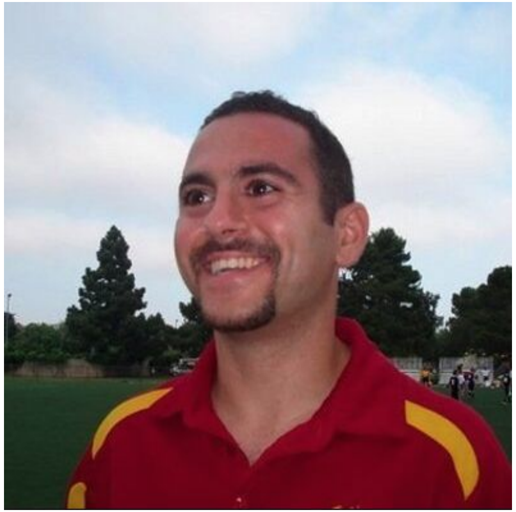
TensorFlow Extended Part 2

Model Build, Analysis & Serving

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 & Discovery
- Available for Consulting (donigian@LevelUpInference.com)

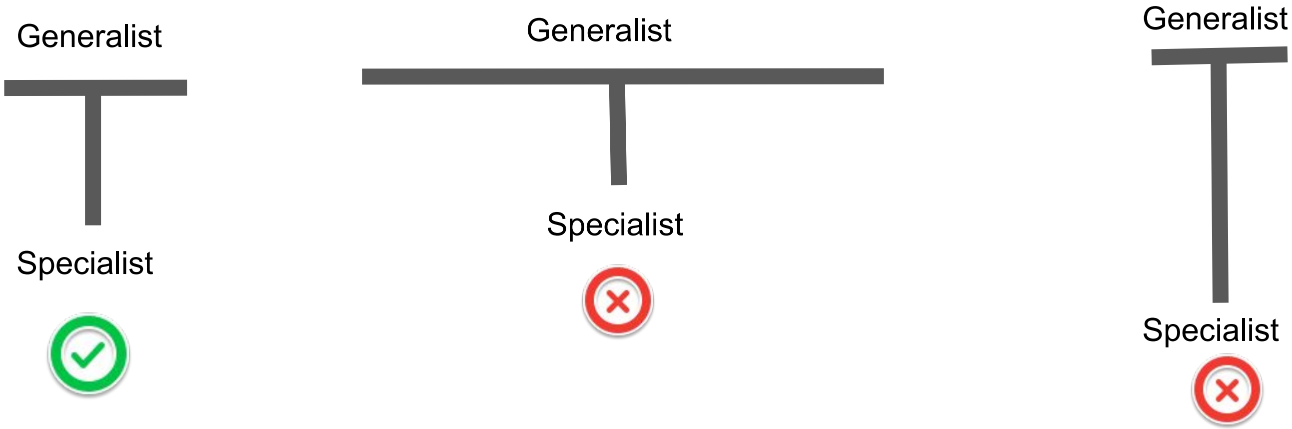


Goals, Breadth vs Depth...

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

- **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 Estimator Overview (35 mins)**
  a. What is TensorFlow & Why Should You Care?
  b. What is TF Estimator?
  c. How to train a model using TF Estimator?
  d. Dataset Overview
  e. TF Estimator notebook demo

  10 min Break

- **TensorFlow Model Analysis Overview (40 mins)**
  a. What is it & why should you care?
  b. TFMA API Overview
  c. TFMA Usage
  d. TFMA notebook demo

  10 min Break

- **Tensorflow Serving (45 mins)**
  a. What is it & why should you care?
  b. TF Serving Intro
  c. TF Serving w/ Docker notebook demo
     i. CPU / GPU / TPU
  d. TF Model Server REST API

TensorFlow Extended Overview

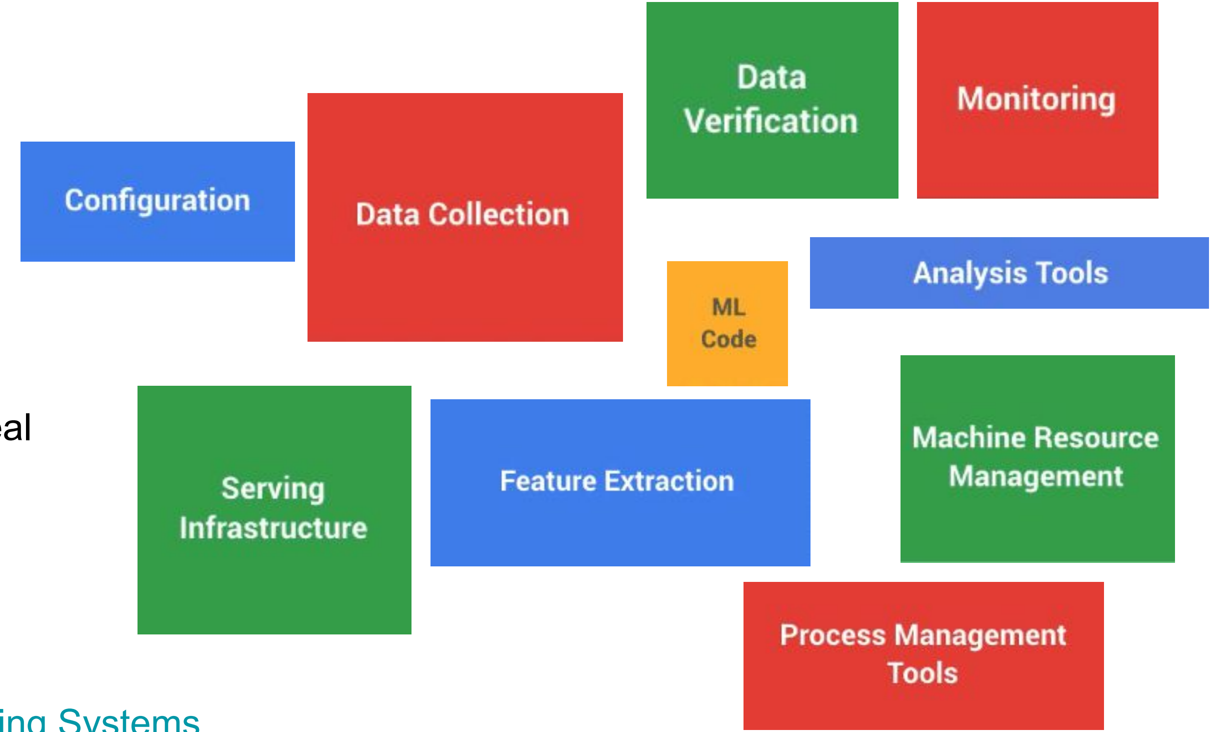
TensorFlow Extended (TFX)

