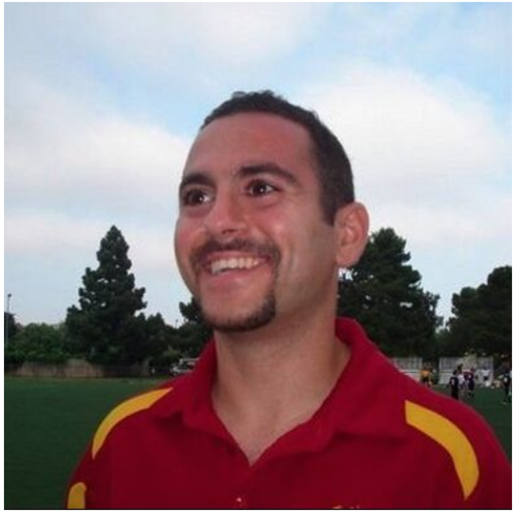
TensorFlow Extended Part 1

Data Validation & Transform

Armen Donigian

Who am I?



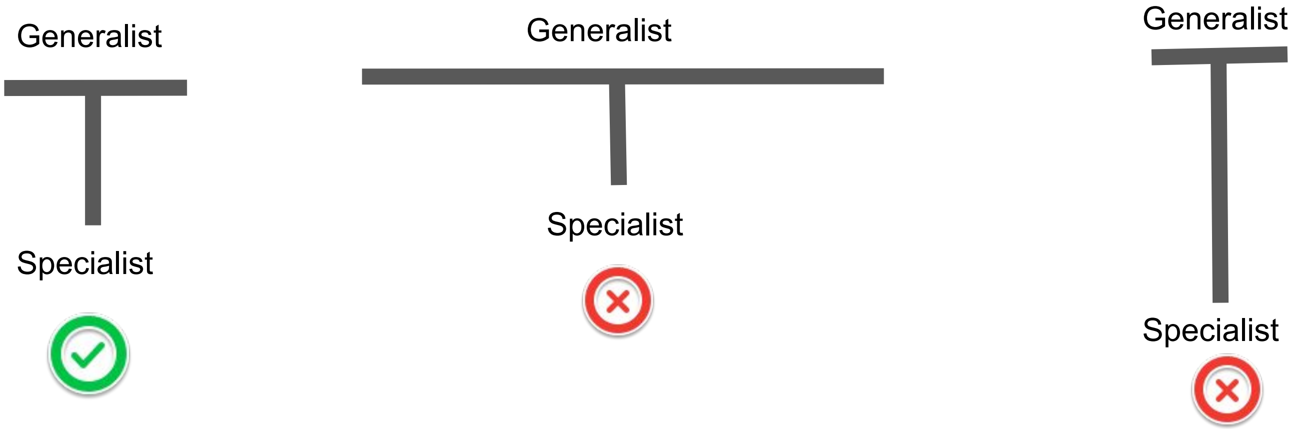
- Computer Science Undergrad degree @UCLA
- Computer Science Grad degree @USC
- 15+ years experience as Software & Data Engineer
- Computer Science Instructor
- Mentor @Udacity Deep Learning Nanodegree
- Real-time wagering algorithms @GamePlayerNetwork
- Differential GPS corrections @Jet Propulsion Laboratory, 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: Part 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

Day 1: Part 2

- **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

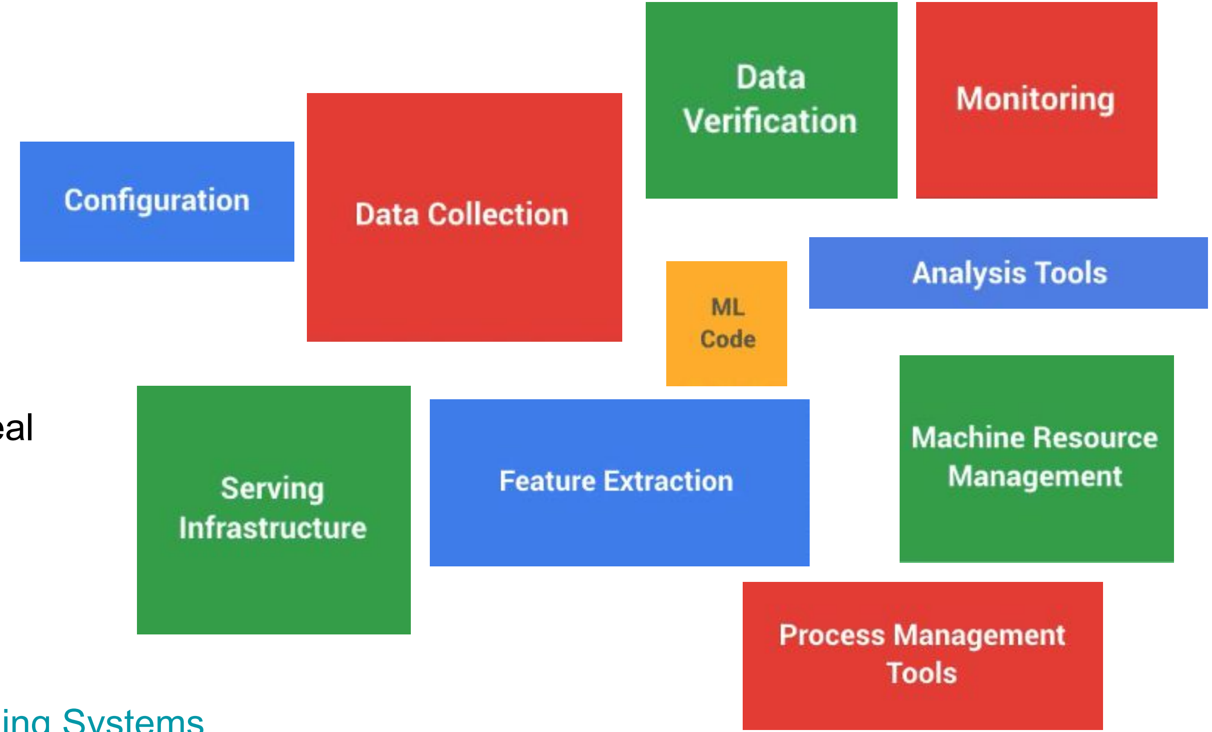
TensorFlow Extended (TFX)

