Evaluating MLOps Tools

Watch this talk on YouTube

Hamel Husain

Stanford CS329S ML Systems Design, February 2022

About Hamel

Building Tools

DataRobot

fast.ai

“Cool” Applied ML

OSS Contributions

fast.ai

jupyter

Kubeflow

great_expectations

“Uncool” Applied ML

LI & FUNG

AT&T

Sprint

QUIKS

crocs

Ruby Tuesday

CHASE

MGM

Motivation

It All Started With Some Provocative Comments

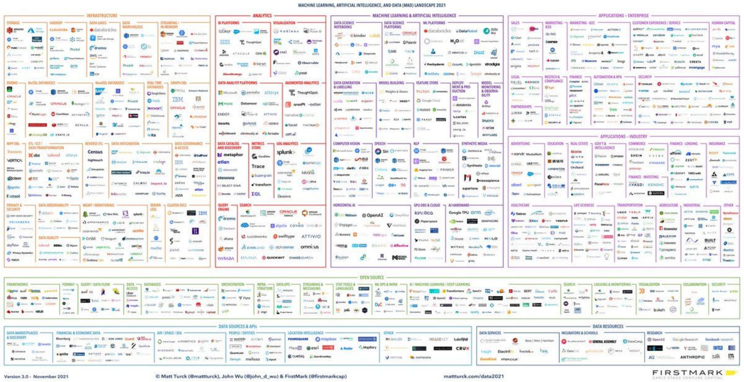
ML Educator, Author,
Frequently Gives Talks

"... People don't use PyTorch for production, no **real** applied ML occurs in pytorch "

"data drift and model drift are a completely solved problem in TFX"

Me

This Is Incredibly Common In Tech, But More Acute In MLOPS



- Zealouts **often appear when there is an incredible amount of entropy** in a domain
- Zealouts often come prepared with an arsenal of cherry picked features
- Appeal to authority: This worked at {Google, Facebook, etc}

Criterion For Evaluating Tools

- Friction in critical parts of the workflow
- Rapid prototyping / iteration
- As few DSLs as possible
- Ergonomic
- Interopability w/other tools
- Quality of documentation
- Progressive complexity: easy to get started

TensorFlow Extended

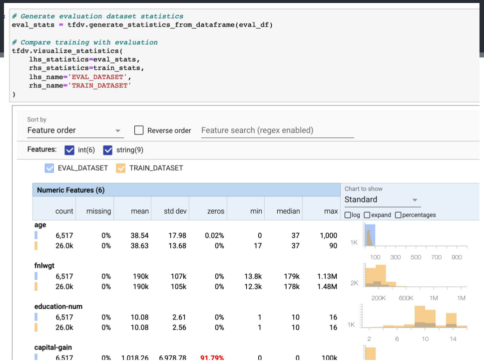
Things I Like About TFX

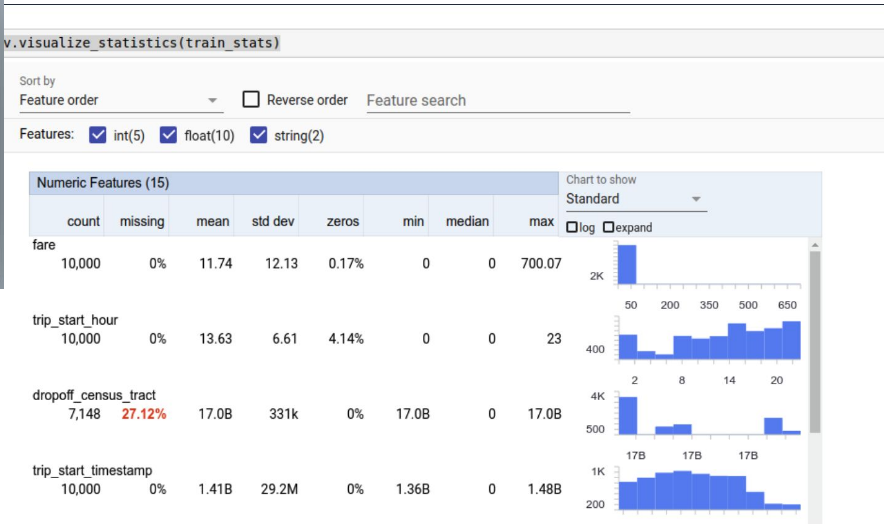
1. TF Data Validation
2. Tensorboard
3. TF Serving

Things I like about TFX: TFDV For Auto-EDA

The way we currently do EDA is broken. We often end up creating the same set of viz for each dataset. Two lines of code and we can get a headstart. Works with existing tools. Really light weight.

```
# Generate training dataset statistics
train_stats = tfdv.generate_statistics_from_dataframe(train_df)

# Visualize training dataset statistics
tfdv.visualize_statistics(train_stats)
```





Things I like about TFX: TFDV For Data Validation

TFDV offers reasonable data validation that allows you to quickly compare two datasets and detect (1) anomalies and (2) schema changes.

```
# Check evaluation data for errors by validating the evaluation dataset statisti
anomalies =  tfdv.validate_statistics(statistics=eval_stats, schema=schema)

# Visualize anomalies
tfdv.display_anomalies(anomalies)
```