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

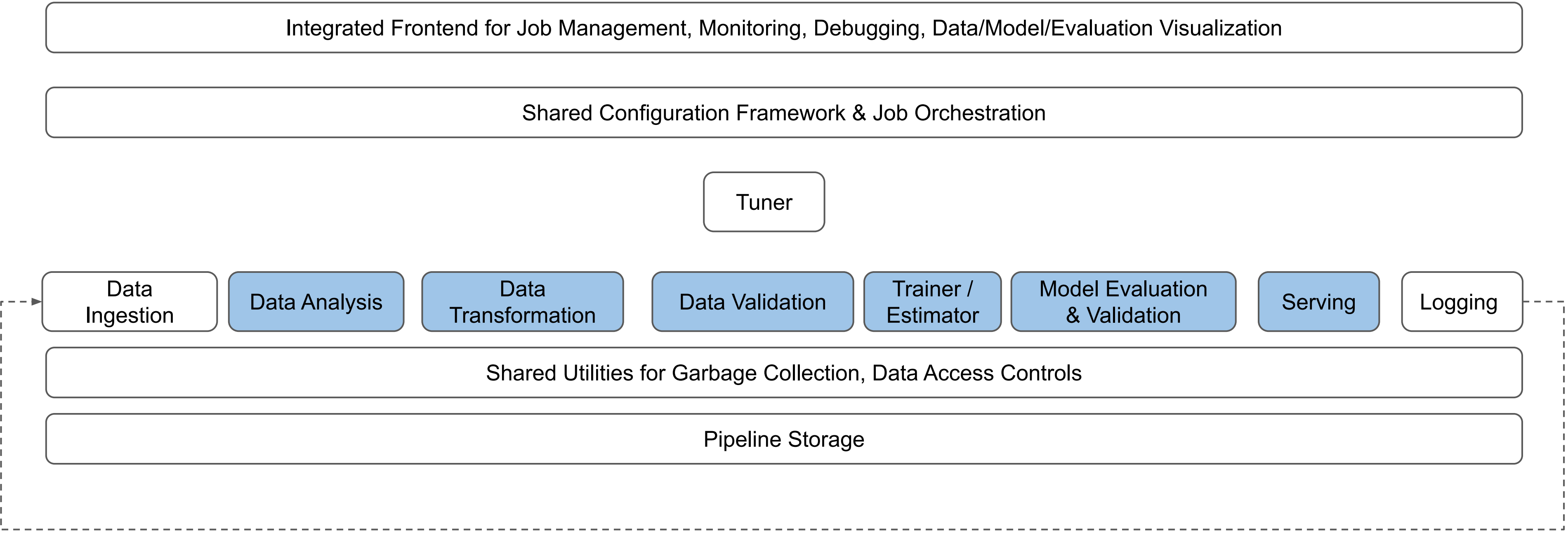


Takeaway: Doing machine learning in real world is HARD!