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

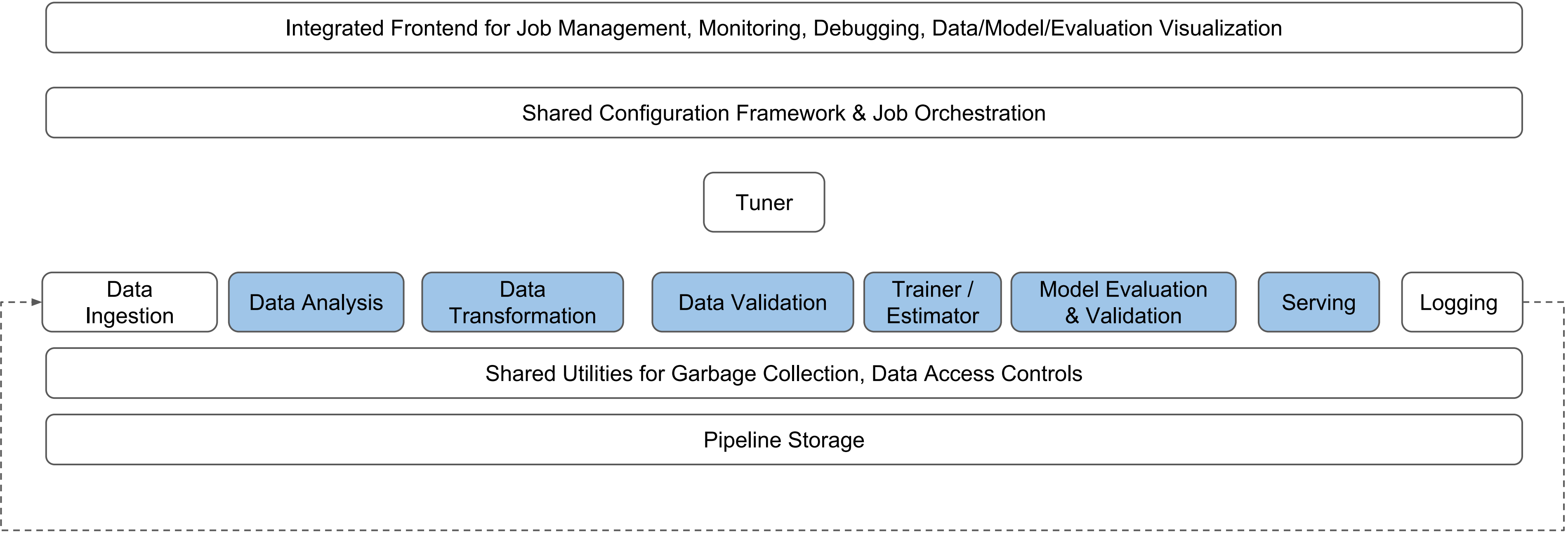


Takeaway: Doing machine learning in real world is HARD!