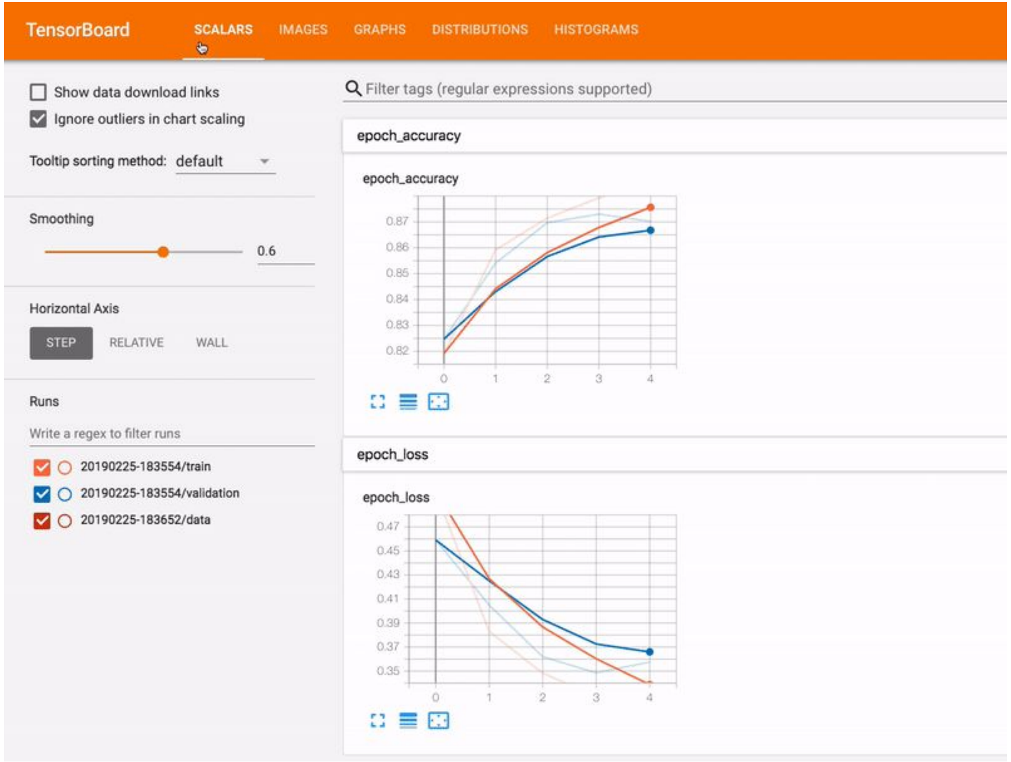
Feature name	Anomaly short description	Anomaly long description
'race'	Unexpected string values	Examples contain values missing from the schema: Asian (<1%).
'native-country'	Unexpected string values	Examples contain values missing from the schema: Mongolia (<1%).
'occupation'	Unexpected string values	Examples contain values missing from the schema: gamer (<1%).

Works with existing tools out of the box.

Very lightweight.

Things I like about TFX: Tensorboard

Loved by ML people everywhere. Framework agnostic. Easy to use. Just works.



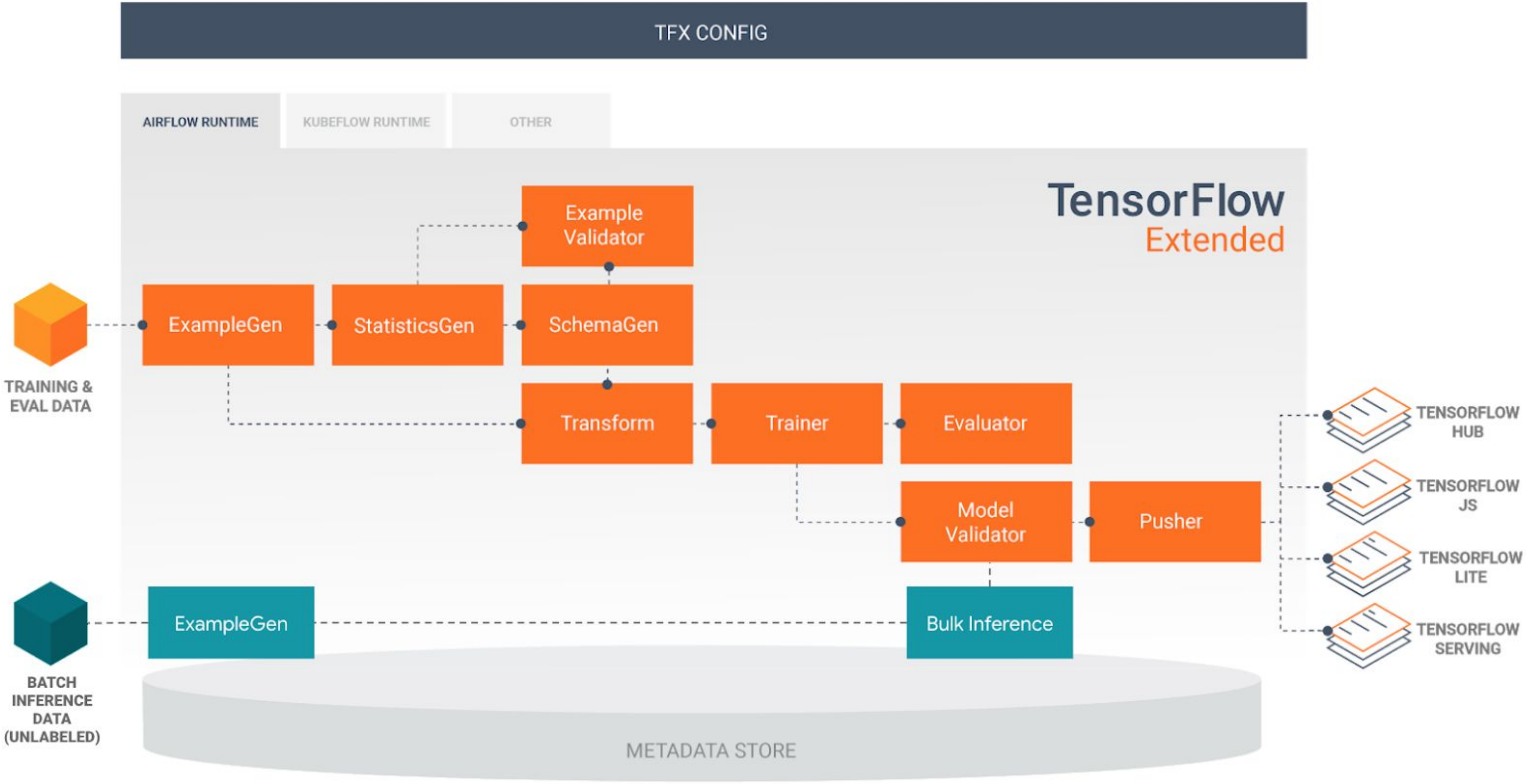
Things I like about TFX: TF Serving

TF Specific, but easy to use. Lots of good features. Easy to get started.

```
tensorflow_model_server \
  --rest_api_port=8505 \
  --model_name=my_model \
  --model_base_path="/content/${HAMEL_MODEL_DIR}" >server.log 2>&1
```

```
data = json.dumps({"instances": datalist})
headers = {"content-type": "application/json"}

json_response = requests.post('http://localhost:8505/v1/models/hamel_test:predict',
                              data=datalist,
                              headers=headers)
```

Things I don’t like about TFX: Everything Else



Tools Need To Promote Rapid Iteration On Data

Andrew Ng

“I recommend you hold the model or code fixed and iteratively improve the quality of the data.”

