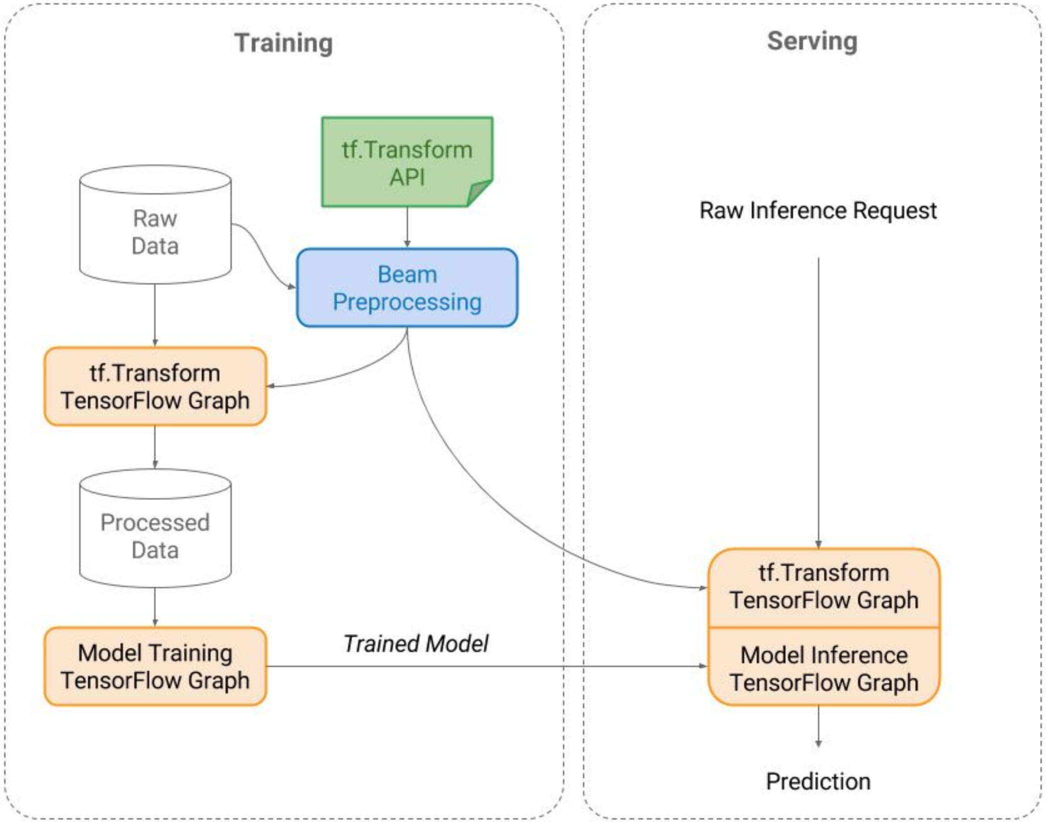
Apache Beam (Intro)

Example:

[Output PCollection] = ([First Input PCollection] | [First Transform] | [Second Transform])

- “|” operator equivalent to apply method
- Data is represented as PCollection which is immutable
- Transforms can be chained
  - *AnalyzeDataset*: Takes a preprocessing_fn and computes the relevant statistics
  - *TransformDataset*: Applies the transformation computed by transforming a Dataset
  - *AnalyzeAndTransformDataset*: Combination of AnalyzeDataset and TransformDataset
- Beam Readers & Writers
  - Variety of built in I/O readers & writers available here
- Guide to various runners available

TensorFlow Transform Visualized



[source]