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

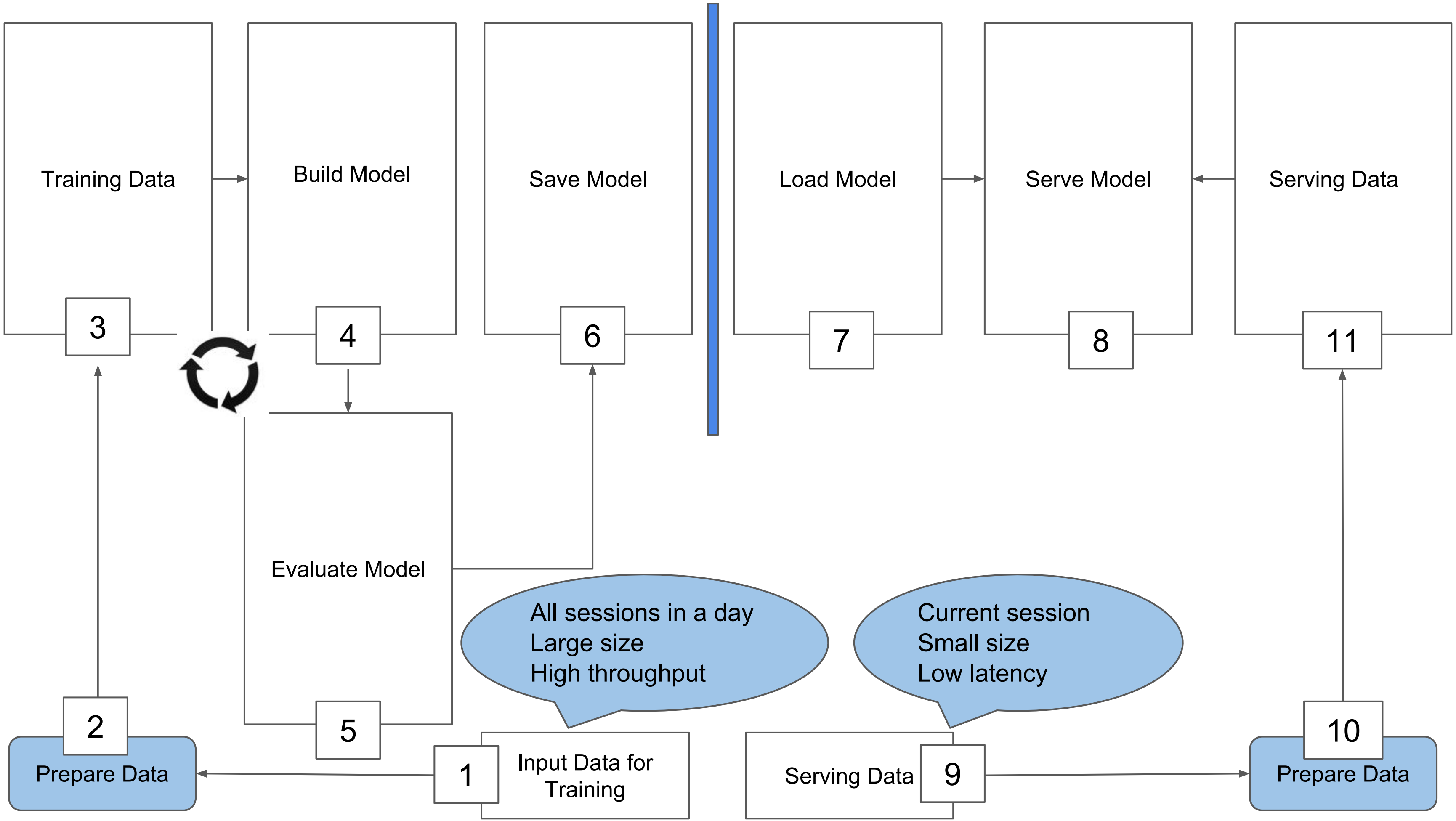
Open Source

Not Public Yet

Link to TFX paper

Train / Serving Data Flows

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



What Could Go Wrong...