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

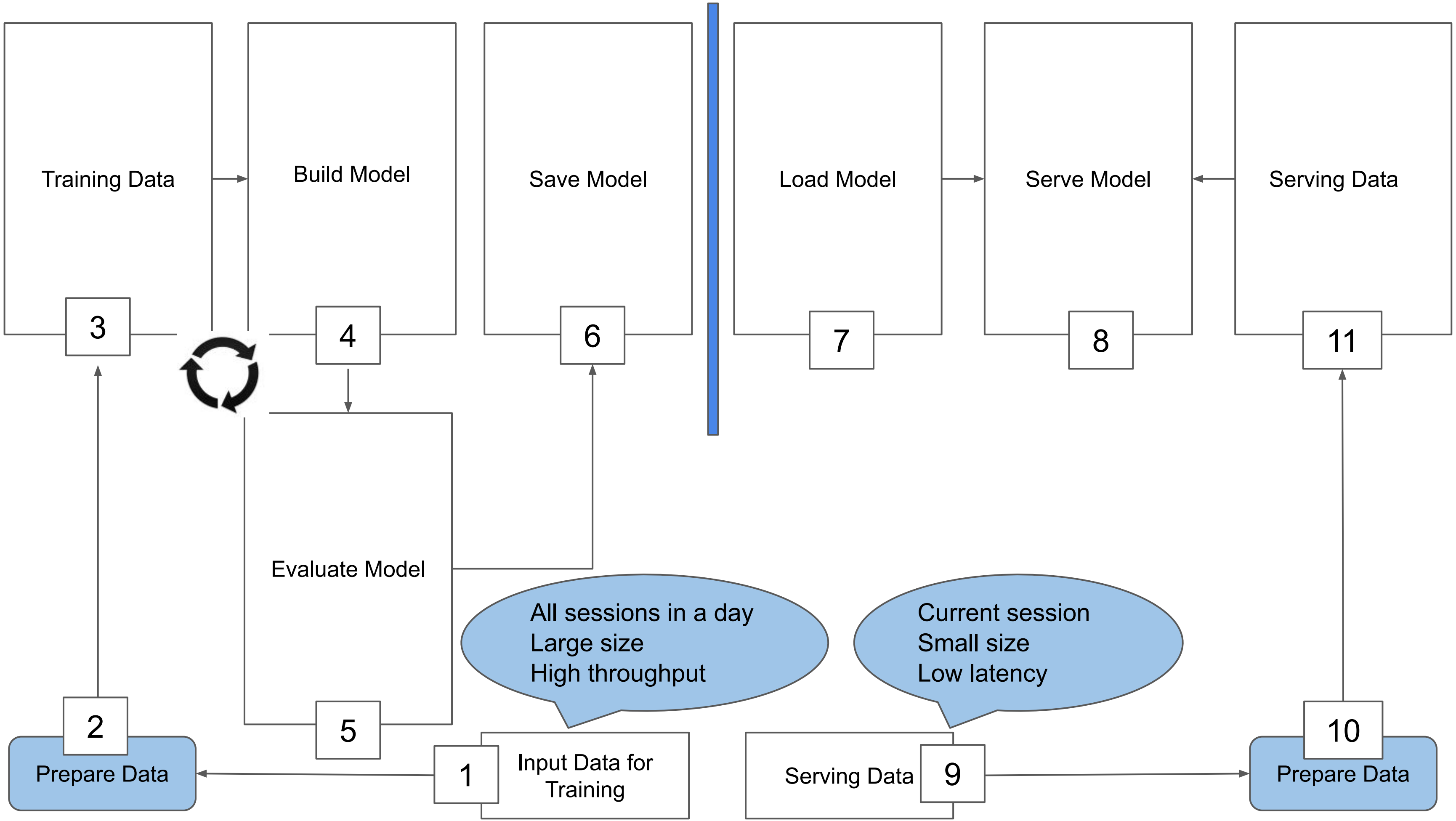
Open Source

Not Public Yet

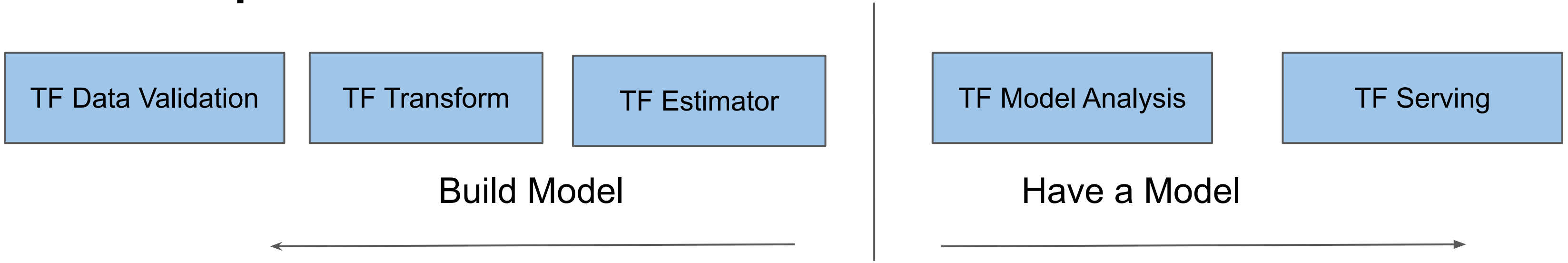
Link to TFX paper

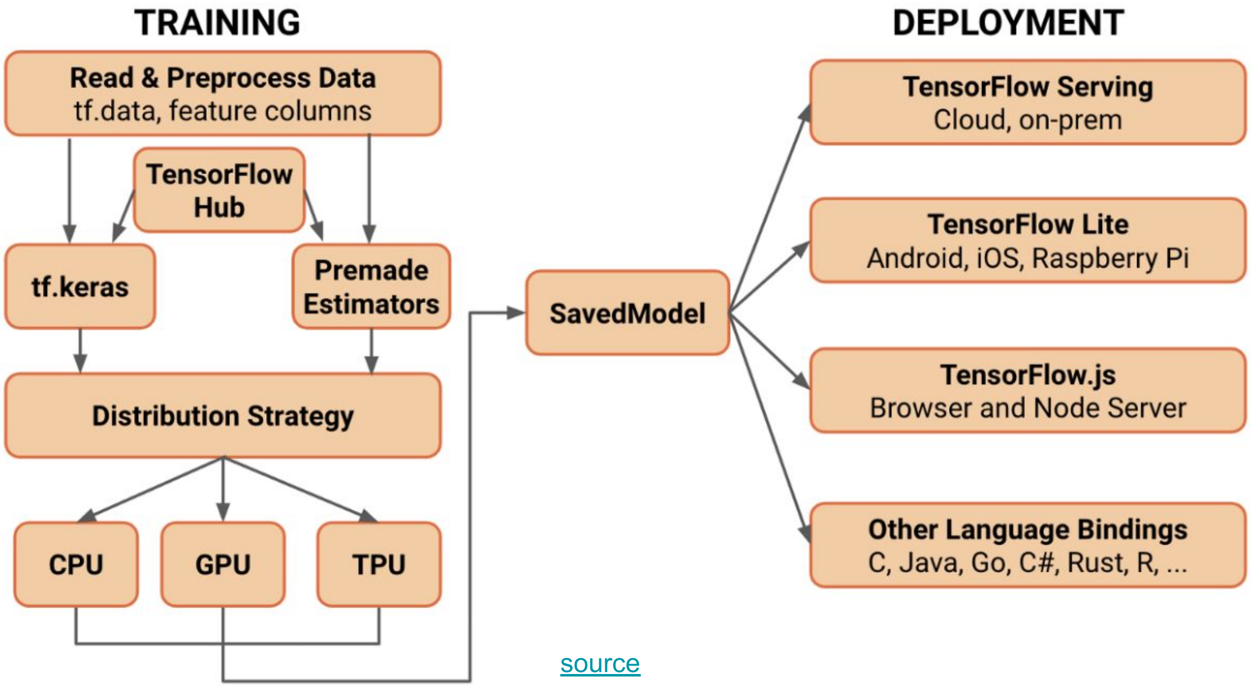
Train / Serving Data Flows

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



TFX Pipeline

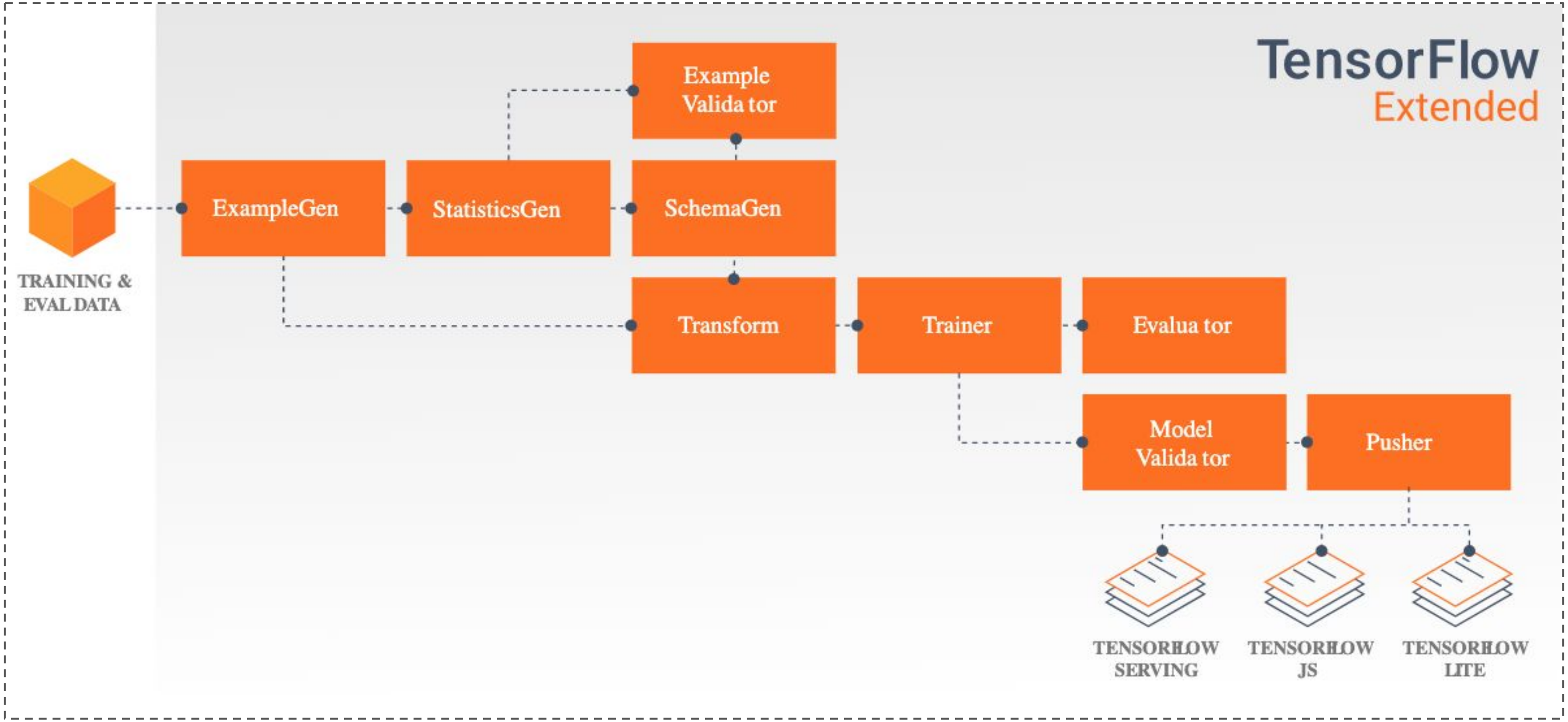




source

Architecture Overview

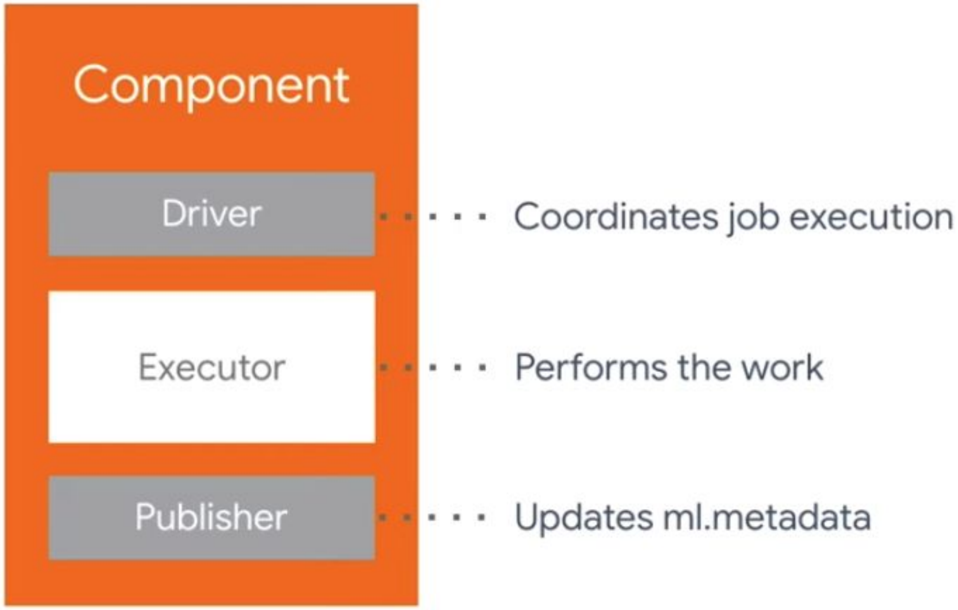
TFX pipelines can be orchestrated using Apache Airflow and Kubeflow Pipelines.
For this workshop, we will be running in interactive mode.