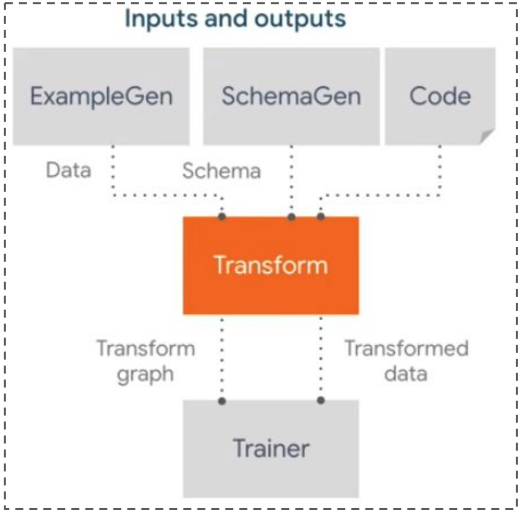
Transform TFX Pipeline Component

Performs feature engineering

- If you do it within TensorFlow Transform, transforms become part of the TensorFlow graph
  - Eliminates training vs serving skew
- Input: Data formatted in tf.Examples emitted from ExampleGen using schema generated by SchemaGen
- Output: SavedModel to a Trainer component
- Transformations implemented in Beam under the hood. Performs full pass over your data.
- Here's how to call it...

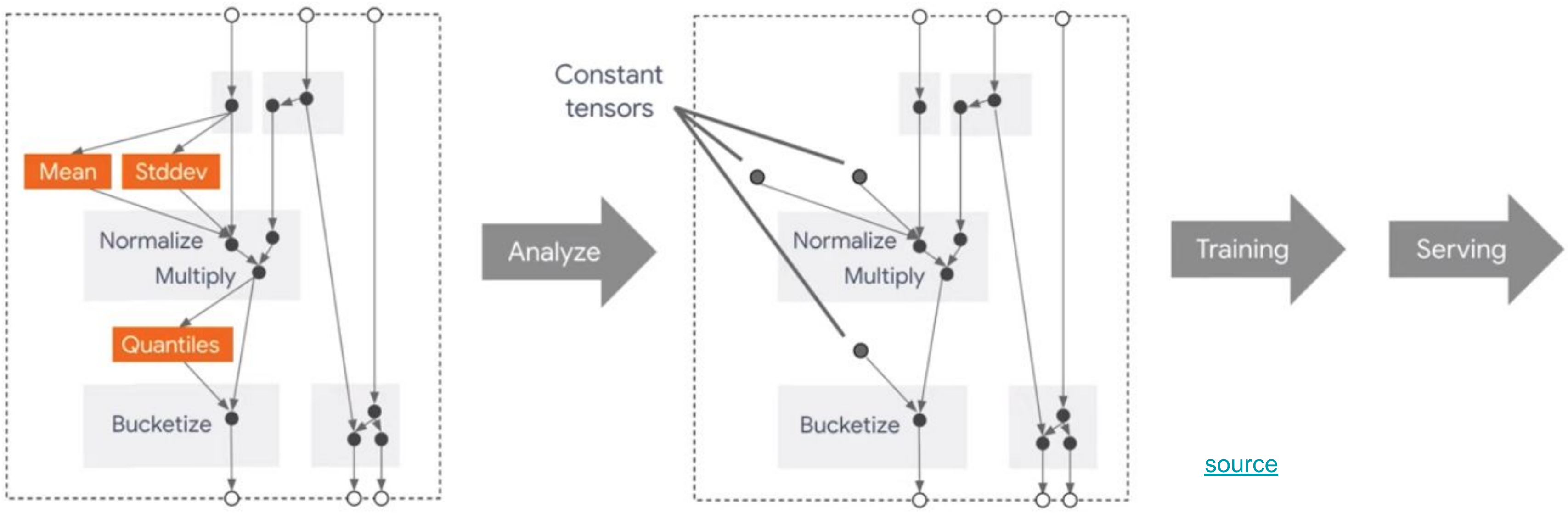
```
transform_training = components.Transform(
    input_data=examples_gen.outputs.training_examples,
    schema=infer_schema.outputs.output,
    module_file=taxi_pipeline_utils,
    name='transform-training')

transform_eval = components.Transform(
    input_data=examples_gen.outputs.eval_examples,
    schema=infer_schema.outputs.output,
    transform_dir=transform_training.outputs.output,
    name='transform-eval')
```