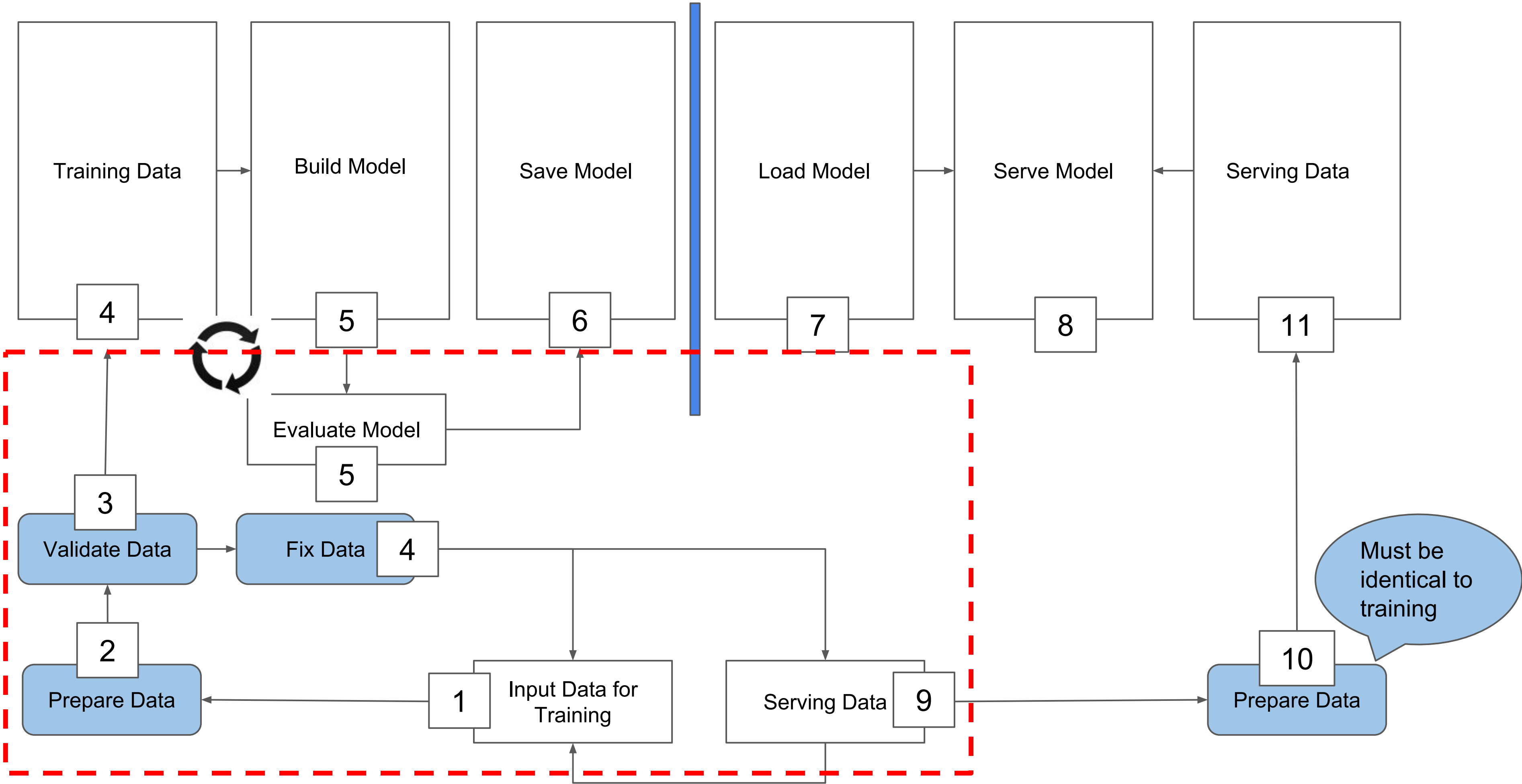
In no particular order…

- 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?

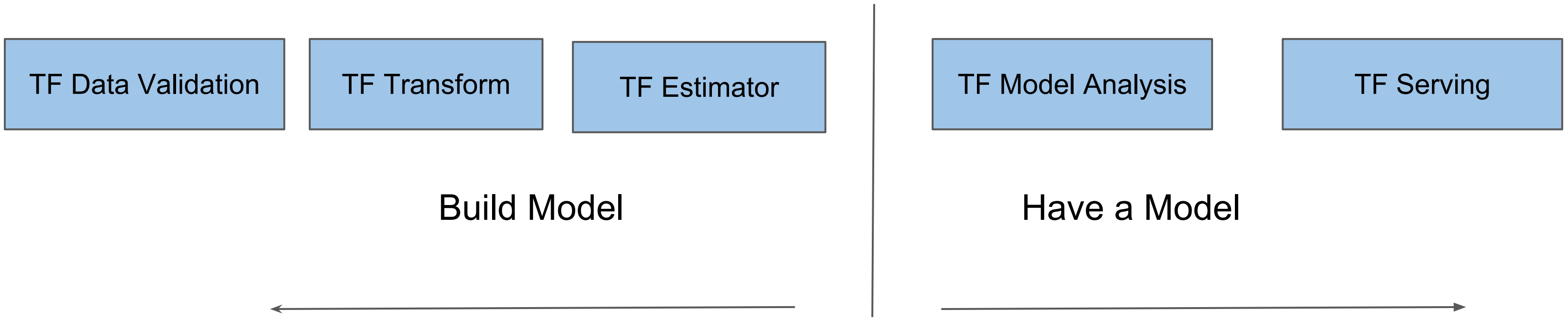
Click links above to find related research papers & projects.