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

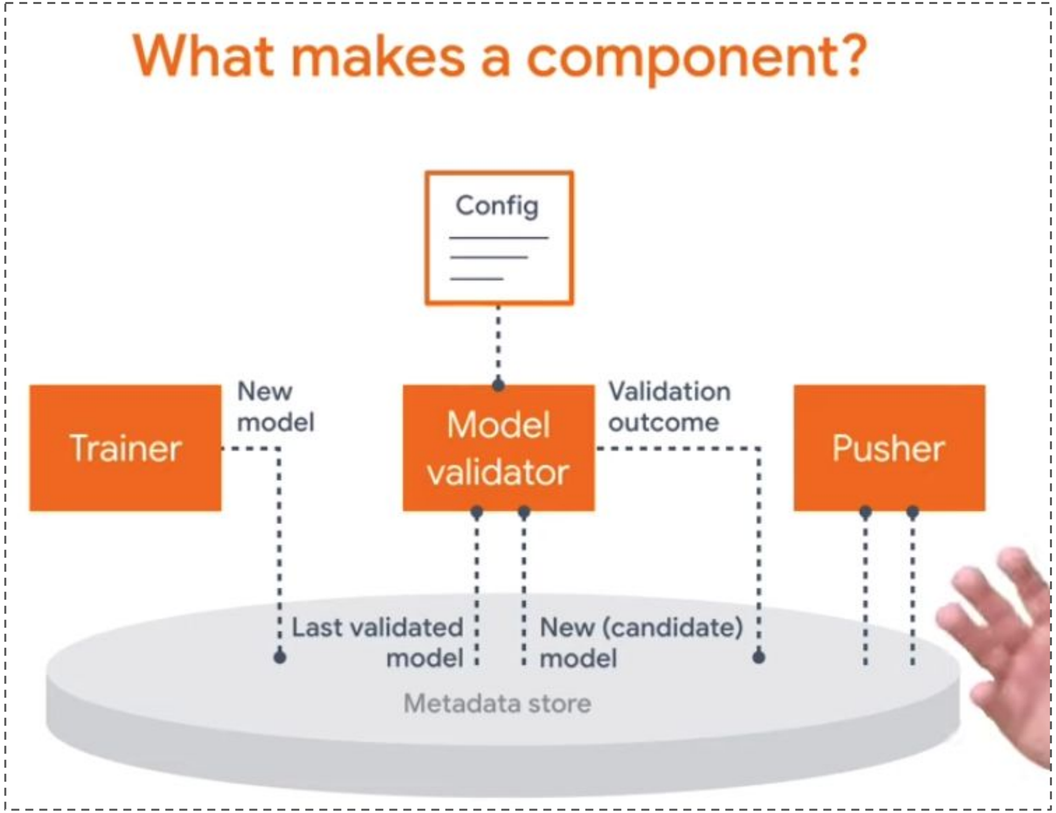


source

What is a TFX Component?

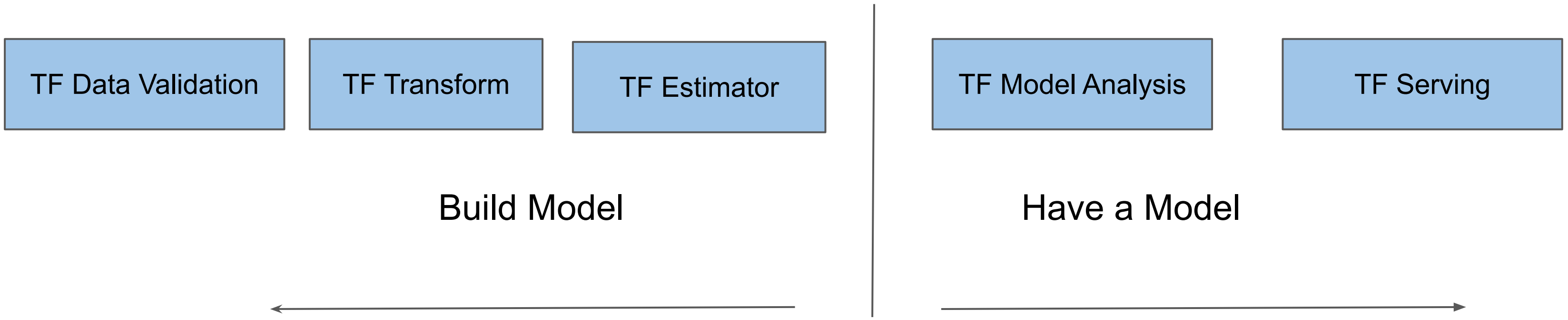
As data flows through pipelines…

- 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.

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
Model Build

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

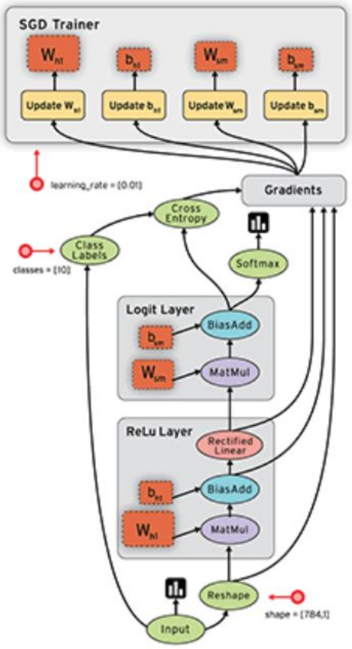
Target to predict: fare_amount

New York Yellow Cab dataset available via BigQuery public datasets

What is TF & Why Should You Care?

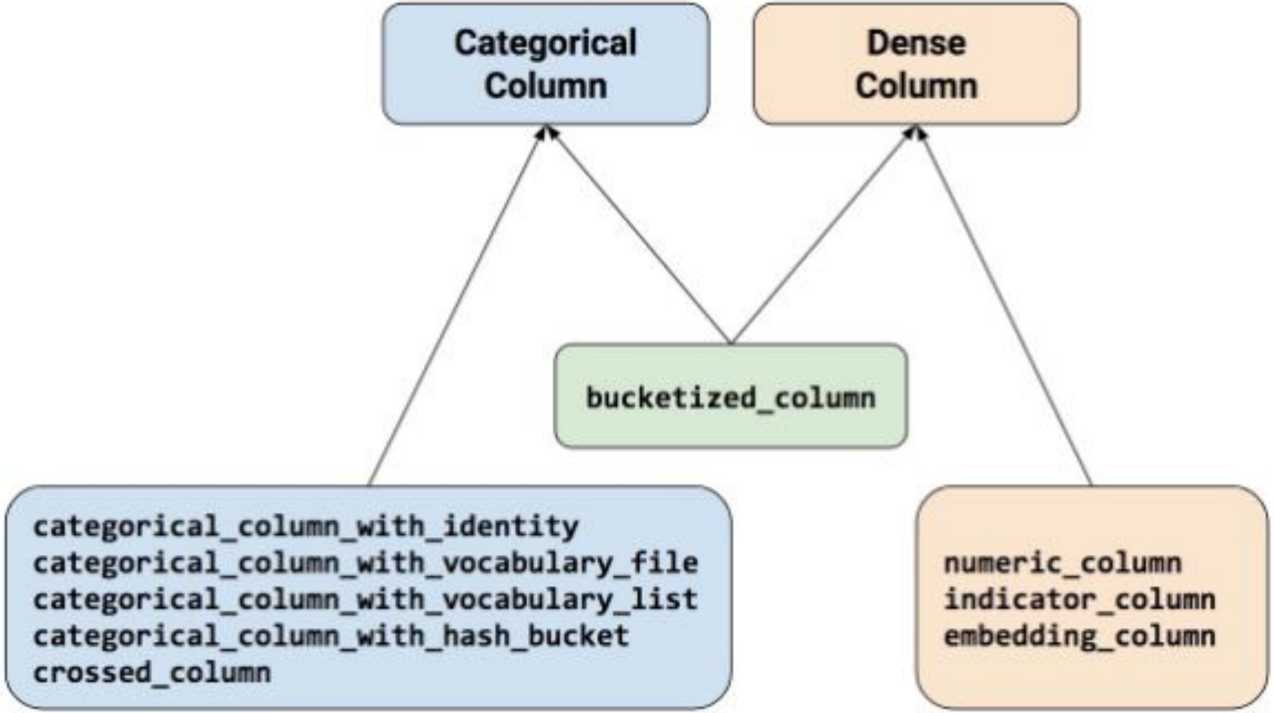
TensorFlow is an open source high performance library which uses directed graphs (see overview)