source

Transform Component Continued



source

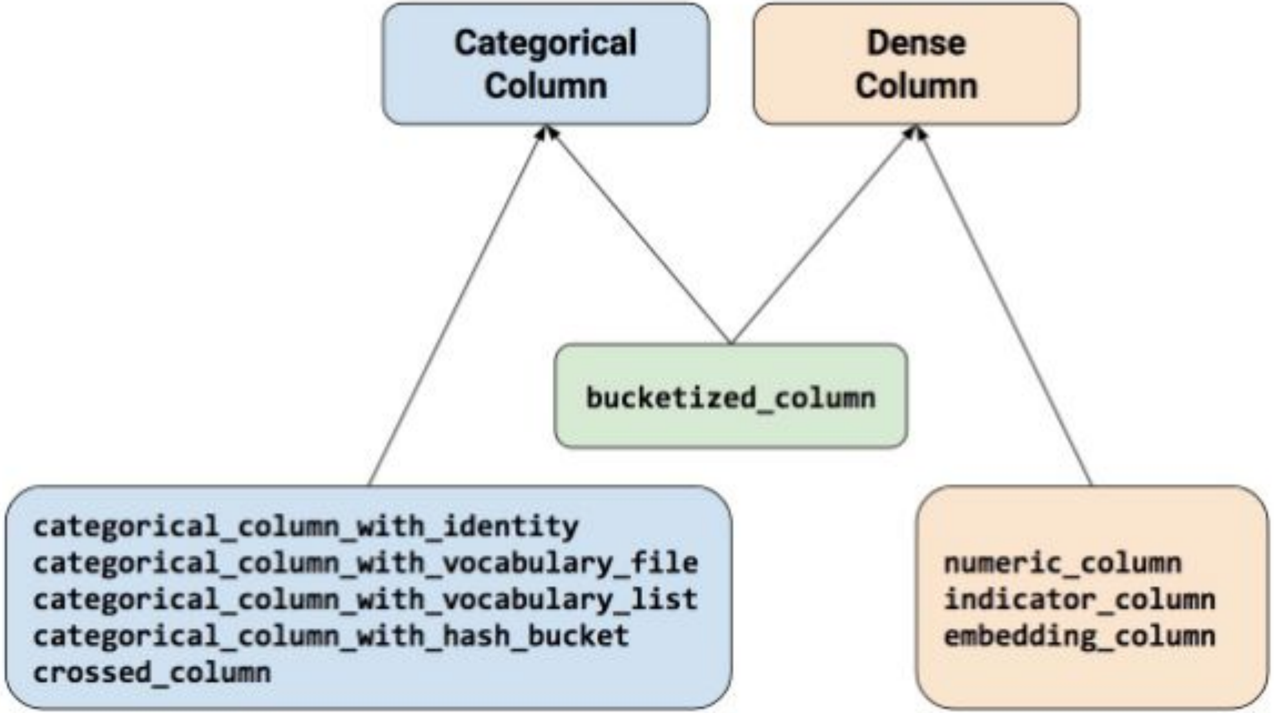
Transform makes full path over data to create two different kinds of results:

- For mean, std deviation & quantiles (same for all examples), it outputs constant tensors
- For normalizing a value (different for different examples), it will output tensorflow ops
- Final output is a Tensorflow graph with constants & ops
- Use for both training & serving

Support for FeatureColumn out of the box

Method	Description
Embedding	Convert sparse features (high dimensional space) into dense features (lower dimensional space) by using an embedding function & a user specified embedding size.
Categorical with known Vocabulary	Convert non-numerics (strings) into integers by creating a vocabulary that maps each unique value to an ID number.
Normalize numerical values	Convert numeric features so they fall into a range of values.
Bucketization of numerical values	Convert continuous-valued features into a categorical features by assigning values to discrete buckets.
Enhance text features	Create features from raw text data like tokens, n-grams, entities and sentiment.