Train / Serving Data Flows

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



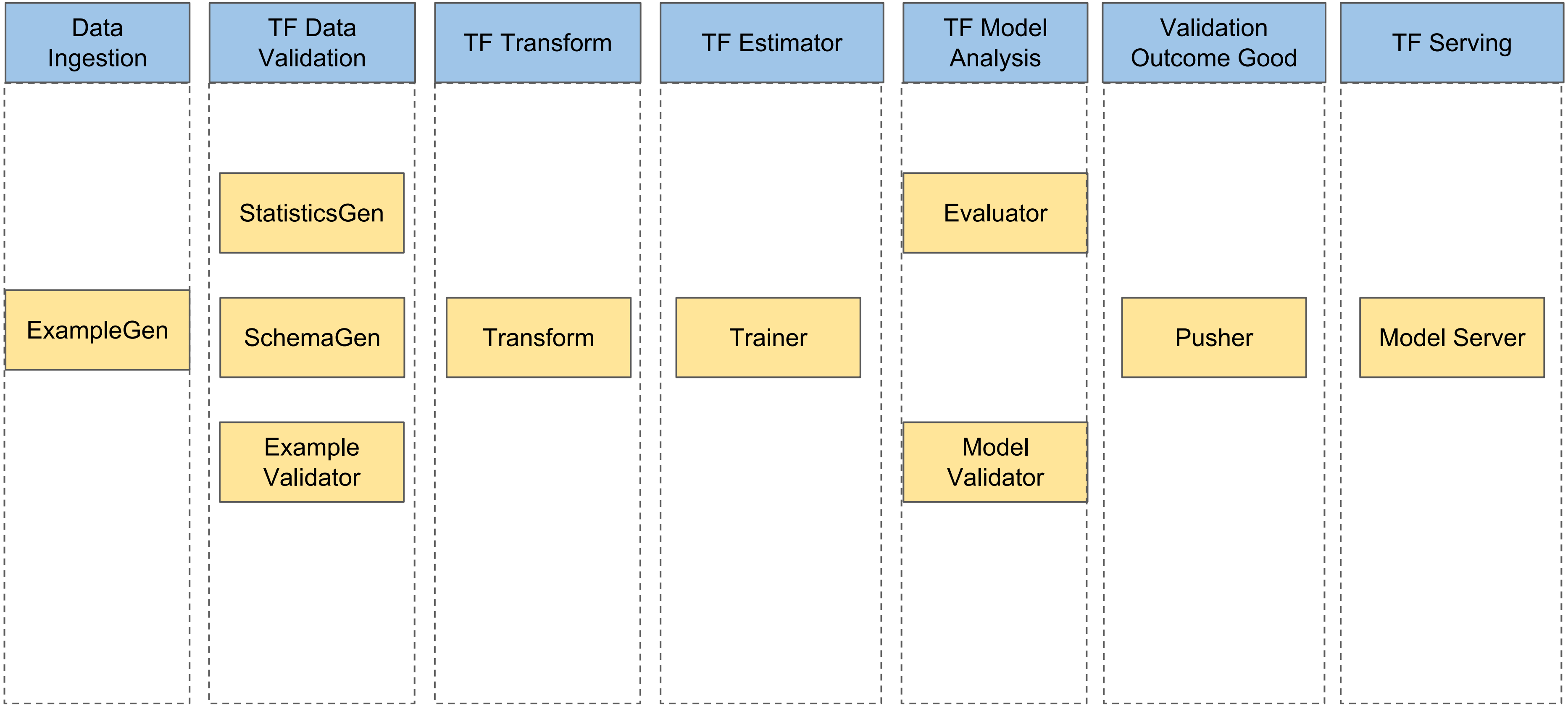
TFX Pipeline



Components API is also available...

- 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


Data Ingestion
ExampleGen
TF Data Validation
StatisticsGen
SchemaGen
Example Validator
TF Transform
Transform
TF Estimator
Trainer
TF Model Analysis
Evaluator
Model Validator
Validation Outcome Good
Pusher
TF Serving
Model Server