- Dataflow Model (link to paper)
  - Nodes represent math operations, Edges represent arrays of data
  - tf.math.add represented as..
    - single node w/ 2 input edges (matrices to be added)
    - 1 output edge (result of addition)
- **Flexible**
  - Works w/ image, audio, text and numerical data
- **Parallelism**
  - dataflow graph represents dependencies between operations, figure out which ops can execute in parallel)
- **Distributed Execution**
  - TF partitions your program across multiple devices (CPUs, GPUs, TPUs attached to different machines)
  - TF takes care of networking between machines
- **Compilation**
  - Benefit from compiler optimizations for dataflow graph using XLA
- **Portability**
  - Train model in Python, export SavedModel, serve in C++)



source

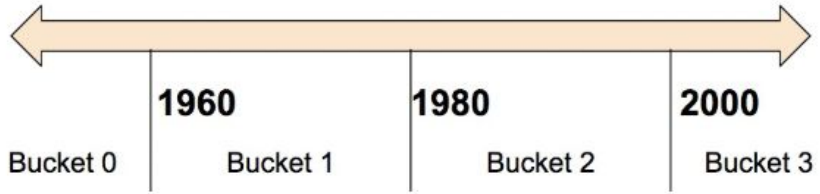
Tensorflow FeatureColumns Part 1



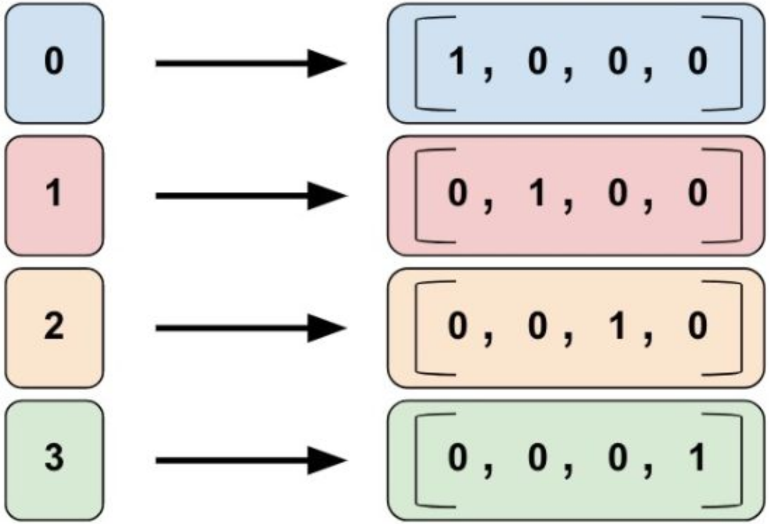
Feature column methods fall into two main categories and one hybrid category.

source

Tensorflow FeatureColumns Part 2



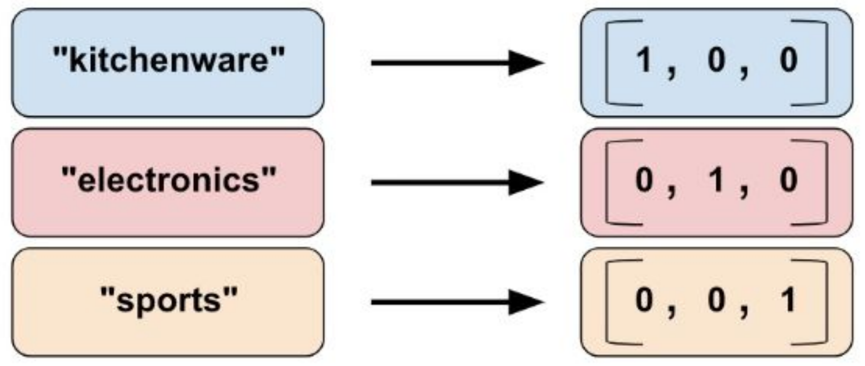
Dividing year data into four buckets.



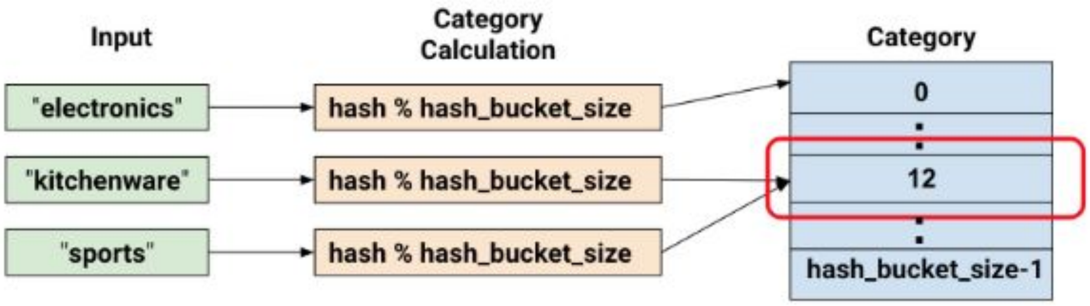
A categorical identity column mapping. Note that this is a one-hot encoding, not a binary numerical encoding.

[source]()

Tensorflow FeatureColumns Part 3



Mapping string values to vocabulary columns.



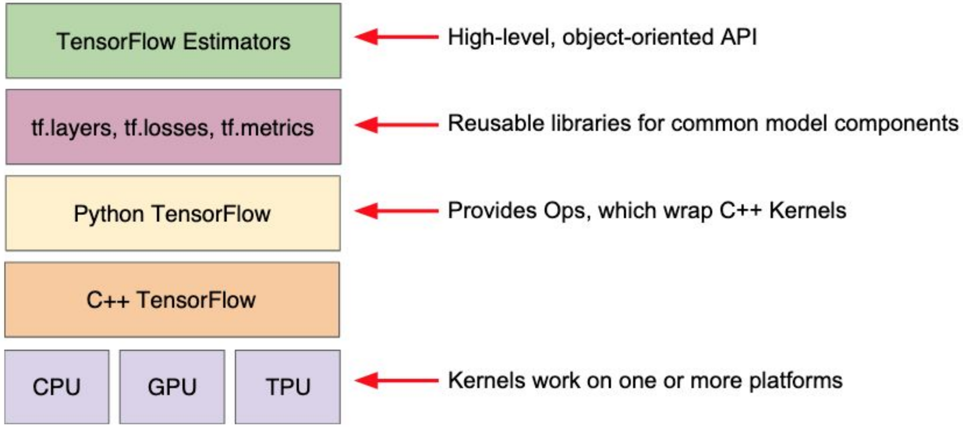
Representing data with hash buckets.

source

What is TF Estimator & Why Should You Care?

TF Estimator is a high level OOP API which makes it easier to train models (see overview)

- TF Estimator is compatible with the scikit-learn API
- Train models using CPU / GPU / TPUs
- Quicker model (graph) development
- Load large amounts of data
- Model checkpointing & recover from failures
- Train / Evaluation / Monitor
- Distributed Training
- Save summaries for TensorBoard
- Hyper-parameter tuning using ML Engine
- Serving predictions from a trained model
- Easily create Estimators from Keras models
- How to create custom estimators
- Need to implement ***preprocessing_fn*** & ***_build_estimator*** methods!