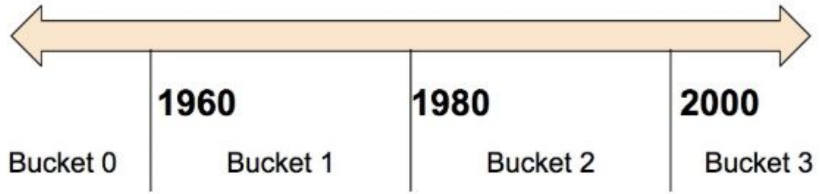
Tensorflow FeatureColumns Part 1



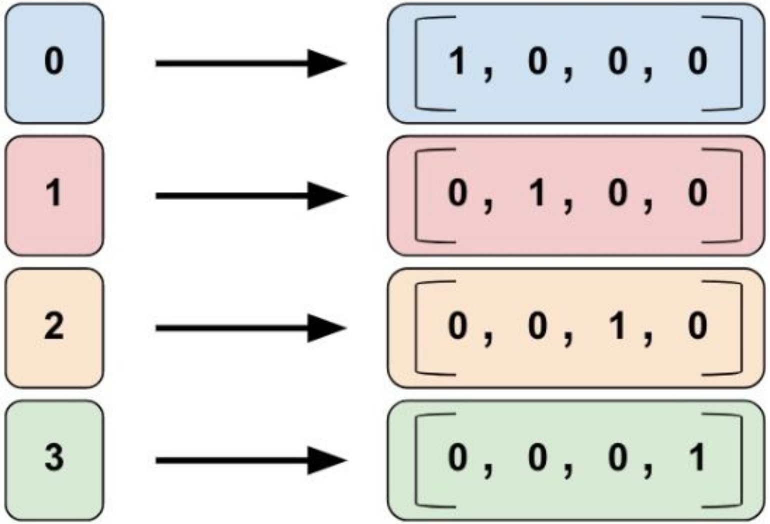
Feature column methods fall into two main categories and one hybrid category.

source

Tensorflow FeatureColumns Part 2



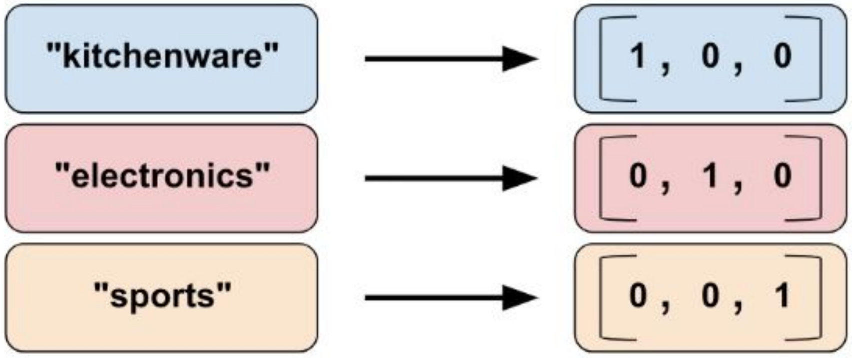
Dividing year data into four buckets.



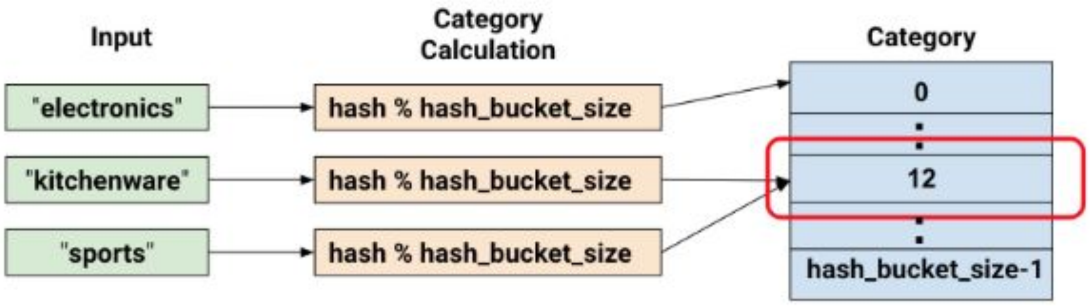
A categorical identity column mapping. Note that this is a one-hot encoding, not a binary numerical encoding.

source

Tensorflow FeatureColumns Part 3



Mapping string values to vocabulary columns.