TensorFlow Data Validation

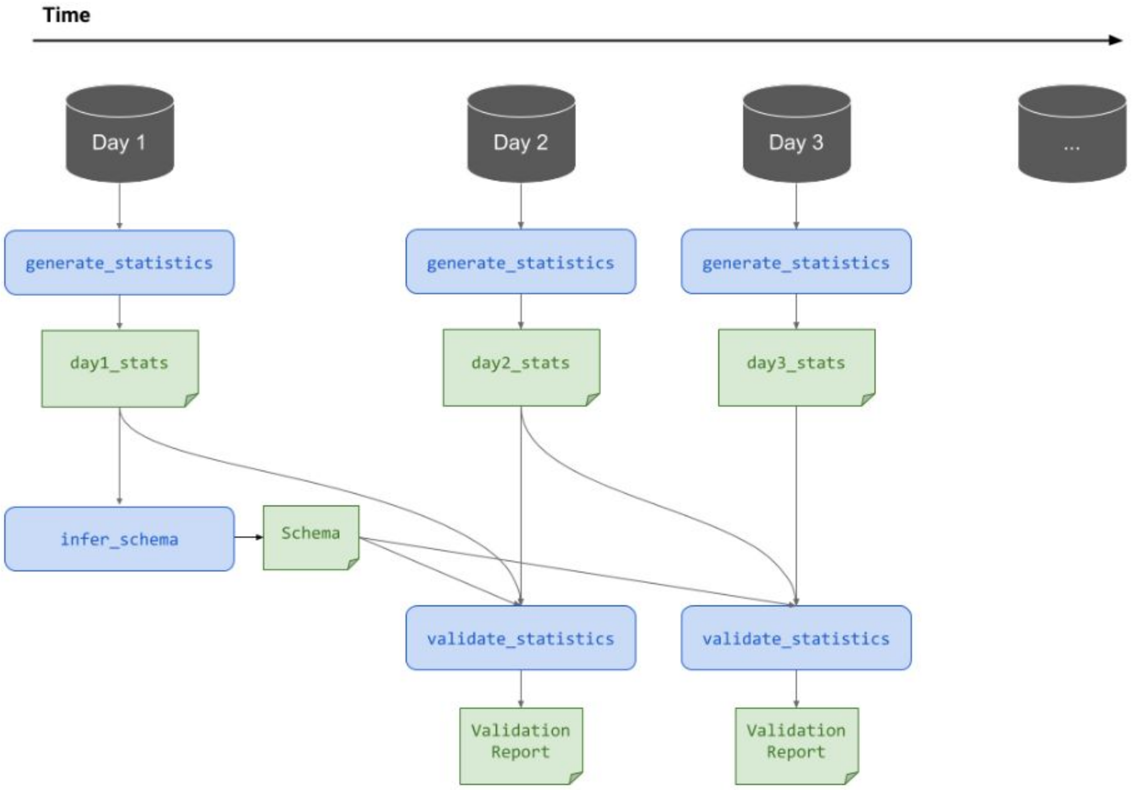
TensorFlow Data Validation Overview

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



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

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

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

```
# infer schema from training data
schema = tfdv.infer_schema(statistics=train_stats, infer_feature_shape=False)
tfdv.display_schema(schema=schema)
```

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

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}
```

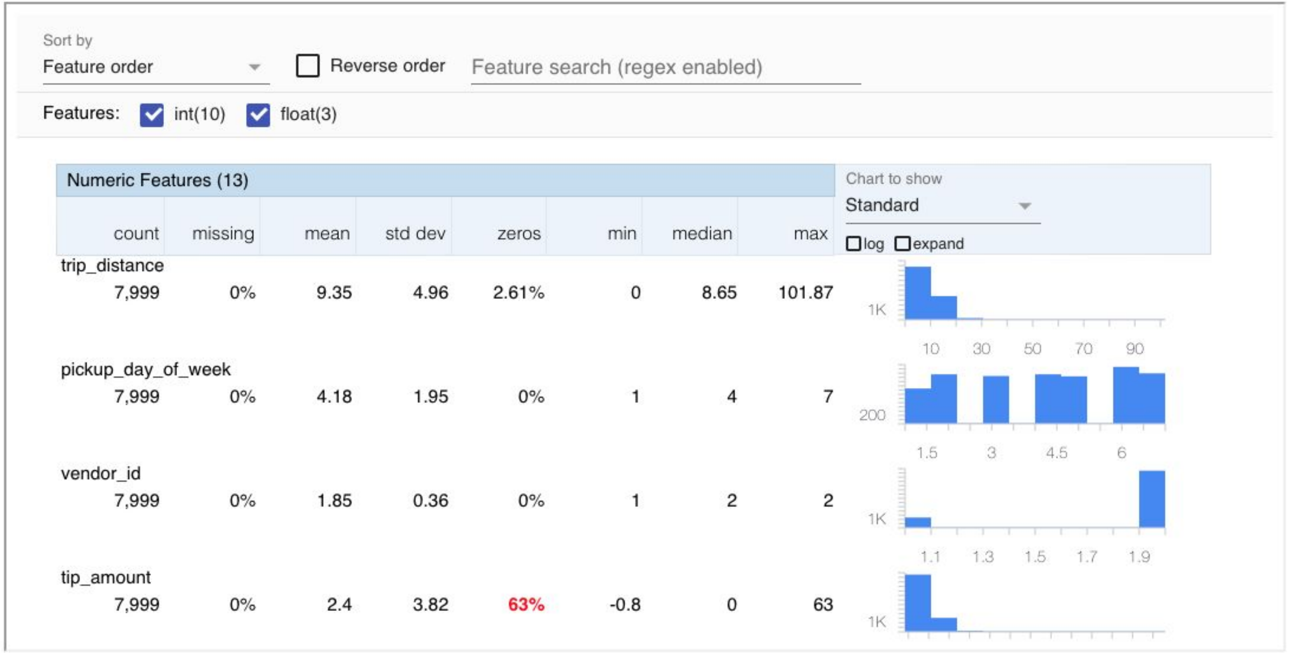
```
feature {
  name: "trip_distance"
  value_count {
    min: 1
    max: 1
  }
  type: FLOAT
  presence {
    min_fraction: 1.0
    min_count: 1
  }
}
feature {
  name: "pickup_day_of_week"
  value_count {
    min: 1
    max: 1
  }
  type: INT
  presence {
```

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)
```



	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)