“I found that rather than taking a model centric view .. you can use an open source implementation of something you download of GitHub and instead just focus on optimizing the data.”

TFX Transform

Let's Discourage Feature Engineering

```
    # Use `tf.cast` to cast the label key to float32 and fill in the missing
values.
    traffic_volume = tf.cast(inputs[_VOLUME_KEY], tf.float32)


    # Create a feature that shows if the traffic volume is greater than the mean
and cast to an int
    outputs[_transformed_name(_VOLUME_KEY)] = tf.cast(

        # Use `tf.greater` to check if the traffic volume in a row is greater than
the mean of the entire traffic volumn column
        tf.greater(tf.cast(inputs[_VOLUME_KEY], tf.float32),
tft.mean(tf.cast(inputs[_VOLUME_KEY], tf.float32))),

        tf.int64)
```

You are limited to the narrow set of ops that TF provides.

Must learn a new DSL to perform simple math operations like >

You wrote plain python code or used pandas? Too bad. Refactor it.

Spacy? Forget about it

TFX Transform

Want To Iterate On Data? Slowwww Down
Scripts only please! No notebooks.

```
%%writefile {_traffic_transform_module_file}

import tensorflow as tf
import tensorflow_transform as tft

import traffic_constants

# Unpack the contents of the constants module
_DENSE_FLOAT_FEATURE_KEYS = traffic_constants.DENSE_FLOAT_FEATURE_KEYS
_RANGE_FEATURE_KEYS = traffic_constants.RANGE_FEATURE_KEYS
_VOCAB_FEATURE_KEYS = traffic_constants.VOCAB_FEATURE_KEYS
_VOCAB_SIZE = traffic_constants.VOCAB_SIZE
_OOV_SIZE = traffic_constants.OOV_SIZE
_CATEGORICAL_FEATURE_KEYS = traffic_constants.CATEGORICAL_FEATURE_KEYS
```

Difficult to perform feature transforms interactively.

Even official tutorials export a python script so transforms can be executed.

Iteration is slow. High cognitive load.

“Hello World” / Paved Paths Use Distributed Compute APIs

```python
import os
from typing import Optional, Text, List
from absl import logging
from ml_metadata.proto import metadata_store_pb2
import tfx.v1 as tfx

PIPELINE_NAME = 'my_pipeline'
PIPELINE_ROOT = os.path.join('.', 'my_pipeline_output')
METADATA_PATH = os.path.join('.', 'tfx_metadata', PIPELINE_NAME, 'metadata.db')
ENABLE_CACHE = True

def create_pipeline(
  pipeline_name: Text,
  pipeline_root:Text,
  enable_cache: bool,
  metadata_connection_config: Optional[
    metadata_store_pb2.ConnectionConfig] = None,
  beam_pipeline_args: Optional[List[Text]] = None
):
  components = []

  return tfx.dsl.Pipeline(
        pipeline_name=pipeline_name,
        pipeline_root=pipeline_root,
        components=components,
        enable_cache=enable_cache,
        metadata_connection_config=metadata_connection_config,
        beam_pipeline_args=beam_pipeline_args,
    )

def run_pipeline():
  my_pipeline = create_pipeline(
      pipeline_name=PIPELINE_NAME,
      pipeline_root=PIPELINE_ROOT,
      enable_cache=ENABLE_CACHE,
      metadata_connection_config=tfx.orchestration.metadata.sqlite_m
      )

  tfx.orchestration.LocalDagRunner().run(my_pipeline)

if __name__ == '__main__':
  logging.set_verbosity(logging.INFO)
  run_pipeline()
```

Distributed computing framework and new DSLs from the beginning vs using existing tools.

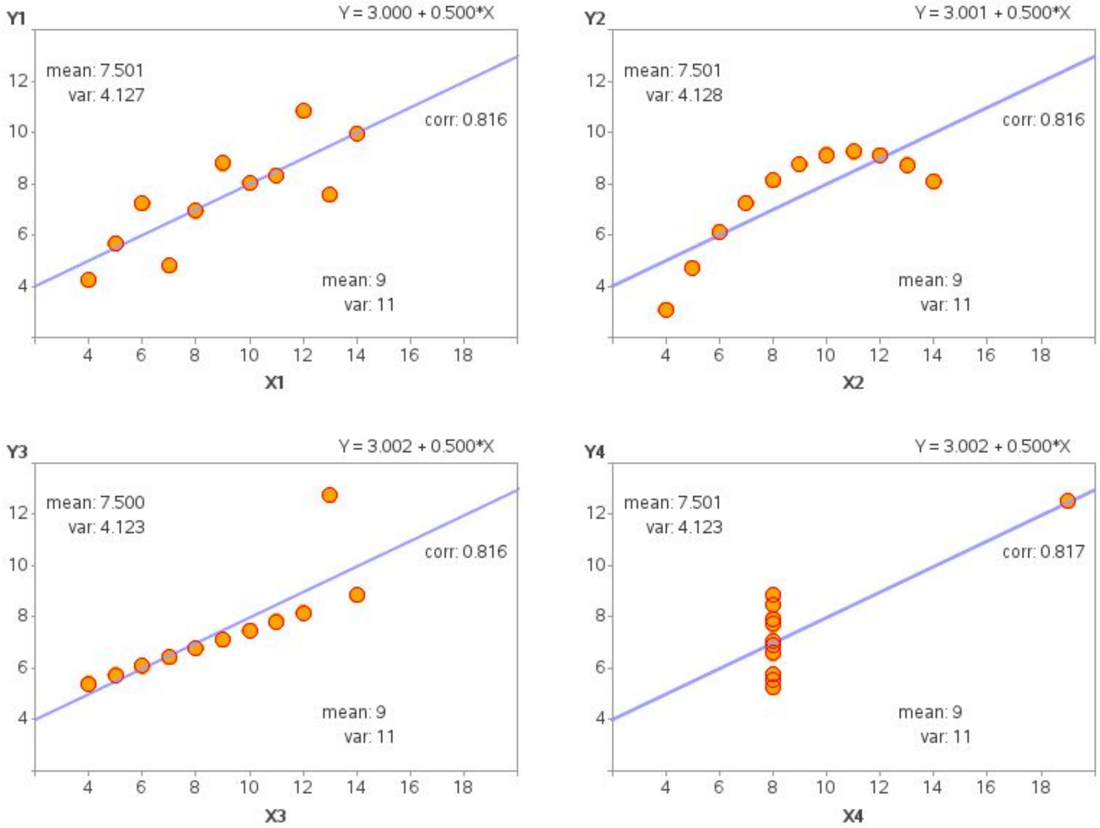
Jump straight into the deep end of complexity