Representing data with hash buckets.

source

What Do I Need to Do?

As a TFX end user, you need to create a **preprocessing_fn(...)** method.

This method should:

- Input to **preprocessing_fn(...)** is determined by a Schema (tf.int64, tf.float32, tf.string)
- Define a series of functions that manipulate the input dictionary of tensors to produce output of dictionary tensors
- You can use methods available in [Transform API] or any regular Tensorflow functions
  - For numerical feature sets, you may want to apply **tft.scale_to_z_score**
  - For categorical feature sets with known vocabulary, you may want to apply **tft.compute_and_apply_vocabulary**
  - For continuous numerical value feature sets, you may want to apply **tft.bucketize**

Pre-processing with tf.Transform

tf.Transform works on data at any size!

tft.min is an example of one of many analyzers which can run over your entire dataset.

Transform raw training data using a preprocessing pipeline

- Scale numeric data
- Replaces missing values
- Bucketize feature values

```
def preprocessing_fn(inputs):
  """tf.transform's callback function for preprocessing inputs.

  Args:
    inputs: map from feature keys to raw not-yet-transformed features.

  Returns:
    Map from string feature key to transformed feature operations.
  """
  outputs = {}
  for key in _DENSE_FLOAT_FEATURE_KEYS:
    # Preserve this feature as a dense float, setting nan's to the mean.
    outputs[_transformed_name(key)] = tft.scale_to_z_score(
        _fill_in_missing(inputs[key]))

  for key in _VOCAB_FEATURE_KEYS:
    # Build a vocabulary for this feature.
    outputs[_transformed_name(key)] = tft.compute_and_apply_vocabulary(
        _fill_in_missing(inputs[key]),
        top_k=_VOCAB_SIZE,
        num_oov_buckets=_OOV_SIZE)

  for key in _BUCKET_FEATURE_KEYS:
    outputs[_transformed_name(key)] = tft.bucketize(
        _fill_in_missing(inputs[key]), _FEATURE_BUCKET_COUNT)

  for key in _CATEGORICAL_FEATURE_KEYS:
    outputs[_transformed_name(key)] = _fill_in_missing(inputs[key])

  fare_amount = _fill_in_missing(inputs[_LABEL_KEY])

  outputs[_transformed_name(_LABEL_KEY)] = fare_amount
  return outputs
```

Iterate over feature set

Note usage of tft API

User defined function to impute missing values

Using Pre-processed data to train a model

Trainer TFX pipeline component trains a TensorFlow model.

- Input:
  - tf.Examples transformed by a Transform component
  - Eval tf.Examples transformed by a Transform component
  - Data schema create by a SchemaGen component
- Output
  - SavedModel and an EvalSavedModel
- Strongly recommend using TF Estimator API for performance & efficient models (**trainer_fn** for details)
- Here's how to call it...

```
trainer = Trainer(
    module_file=ny_taxi_module,
    transformed_examples=transform.outputs['transformed_examples'],
    schema=infer_schema.outputs['output'],
    transform_output=transform.outputs['transform_output'],
    train_args=trainer_pb2.TrainArgs(num_steps=10000),
    eval_args=trainer_pb2.EvalArgs(num_steps=5000))
context.run(trainer)
```

TFT Exercises

Q: Tensorflow Transform code needs to be re-written based on the size of the dataset I need to process?

a) True
b) False

TFT Exercises

Q: Tensorflow Transform requires that I run on Google Cloud Platform?

a) True
b) False

Next Steps:

- Check out the readme doc for links to TFDV & TFT notebooks at course repo page