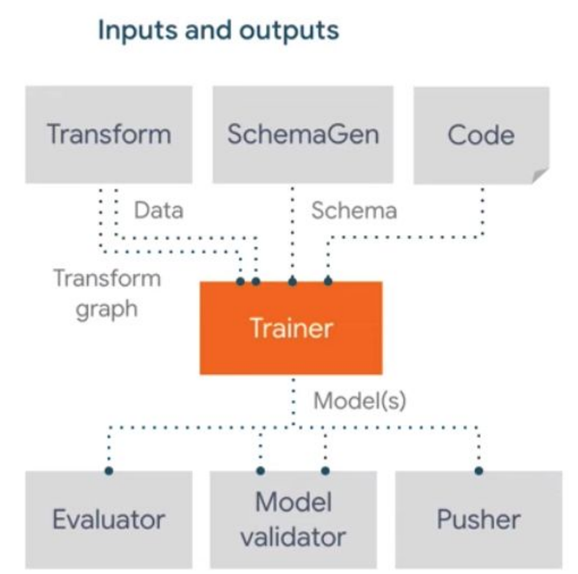
source

Trainer TFX Pipeline Component

Notes:

- To ingest data into your ML pipeline
  - Input: Transform graph, schema from SchemaGen, Code (model training code)
  - Output: Two different SavedModel (one for production inference, other for evaluation)

```
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)
```



source

preprocessing_fn(...)

```python
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 = {}
  print("inputs: ", inputs)
  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)

  fare_amount = _fill_in_missing(inputs[_LABEL_KEY])

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

source

_build_estimator(...)

```python
def _build_estimator(config, hidden_units=None, warm_start_from=None):
  """Build an estimator for predicting the tipping behavior of taxi riders.

  Args:
    config: tf.estimator.RunConfig defining the runtime environment for the
      estimator (including model_dir).
    hidden_units: [int], the layer sizes of the DNN (input layer first)
    warm_start_from: Optional directory to warm start from.

  Returns:
    A dict of the following:
      - estimator: The estimator that will be used for training and eval.
      - train_spec: Spec for training.
      - eval_spec: Spec for eval.
      - eval_input_receiver_fn: Input function for eval.
  """
  real_valued_columns = [
      tf.feature_column.numeric_column(key, shape=())
      for key in _transformed_names(_DENSE_FLOAT_FEATURE_KEYS)
  ]
  categorical_columns = [
      tf.feature_column.categorical_column_with_identity(
          key, num_buckets=_VOCAB_SIZE + _OOV_SIZE, default_value=0)
      for key in _transformed_names(_VOCAB_FEATURE_KEYS)
  ]
  categorical_columns += [
      tf.feature_column.categorical_column_with_identity(
          key, num_buckets=_FEATURE_BUCKET_COUNT, default_value=0)
      for key in _transformed_names(_BUCKET_FEATURE_KEYS)
  ]

  return tf.estimator.DNNLinearCombinedRegressor(
      config=config,
      dnn_feature_columns=real_valued_columns,
      dnn_hidden_units=hidden_units or [100, 70, 50, 25],
      warm_start_from=warm_start_from)
```

source

TF Model Build Knowledge Check

Q: Which of the following is a supported feature column method in TF Estimator?

a) tf.feature_column.numeric_column()
b) tf.feature_column.categorical_column_with_vocabulary_list()
c) tf.feature_column.categorical_column_with_identity()
d) tf.feature_column.indicator_column()
e) All of the above

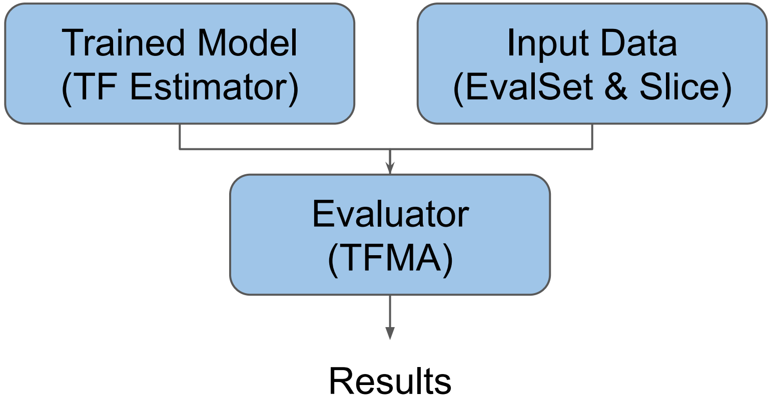
TensorFlow Model Analysis

What is TF Model Analysis & Why Should You Care?

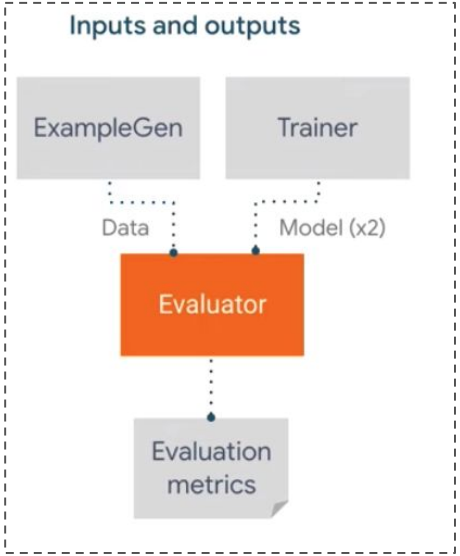
TF Model Analysis is a library for evaluating TF models.

Benefits include…

- Allows you to evaluate models on large amounts of data
- You can choose which metric & what slice/segment of your data to evaluate model predictions on
    - This helps you find slices of data for a given feature where the model performs poorly
    - Great model debugging tool
- Track performance over time
    - Trends of different models over time
    - As you get new data
- User friendly visualization tool



Evaluator TFX Pipeline Component



Notes:

- To evaluate overall and individual data slices
  - Input: ExampleGen, Trainer
  - Output: Evaluation Metrics
  - Helps identify individual points where model performs poorly

```
model_analyzer = Evaluator(
    examples=example_gen.outputs['examples'],
    model_exports=trainer.outputs['output'],
    feature_slicing_spec=evaluator_pb2.FeatureSlicingSpec(
        specs=ALL_SPECS
    ))
context.run(model_analyzer)
```

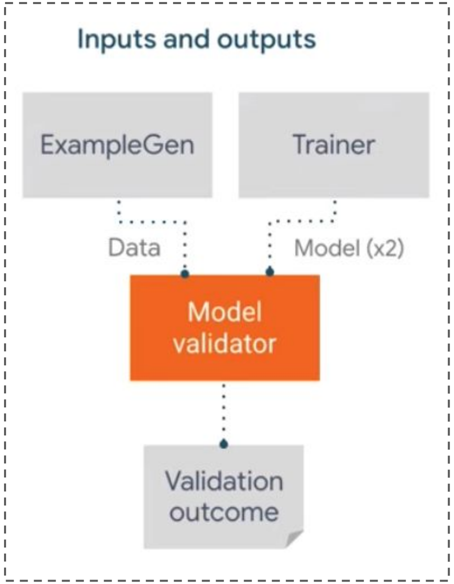
source

ModelValidator TFX Pipeline Component

Notes:

- Is the new model better or worse than what we have in production?
    - Input: ExampleGen, Trainer
    - Output: Validation Outcome

```
model_validator = ModelValidator(
    examples=example_gen.outputs['examples'],
    model=trainer.outputs['output'])
context.run(model_validator)
```



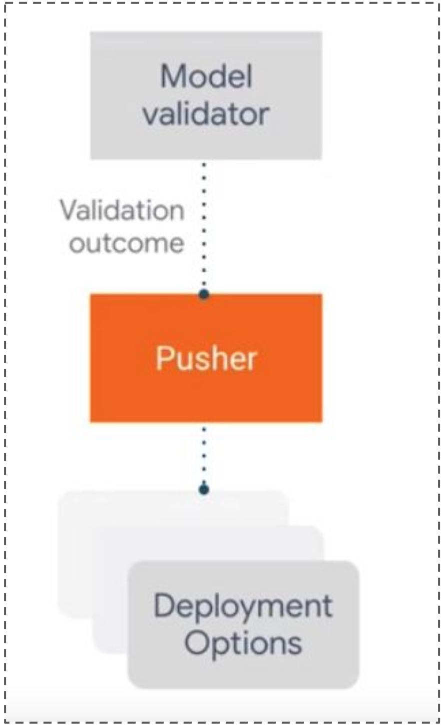
source

Pusher TFX Pipeline Component

Notes:

- If model validation passes, push to production
  - Input: Model Validator
  - Output: Deployment options
    - TF Lite
    - TF JS
    - TF Serving