Orchestrating TFX Pipelines

Apache Beam

Several TFX components rely on Beam for distributed data processing. In addition, TFX can use Apache Beam to orchestrate and execute the pipeline DAG. Beam orchestrator uses a different BeamRunner than the one which is used for component data processing. With the default DirectRunner setup the Beam orchestrator can be used for local

You Can’t Understand Your Data w/o Visualizations



Model Validation (TFMA)

Data visualizations with ~~code~~ **config files** using ~~existing~~ **a new DSL**

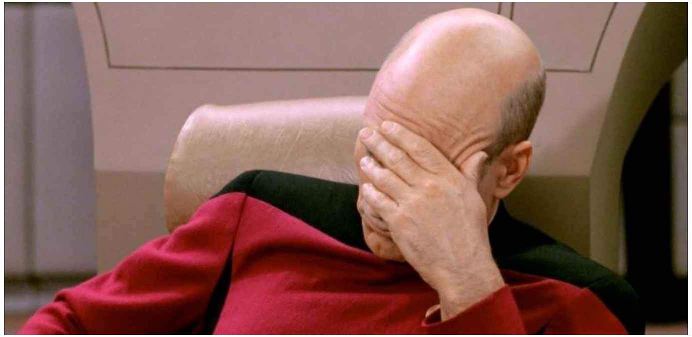
```
import tensorflow_model_analysis as tfma
from google.protobuf import text_format

# Setup tfma.EvalConfig settings
eval_config = text_format.Parse("""
  ## Model information
  model_specs {
    # For keras (and serving models), you need to add a `label_key`.
    label_key: "label"
  }

  ## Post training metric information. These will be merged with any built-in
  ## metrics from training.
  metrics_specs {
    metrics { class_name: "ExampleCount" }
    metrics { class_name: "BinaryAccuracy" }
    metrics { class_name: "BinaryCrossentropy" }
    metrics { class_name: "AUC" }
    metrics { class_name: "AUCPrecisionRecall" }
    metrics { class_name: "Precision" }
    metrics { class_name: "Recall" }
    metrics { class_name: "MeanLabel" }
    metrics { class_name: "MeanPrediction" }
    metrics { class_name: "Calibration" }
    metrics { class_name: "CalibrationPlot" }
    metrics { class_name: "ConfusionMatrixPlot" }
    # ... add additional metrics and plots ...
  }

  ## Slicing information

  # overall slice
  slicing_specs {}

  # slice specific features
  slicing_specs {
    feature_keys: ["sex"]
  }
  slicing_specs {
    feature_keys: ["race"]
  }

  # slice specific values from features
  slicing_specs {
    feature_values: {
      key: "native-country"
      value: "Cambodia"
    }
  }
  slicing_specs {
    feature_values: {
```

```
  }
  slicing_specs {
    feature_values: {
      key: "native-country"
      value: "Canada"
    }
  }

  # slice feature crosses
  slicing_specs {
    feature_keys: ["sex", "race"]
  }
""", tfma.EvalConfig())
```

```
# Specify output path for the evaluation results
OUTPUT_DIR = os.path.join(BASE_DIR, 'output')

# Run TFMA. You can ignore the warnings generated.
eval_result = tfma.run_model_analysis(
    eval_shared_model=eval_shared_model,
    eval_config=eval_config,
    data_location=TFRECORD_FULL,
    output_path=OUTPUT_DIR)
WARNING:tensorflow:SavedModel saved prior to TF 2.5 detec
```

- Config based*, difficult to iterate.
- Need to learn a special DSL you can't use anywhere else.
- Data must be in TFRecord format. Additional prerequisites.

* You can use more python than what is shown here, but it's still incredibly difficult.

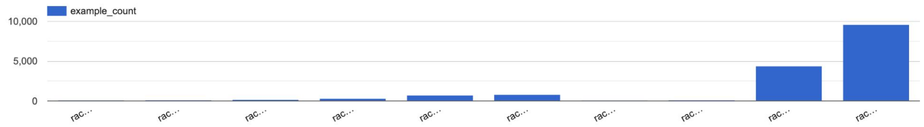


Turn what is usually code into a giant config file!