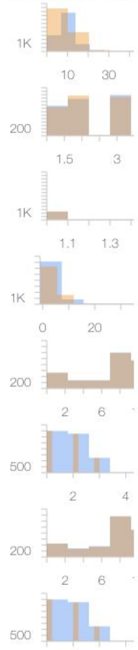
Features: int(10) float(3)

EVAL_DATA.SET TRAIN_DATASET

Numeric Features (13)

Chart to show
Standard
log expand per

		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



Data Anomalies

Sanity checks…

- Identify constraints & check whether dataset violates it or not
- Environments allow you to define slightly different schemas for each use case
    - Label will not exist in for serving set

```
# update the schema based on the observed anomalies.
vendor_id = tfdv.get_feature(schema, 'vendor_id')
# we want feature vendor_id to be populated in at least 50% of the examples
vendor_id.presence.min_fraction = 0.5

# validate eval stats after updating the schema
updated_anomalies = tfdv.validate_statistics(eval_stats, schema)
tfdv.display_anomalies(updated_anomalies)
```

No anomalies found.

```
# 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 Exercises

TensorFlow Data Transform

Performs Feature Engineering

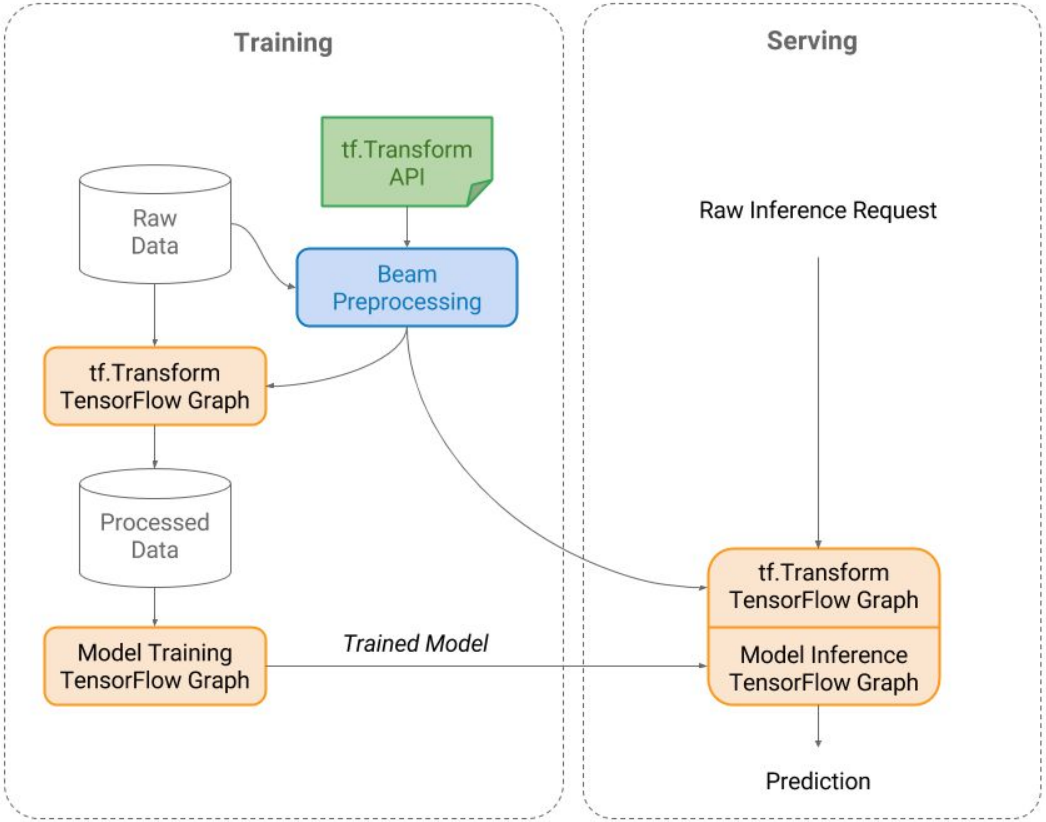
TensorFlow Transform

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 now, 3 coming soon
- TFX example on Kubeflow Pipelines

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()

TensorFlow Transform Visualized



[source]

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.

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

List of column names which contain floats

Note usages of tft.*

List of column names which contain values which can be split into bins

User defined function...
Replace missing values

Transform using Apache Beam Pipeline

Steps of our pipeline:

1) Read in data using CSV reader

2) Transform raw training data using a preprocessing pipeline
    - Scale numeric data
    - Replaces missing values
    - Bucketize feature values

3) Shuffling data before to improve training

4) Output result as TFRecords