```
pusher = Pusher(
    model_export=trainer.outputs['output'],
    model_blessing=model_validator.outputs['blessing'],
    push_destination=pusher_pb2.PushDestination(
        filesystem=pusher_pb2.PushDestination.Filesystem(
            base_directory=_serving_model_dir)))
context.run(pusher)
```



source

TensorFlow Model Analysis API

Feature Engineering @ Scale	**Transformations**
Evaluate & persist results.	tfma.ExtractEvaluateAndWriteResults()
Creates an EvalResult object for use with the visualization functions.	tfma.load_eval_result()
Run model analysis for a single model on multiple data sets.	tfma.multiple_data_analysis()
Run model analysis for multiple models on the same data set.	tfma.multiple_model_analysis()
Runs TensorFlow model analysis.	tfma.run_model_analysis()

Define Feature Slices for TFMA

```
# an empty slice spec means the whole dataset
OVERALL_SLICE_SPEC = evaluator_pb2.SingleSlicingSpec()

# data can be sliced along a feature column.
FEATURE_COLUMN_SLICE_SPEC = evaluator_pb2.SingleSlicingSpec(
    column_for_slicing=['pickup_hour'])

# slices are computed for pickup_day_of_week x pickup_month.
FEATURE_COLUMN_CROSS_SPEC = evaluator_pb2.SingleSlicingSpec(
    column_for_slicing=['trip_distance', 'tip_amount'])

ALL_SPECS = [
    OVERALL_SLICE_SPEC,
    FEATURE_COLUMN_SLICE_SPEC,
    FEATURE_COLUMN_CROSS_SPEC,
]
```

Number of rows per hour of day!

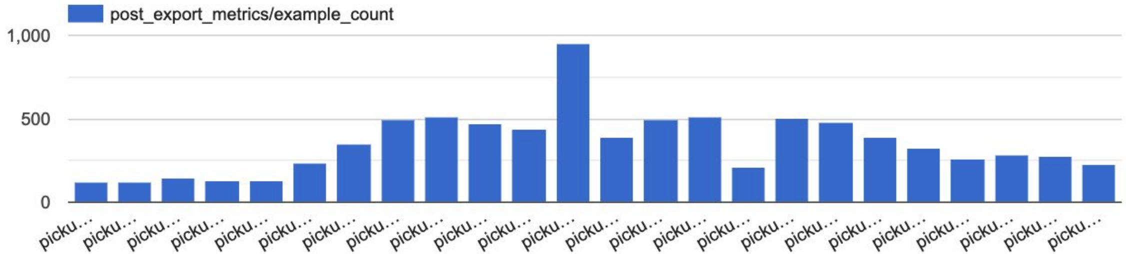
```
[29]: tfma.view.render_slicing_metrics(tfma_result_1, slicing_column='pickup_hour')
```

Visualization: Slices Overview

Examples (Weighted) Threshold: 0

Show: post_export_metrics/example_count

Sort by: Slice



feature	average_loss	label/mean	post_export_metrics/example_count	prediction/mean
pickup_hour:10	525.09387	33.54107	470	32.56553
pickup_hour:11	176.18619	32.91829	443	32.55243
pickup_hour:8	88.13461	32.06847	498	31.39557
pickup_hour:9	86.65687	32.35592	515	31.66985
pickup_hour:14	93.70968	33.71512	496	31.97082
pickup_hour:15	101.93111	34.50546	513	32.26447
pickup_hour:12	79.43258	35.09818	952	33.88065
pickup_hour:13	80.08283	33.61849	392	33.47749

```
tfma.view.render_slicing_metrics(tfma_result_1, slicing_column='pickup_hour')
```

Visualization
Slices Overview

Examples (Weighted) Threshold
0

Note distribution of mean prediction

Show
prediction/mean

Sort by
Slice

feature	average_loss	label/mean	post_export_metrics/example_count	prediction/mean
pickup_hour:10	525.09387	33.54107	470	32.56553
pickup_hour:11	176.18619	32.91829	443	32.55243
pickup_hour:8	88.13461	32.06847	498	31.39557
pickup_hour:9	86.65687	32.35592	515	31.66985
pickup_hour:14	93.70968	33.71512	496	31.97082
pickup_hour:15	101.93111	34.50546	513	32.26447
pickup_hour:12	79.43258	35.09818	952	33.88065
pickup_hour:13	80.08283	33.61849	392	33.47749

Note average loss anomaly!

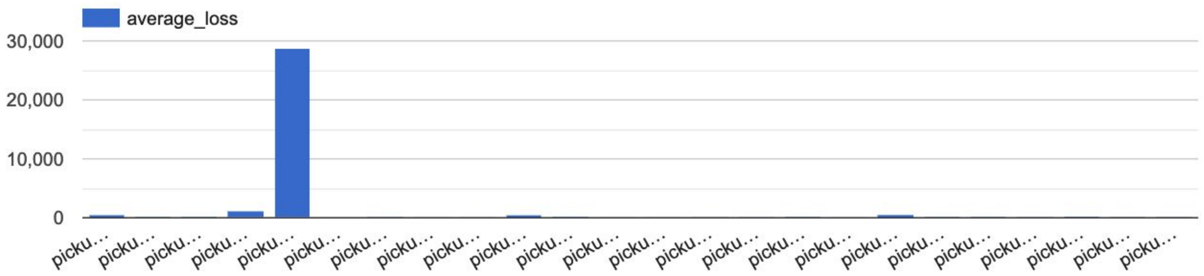
```
tfma.view.render_slicing_metrics(tfma_result_1, slicing_column='pickup_hour')
```

Visualization
Slices Overview

Examples (Weighted) Threshold
0

Show
average_loss

Sort by
Slice



feature	average_loss	label/mean	post_export_metrics/example_count	prediction/mean
pickup_hour:10	525.09387	33.54107	470	32.56553
pickup_hour:11	176.18619	32.91829	443	32.55243
pickup_hour:8	88.13461	32.06847	498	31.39557
pickup_hour:9	86.65687	32.35592	515	31.66985
pickup_hour:14	93.70968	33.71512	496	31.97082
pickup_hour:15	101.93111	34.50546	513	32.26447
pickup_hour:12	79.43258	35.09818	952	33.88065
pickup_hour:13	80.08283	33.61849	392	33.47749

Feature Cross Analysis

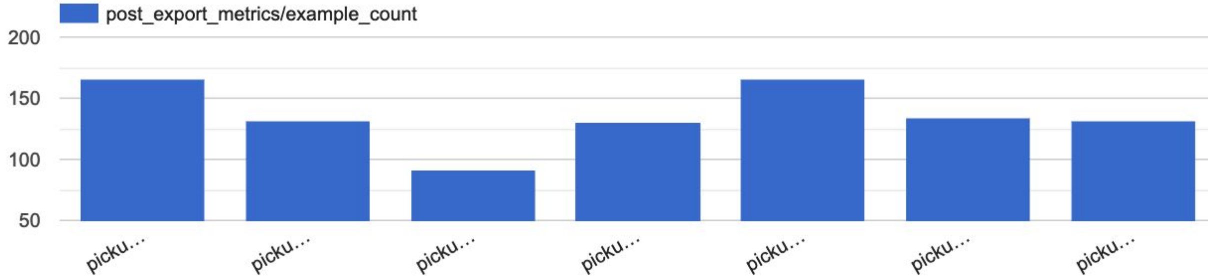
```
tfma.view.render_slicing_metrics(tfma_result_1, slicing_spec=COLUMN_CROSS_VALUE_SPEC)
```

Visualization
Slices Overview

Examples (Weighted) Threshold
0

Show
post_export_metrics/example_count

Sort by
Slice



feature	average_loss	label/mean	post_export_metrics/example_coun
pickup_day_of_week_X_pickup_hour:3_X_12	102.29386	36.47470	16
pickup_day_of_week_X_pickup_hour:2_X_12	37.41487	34.57879	13
pickup_day_of_week_X_pickup_hour:1_X_12	92.20728	39.41304	9
pickup_day_of_week_X_pickup_hour:5_X_12	93.57827	32.34354	13
pickup_day_of_week_X_pickup_hour:6_X_12	83.22913	36.78675	16
pickup_day_of_week_X_pickup_hour:4_X_12	53.81750	35.88806	13
pickup_day_of_week_X_pickup_hour:7_X_12	91.09421	30.66667	13

TF Model Analysis Knowledge Check

Q: TFMA is only useful if you’re building a model using TensorFlow?

a) True
b) False

TensorFlow Serving

What is TF Serving & Why Should You Care?

Requirements of a Model Serving System...

1. Low latency
   a. Isolation of load & serve threads
2. Efficient
   a. Dynamic request batching
3. Scale Horizontally
4. Reliable & Robust
5. Support loading/hosting multiple model versions dynamically
   a. Serve one model, while sending canary requests to new model
   b. Built in A/B testing
6. Deployment roll forward / backward
7. Serves over 1,500 models @Google, 100 predictions/sec

Dockerfile(s) maintained by Google

- Dockerfile, VM w/ TensorFlow Serving
- Dockerfile.gpu, VM w/ TensorFlow Serving (GPU support to be used with nvidia-docker)
- Dockerfile.devel, VM w/ all dependencies needed to build TensorFlow Serving
- Dockerfile.devel-gpu, VM w/ all dependencies needed to build TensorFlow Serving w/ GPU support.

Test Drive it Yourself...