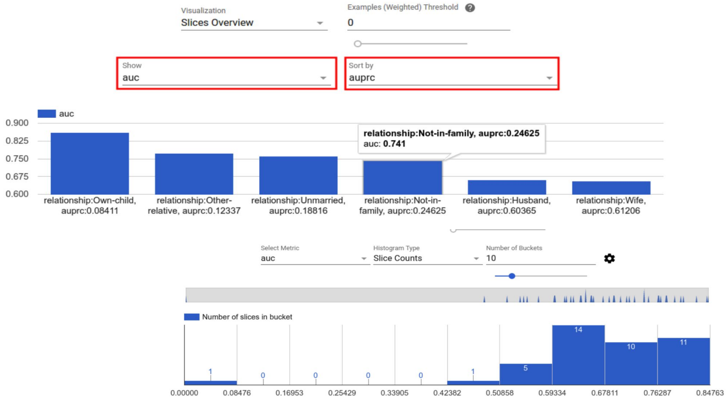
Model Validation (TFMA)

```python
# Render metrics for feature crosses
tfma.view.render_slicing_metrics(
    eval_result,
    slicing_spec=tfma.SlicingSpec(
        feature_keys=['sex', 'race']))
```

- Nothing new here
- Completely Siloed From Tensorboard
- Metrics on slices
- “Tableau in a notebook” without the flexibility



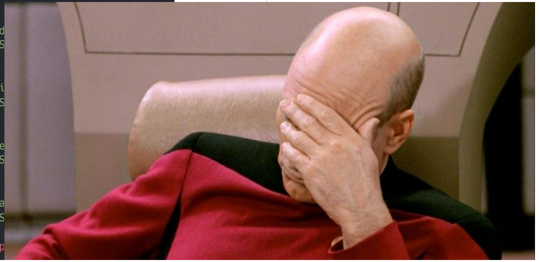
feature	auc	auc_precision_recall	binary_accuracy	binary_crossentropy	calibration	example_count	loss	mean_label	mean_prediction	precision
race_X_sex:Black_X_Male	0.80595	0.45214	0.83787	0.86669	1.04805	808	0.86669	0.16955	0.17770	0.52381
race_X_sex:White_X_Male	0.83771	0.69010	0.78433	0.62535	1.05650	9561	0.62535	0.31126	0.32885	0.6589
race_X_sex:White_X_Female	0.87137	0.54850	0.90490	0.39942	0.96108	4385	0.39942	0.11722	0.11266	0.6114
race_X_sex:Asian-Pac-Islander_X_Male	0.76140	0.59973	0.73463	2.30789	1.07607	309	2.30789	0.34628	0.37262	0.6087
race_X_sex:Black_X_Female	0.81028	0.44237	0.94954	0.42786	0.74667	753	0.42786	0.05578	0.04165	0.57143



TFMA Fairness Indicators: Another Viz & Slicing Library

```
# Specify Fairness Indicators in eval_config.
eval_config = text_format.Parse("""
  model_specs {
    prediction_key: 'dnn_bar_pass_prediction',
    label_key: 'pass_bar'
  }
  metrics_specs {
    metrics {class_name: "AUC"}
    metrics {
      class_name: "FairnessIndicators"
      config: '{"thresholds": [0.50, 0.90]}'
    }
  }
  slicing_specs {
    feature_keys: 'race1'
  }
  slicing_specs {}
  """, tfma.EvalConfig())

# Run TensorFlow Model Analysis.
eval_result = tfma.analyze_raw_data(
  data=_LSAT_DF,
  eval_config=eval_config,
  output_path=_DATA_ROOT)
```

```
xt_format.Parse(

sor_representation_group {
ey: ""
alue {
 tensor_representation {
   key: "comment_text"
   value {
     dense_tensor {
       column_name: "comment_text"
       shape {}
     }
   }
 }

ture {
ame: "comment_text"
ype: BYTES

ture {
ame: "toxicity"
ype: FLOAT

ture {
ame: "sexual_orientation"
ype: BYTES

ture {
ame: "gend
ype: BYTES

ture {
ame: "reli
ype: BYTES

ture {
ame: "race
ype: BYTES

feature {
  name: "disa
  type: BYTES
}
""", schema_p
```



```
1  # Render Fairness Indicators.
2  tfma.addons.fairness.view.widget_view.render_fairness_indicator(eval_result)
```

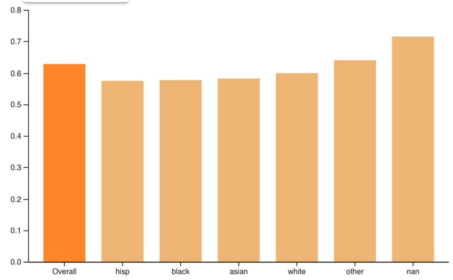
Select metrics to display:

- [ ] Select all
- [x] **auc**
- [ ] false_positive_rate
- [ ] false_negative_rate
- [ ] false_discovery_rate
- [ ] false_omission_rate
- [ ] negative_rate
- [ ] positive_rate
- [ ] true_negative_rate
- [ ] true_positive_rate

Baseline
Overall

auc

SELECT SLICES



feature	auc	auc against Overall	example count
Overall	0.629	0%	NO_DATA
race1:asian	0.582	↓ -7.449%	NO_DATA
race1:black	0.578	↓ -8.063%	NO_DATA
race1:hisp	0.575	↓ -8.462%	NO_DATA
race1:nan	0.714	↑ 13.629%	NO_DATA
race1:other	0.639	↑ 1.645%	NO_DATA
race1:white	0.599	↓ -4.743%	NO_DATA