```
schema = read_schema(schema_file)
raw_feature_spec = get_raw_feature_spec(schema)
raw_schema = dataset_schema.from_feature_spec(raw_feature_spec)
raw_data_metadata = dataset_metadata.DatasetMetadata(raw_schema)

with beam.Pipeline(argv=pipeline_args) as pipeline:
    with tft_beam.Context(temp_dir=working_dir):
        if input_handle.lower().endswith('csv'):
            # read raw train data from csv file
            csv_coder = make_csv_coder(schema)
            raw_data = (
                pipeline
                | 'ReadFromText' >> beam.io.ReadFromText(
                    input_handle, skip_header_lines=1)
                | 'ParseCSV' >> beam.Map(csv_coder.decode))
        if transform_dir is None:
            # analyze and transform raw training data to produced transform_fn
            transform_fn = (
                (raw_data, raw_data_metadata)
                | ('Analyze' >> tft_beam.AnalyzeDataset(preprocessing)))
            # write transform_fn as tf.graph
            _ = (
                transform_fn
                | ('WriteTransformFn' >>
                   tft_beam.WriteTransformFn(working_dir)))
        else:
            transform_fn = pipeline | tft_beam.ReadTransformFn(transform_dir)

        # shuffling the data before to improve training
        shuffled_data = raw_data | 'RandomizeData' >> beam.transforms.Reshuffle()

        # get data and schema separately from the raw_data_metadata
        (transformed_data, transformed_metadata) = (
          ((shuffled_data, raw_data_metadata), transform_fn)
          | 'Transform' >> tft_beam.TransformDataset())

        # write transformed train data to sink
        coder = example_proto_coder.ExampleProtoCoder(transformed_metadata.schema)
        _ = (
          transformed_data
          | 'SerializeExamples' >> beam.Map(coder.encode)
          | 'WriteExamples' >> beam.io.WriteToTFRecord(
              os.path.join(working_dir, outfile_prefix), file_name_suffix='.gz')
        )
```

Using Pre-processed data to train a model

tf.Transform prevents train/serve skew!

We do this by creating input functions

Training input function contains the labels… while serving input function does not.

```
def input_fn(filenames, tf_transform_dir, batch_size=200):
    """Generates features and labels for training or evaluation.

    Args:
    filenames: [str] list of CSV files to read data from.
    tf_transform_dir: directory in which the tf-transform model was written
      during the preprocessing step.
    batch_size: int First dimension size of the Tensors returned by input_fn

    Returns:
    A (features, indices) tuple where features is a dictionary of
      Tensors, and indices is a single Tensor of label indices.
    """
    metadata_dir = os.path.join(tf_transform_dir,
                                transform_fn_io.TRANSFORMED_METADATA_DIR)
    transformed_metadata = metadata_io.read_metadata(metadata_dir)
    transformed_feature_spec = transformed_metadata.schema.as_feature_spec()

    transformed_features = tf.contrib.learn.io.read_batch_features(
      filenames, batch_size, transformed_feature_spec, reader=_gzip_reader_fn)

    # we pop the label because we do not want to use it as a feature while we're
    # training.
    return transformed_features, transformed_features.pop(
      transformed_name(LABEL_KEY))
```

Put it all together...

Train, Evaluate, Transform & Export model.

```
def train_and_maybe_evaluate(hparams):
    """Run the training and evaluate using the high level API.

    Args:
    hparams: Holds hyperparameters used to train the model as name/value pairs.

    Returns:
    The estimator that was used for training (and maybe eval)
    """
    schema = read_schema(hparams.schema_file)

    train_input = lambda: input_fn(
      hparams.train_files,
      hparams.tf_transform_dir,
      batch_size=TRAIN_BATCH_SIZE
    )

    eval_input = lambda: input_fn(
      hparams.eval_files,
      hparams.tf_transform_dir,
      batch_size=EVAL_BATCH_SIZE
    )

    train_spec = tf.estimator.TrainSpec(
      train_input, max_steps=hparams.train_steps)

    serving_receiver_fn = lambda: example_serving_receiver_fn(
      hparams.tf_transform_dir, schema)
```

Using input function defined earlier

High level API which simplifies train evaluate, predict & serving of TF models.

```
    serving_receiver_fn = lambda: example_serving_receiver_fn(
      hparams.tf_transform_dir, schema)

    exporter = tf.estimator.FinalExporter('nyc-taxi', serving_receiver_fn)
    eval_spec = tf.estimator.EvalSpec(
      eval_input,
      steps=hparams.eval_steps,
      exporters=[exporter],
      name='nyc-taxi-eval')

    run_config = tf.estimator.RunConfig(
      save_checkpoints_steps=999, keep_checkpoint_max=1)

    serving_model_dir = os.path.join(hparams.output_dir, SERVING_MODEL_DIR)
    run_config = run_config.replace(model_dir=serving_model_dir)

    estimator = build_estimator(
      hparams.tf_transform_dir,

      # Construct layers sizes with exponetial decay
      hidden_units=[
          max(2, int(FIRST_DNN_LAYER_SIZE * DNN_DECAY_FACTOR**i))
          for i in range(NUM_DNN_LAYERS)
      ],
      config=run_config)
    tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)
    return estimator
```

TFT Exercises

Next Steps:

- Work through TFDV & TFT Notebooks