```bash
#!/bin/bash

# Download the TensorFlow Serving Docker image and repo
docker pull tensorflow/serving

# for GPU, use...
docker pull tensorflow/serving:latest-gpu

git clone https://github.com/tensorflow/serving
# Location of demo models
TESTDATA="$(pwd)/serving/tensorflow_serving/servables/tensorflow/testdata"

# Start TensorFlow Serving container and open the REST API port
docker run -t --rm -p 8501:8501 \
    -v "$TESTDATA/saved_model_half_plus_two_cpu:/models/half_plus_two" \
    -e MODEL_NAME=half_plus_two \
    tensorflow/serving &

# For GPU, use...
# docker run --runtime=nvidia -p 8501:8501 \
#   --mount type=bind,\
#   source=/tmp/tfserving/serving/tensorflow_serving/servables/tensorflow/testdata/saved_model_half_plus_two_gpu,\
#   target=/models/half_plus_two \
#   -e MODEL_NAME=half_plus_two -t tensorflow/serving:latest-gpu &

# Query the model using the predict API
curl -d '{"instances": [1.0, 2.0, 5.0]}' \
    -X POST http://localhost:8501/v1/models/half_plus_two:predict

# Returns => { "predictions": [2.5, 3.0, 4.5] }
```

TF Serving Out of the Box (w/ Docker)

```
# set name of Tensorflow Serving docker image & download it
# https://github.com/tensorflow/serving/tree/master/tensorflow_serving/tools/docker
DOCKER_IMAGE_NAME=tensorflow/serving
echo Download TF Serving docker image: $DOCKER_IMAGE_NAME
docker pull $DOCKER_IMAGE_NAME

# location of local model to be used by Tensorflow Serving
# should contain a folder named with a UTC timestamp
MODEL_BASE_PATH=$(pwd)/tf/run_0/serving_model_dir/export/nyc-taxi

# location of model dir within container
CONTAINER_MODEL_BASE_PATH=/models/nyc-taxi

# local port where to send inference requests
HOST_PORT=9000

# container listening port for inference requests
CONTAINER_PORT=8500

echo Model directory: $MODEL_BASE_PATH

docker run -it \
  -p 127.0.0.1:$HOST_PORT:$CONTAINER_PORT \
  -v $MODEL_BASE_PATH:$CONTAINER_MODEL_BASE_PATH \
  -e MODEL_NAME=nyc-taxi\
  --rm $DOCKER_IMAGE_NAME
```

SavedModel Artifacts

After training, we have a trained saved model (universal format)

- Learned variable weights
- Graph
- Embeddings & Vocabs
- Inferred Schema
- Transformed features

```
notebooks/tft_train/run_0
├── tftransform_tmp
│   ├── 50a3468d584b42839cfc3c72e5a56e5f
│   │   ├── saved_model.pb
│   │   └── variables
│   ├── 582667bf1cf54f3a9fee504dbaef4ffd
│   │   ├── saved_model.pb
│   │   └── variables
│   └── 956a82774bf1480892cce94dba33eddd
│       ├── saved_model.pb
│       └── variables
├── train_transformed-00000-of-00001.gz
├── transform_fn
│   ├── saved_model.pb
│   └── variables
└── transformed_metadata
    └── v1-json
        └── schema.json
```

TF Serving (w/ Docker)

```python
def do_inference(model_handle, examples_file, num_examples, schema):
    filtered_features = [
      feature for feature in schema.feature if feature.name != LABEL_KEY
    ]
    del schema.feature[:]
    schema.feature.extend(filtered_features)

    csv_coder = make_csv_coder(schema)
    proto_coder = make_proto_coder(schema)

    input_file = open(examples_file, 'r')
    # skip header line
    input_file.readline()

    serialized_examples = []
    for _ in range(num_examples):
        one_line = input_file.readline()
        if not one_line:
            print('End of example file reached')
            break
        one_example = csv_coder.decode(one_line)

        serialized_example = proto_coder.encode(one_example)
        serialized_examples.append(serialized_example)

    parsed_model_handle = model_handle.split(':')
    do_local_inference(
      host=parsed_model_handle[0],
      port=parsed_model_handle[1],
      serialized_examples=serialized_examples)
```

TF Serving Inference

```
SERVER="127.0.0.1:9000"
SCHEMA_FILE="./schema.pbtxt"
NUM_INSTANCES=15
INSTANCE_FILE="../data/train/train.csv"

do_inference(SERVER,
             INSTANCE_FILE,
             NUM_INSTANCES,
             read_schema(SCHEMA_FILE))
```

```
outputs {
  key: "predictions"
  value {
    dtype: DT_FLOAT
    tensor_shape {
      dim {
        size: 15
      }
      dim {
        size: 1
      }
    }
    float_val: 32.9800186157
    float_val: 17.2102165222
    float_val: 31.7519054413
    float_val: 33.0067481995
    float_val: 36.803691864
    float_val: 33.8259849548
    float_val: 34.7675018311
    float_val: 22.1285438538
    float_val: 43.9800224304
    float_val: 22.4845104218
    float_val: 37.6693458557
    float_val: 34.8329048157
    float_val: 31.9676322937
    float_val: 70.7163391113
    float_val: 37.2004470825
  }
}
model_spec {
  name: "nyc-taxi"
  version {
    value: 1551254584
  }
  signature_name: "predict"
}
```

TF Serving ModelServer REST API

```
First, you'll need to install TF Model Server...
apt-get remove tensorflow-model-server

POST http://host:port/<URI>:<VERB>
URI - /v1/models/${MODEL_NAME}[/versions/${MODEL_VERSION}]
VERBS - classify | regress | predict

Classify Format:
POST http://host:port/v1/models/${MODEL_NAME}[/versions/${MODEL_VERSION}]:classify

Classify Example:
POST http://host:port/v1/models/iris/versions/1:classify


Predict Format:
POST http://host:port/v1/models/${MODEL_NAME}[/versions/${MODEL_VERSION}]:predict

Predict Example:
POST http://host:port/v1/models/mnist/versions/1:predict
```

End-2-End example

Next Steps:

- Work through TFMA & TFServing Notebooks