More config-based visualizations

But Python Has Great DataViz Tools

matplotlib

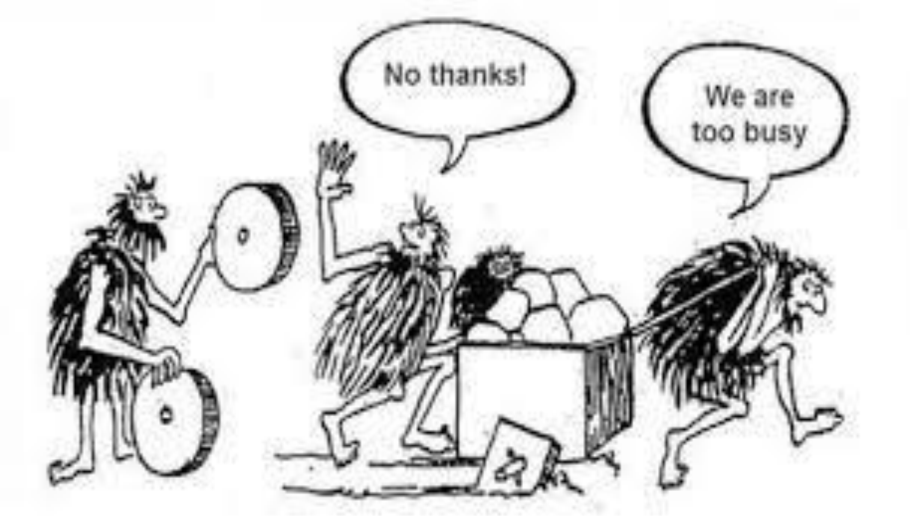
Altair

Declarative Visualization in Python

bokeh

pandas

seaborn



Metrics on “slices” Is Already Easy w/ Existing Tools

And more flexible!

```
from sklearn.metrics import (
    accuracy_score as acc,
    roc_auc_score as roc)

df.groupby('sex').apply(lambda x: pd.Series({
    "Accuracy": acc(x["label"], x["prediction"]),
    "AUC": acc(x["label"], x["prediction"])
}))
```

	Accuracy	AUC
sex		
Female	0.505861	0.505861
Male	0.494292	0.494292

Tool Myopia Can Lead To Blindspots

"... People don't use PyTorch for production, no **real** applied ML occurs in pytorch "

"data drift and model drift are a completely solved problem in TFX"

Me

Skew Detection in TFX: Summary Statistics

```
# Calculate skew for the diabetesMed feature
diabetes_med = tfdv.get_feature(schema, 'diabetesMed')
diabetes_med.skew_comparator.infinity_norm.threshold = 0.03 # domain knowledge helps

# Calculate drift for the payer_code feature
payer_code = tfdv.get_feature(schema, 'payer_code')
payer_code.drift_comparator.infinity_norm.threshold = 0.03 # domain knowledge helps t

# Calculate anomalies
skew_drift_anomalies = tfdv.validate_statistics(train_stats, schema,
                                          previous_statistics=eval_stats,
                                          serving_statistics=serving_stats)

# Display anomalies
tfdv.display_anomalies(skew_drift_anomalies)
```

Categorical: **L-Infinity Norm**
Numeric: **Jensen-Shannon Divergence**

Feature name	Anomaly short description	Anomaly long description
'payer_code'	High Linfty distance between current and previous	The Linfty distance between current and previous is 0.0342144 (up to six significant digits), above the threshold 0.03. The feature value with maximum difference is: MC
'diabetesMed'	High Linfty distance between training and serving	The Linfty distance between training and serving is 0.0325464 (up to six significant digits), above the threshold 0.03. The feature value with maximum difference is: No

Skew Detection Is Very Limited & Not Actionable

Feature name	Anomaly short description	Anomaly long description
'payer_code'	High Linfty distance between current and previous	The Linfty distance between current and previous is 0.0342144 (up to six significant digits), above the threshold 0.03. The feature value with maximum difference is: MC
'diabetesMed'	High Linfty distance between training and serving	The Linfty distance between training and serving is 0.0325464 (up to six significant digits), above the threshold 0.03. The feature value with maximum difference is: No

Categorical Features (37)

	count	missing	unique	top	freq top	avg str len
payer_code						
	34.1k	**52.11%**	16	MC	18.4k	2
	14.2k	**7%**	16	MC	7,191	2

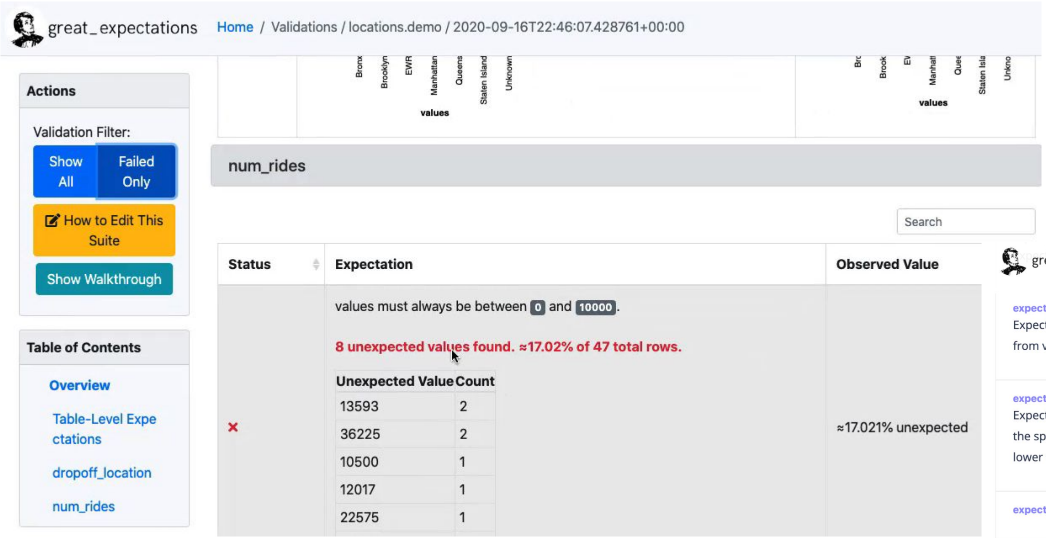
Chart
Star
log
SHOW F
0.2

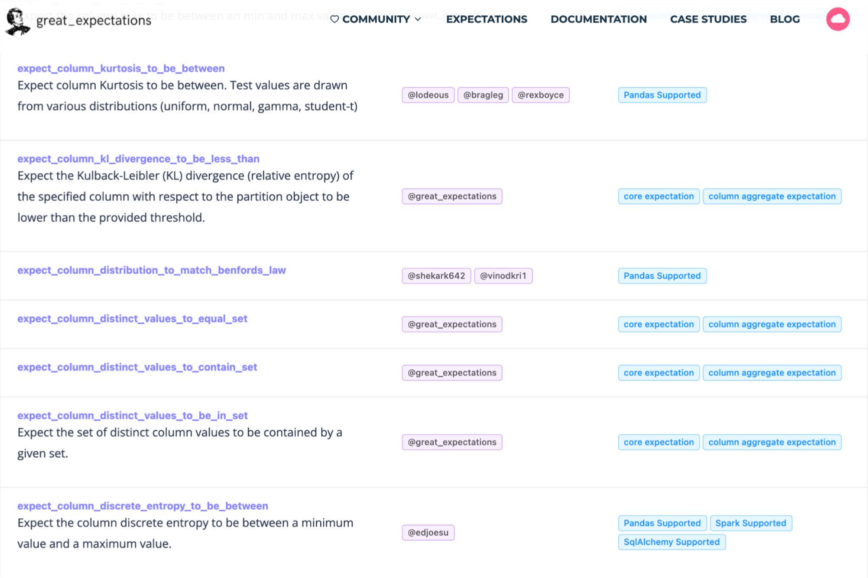
Not clear which summary stats are included in the calculation of L-infinity norm.

Univariate - doesn't account for interactions

Not Very Actionable. **What Now?**

Other tools can offer more visibility and options





A Practical Approach To Detect Skew: Adversarial Validation

Training	Serving

1. Train a model to discriminate between the train / serving set
2. If there is any predictive power there is drift

You can use existing model interpretability tools to figure out whats causing the drift. **No new infra or tools necessary.**

Not limited to univariate analysis, will catch complex interactions between features.

Adversarial Validation Approach to Concept Drift Problem in User Targeting Automation Systems at Uber

Jing Pan
Uber Technologies
San Francisco, California
jing.pan@uber.com

Vincent Pham
Uber Technologies
San Francisco, California
vincent.pham@uber.com

Mohan Dorairaj
Uber Technologies
San Francisco, California
mohan@uber.com

Huigang Chen
Uber Technologies
San Francisco, California
huigang@uber.com

Jeong-Yoon Lee
Uber Technologies
San Francisco, California
jeong@uber.com

Bojan Tunguz
@tunguz

Replying to @sh_reya

I am not familiar with the Uber paper, but I have used adversarial validation with great success with a variety of datasets and problems. And with the modern GPU acceleration (wink, wink) retraining even the biggest models can be done in a relatively short amount of time.

Developer Ergonomics
The Gold Standard: Keras

One of the most loved ML APIs ever designed.

Documentation is more important than features

"The one thing that's super important is investing high quality documentation compared to developing new features."

Source: https://www.youtube.com/watch?v=4tO3TfL0QzY

TFX: Documentation Is Often Missing

infer_output_type

```
infer_output_type(
    unused_input_type
)
```

register_urn

```
@classmethod
register_urn(
    urn, parameter_type, constructor=None
)
```

runner_api_requires_keyed_input

```
runner_api_requires_keyed_input()
```

to_runner_api

```
to_runner_api(
    context, has_parts=False, **extra_kwargs
)
```

tfma.post_export_metrics.auc_plots

View source on GitHub

This is the function that the user calls.

```
tfma.post_export_metrics.auc_plots(
    *args, **kwargs
)
```

Was this helpful?

Reduce cognitive load and boilerplate

"If the cognitive load of a workflow is sufficiently low, it should be possible for a user to go through it from memory **without looking up a tutorial or documentation** after having done it **once or twice**".

Source: https://www.youtube.com/watch?v=4tO3TfL0QzY

Confusing API: removing an item from a set

```
tfdv.get_feature(schema, 'Cover_Type').not_in_environment.append('SERVING')
```

Would it have been possible to remove this item in a more pythonic way?

Special DSLs for simple operations can make it really hard to remember how to do things.

Entry level paths often do not exist: Ex. Metadata store

```
connection_config = metadata_store_pb2.ConnectionConfig()


connection_config.sqlite.filename_uri = '...'
connection_config.sqlite.connection_mode = 3 # READWRITE_OPENCREATE


store = metadata_store.MetadataStore(connection_config)
```

Setup is painful. This is something done automatically for you in Metaflow & MLFlow using sensible defaults.

Doing a Simple Train/Validation Split

```
# Input has a single split 'input_dir/*'.
# Output 2 splits: train:eval=3:1.
output = proto.Output(
             split_config=example_gen_pb2.SplitConfig(splits=[
                 proto.SplitConfig.Split(name='train', hash_buckets=3),
                 proto.SplitConfig.Split(name='eval', hash_buckets=1)
             ]))
example_gen = CsvExampleGen(input_base=input_dir, output_config=output)
```

Tons of boilerplate to perform the most simple operation.

Source: https://www.tensorflow.org/tfx/guide/examplegen

Progressive disclosure of complexity

“A key design principle I follow in libraries (e.g. Keras) is "progressive disclosure of complexity". **Make it easy to get started**, **yet make it possible to handle arbitrarily flexible use cases, only requiring incremental learning** at each step”.

Source: https://twitter.com/fchollet/status/1231285340335267840

“Hello World” and Paved Paths Use Apache Beam

```
import os
from typing import Optional, Text, List
from absl import logging
from ml_metadata.proto import metadata_store_pb2
import tfx.v1 as tfx

PIPELINE_NAME = 'my_pipeline'
PIPELINE_ROOT = os.path.join('.', 'my_pipeline_output')
METADATA_PATH = os.path.join('.', 'tfx_metadata', PIPELINE_NAME, 'metadata.db')
ENABLE_CACHE = True

def create_pipeline(
  pipeline_name: Text,
  pipeline_root:Text,
  enable_cache: bool,
  metadata_connection_config: Optional[
    metadata_store_pb2.ConnectionConfig] = None,
  beam_pipeline_args: Optional[List[Text]] = None
):
  components = []

  return tfx.dsl.Pipeline(
        pipeline_name=pipeline_name,
        pipeline_root=pipeline_root,
        components=components,
        enable_cache=enable_cache,
        metadata_connection_config=metadata_connection_config,
        beam_pipeline_args=beam_pipeline_args,
    )

def run_pipeline():
  my_pipeline = create_pipeline(
      pipeline_name=PIPELINE_NAME,
      pipeline_root=PIPELINE_ROOT,
      enable_cache=ENABLE_CACHE,
      metadata_connection_config=tfx.orchestration.metadata.sqlite_m
      )

  tfx.orchestration.LocalDagRunner().run(my_pipeline)

if __name__ == '__main__':
  logging.set_verbosity(logging.INFO)
  run_pipeline()
```

Distributed computing and new DSLs from the beginning vs using existing tools.

Jump straight into the deep end of complexity

Orchestrating TFX Pipelines

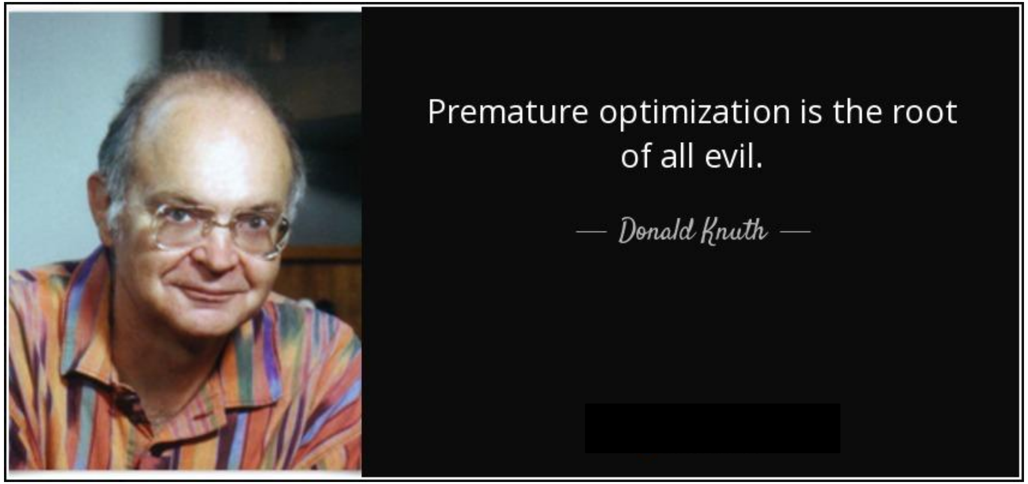
Apache Beam

Several TFX components rely on Beam for distributed data processing. In addition, TFX can use Apache Beam to orchestrate and execute the pipeline DAG. Beam orchestrator uses a different BeamRunner than the one which is used for component data processing. With the default DirectRunner setup the Beam orchestrator can be used for local

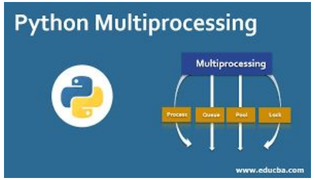
BUT WAIT

The Scalability Argument

But we need scale. Apache Beam allows us to scale.



DASK



+ Large VM

The Portability/Reproducability Argument

If you don't tie your transforms to your model you get train serving skew

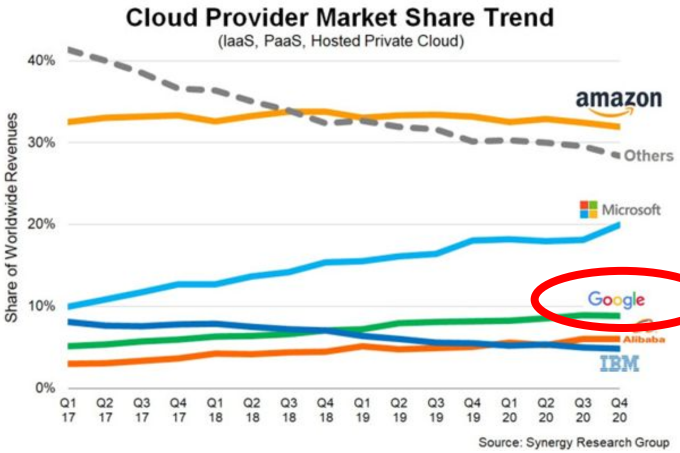
The Don’t Use It Argument

“You don’t have to use tools like TF Transform until you are ready to make pipelines”

I’d rather not mess with this in the first place

This is all easy with GCP: Vertex AI Pipelines

I use Vertex AI Pipelines (GCP) which makes TFX easy



< 10% market share

Invest your time and skills like this

But Google Is Smart

Google has smart people.

You can enjoy success with ML like them if you use their tools.

This is a religious argument

“Use their tools” Is very effective marketing

Final Thoughts

The Irony

The fastest way to accrue debt is to cargo cult these tools

**Machine Learning:
The High-Interest Credit Card of Technical Debt**

**D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov,
Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young**
{dsculley,gholt,dgg,edavydov}@google.com
{toddphillips,ebner,vchaudhary,mwyoung}@google.com
Google, Inc

Abstract

Machine learning offers a fantastically powerful toolkit for building complex systems quickly. This paper argues that it is dangerous to think of these quick wins as coming for free. Using the framework of *technical debt*, we note that it is remarkably easy to incur massive ongoing maintenance costs at the system level when applying machine learning. The goal of this paper is highlight several machine learning specific risk factors and design patterns to be avoided or refactored where possible. These include boundary erosion, entanglement, hidden feedback loops, undeclared consumers, data dependencies, changes in the external world, and a variety of system-level anti-patterns.

Hidden Technical Debt in Machine Learning Systems

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips
{dsculley,gholt,dgg,edavydov,toddphillips}@google.com
Google, Inc.

Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, Dan Dennison
{ebner,vchaudhary,mwyoung,jfcrespo,dennison}@google.com
Google, Inc.

Abstract

Machine learning offers a fantastically powerful toolkit for building useful complex prediction systems quickly. This paper argues it is dangerous to think of these quick wins as coming for free. Using the software engineering framework of *technical debt*, we find it is common to incur massive ongoing maintenance costs in real-world ML systems. We explore several ML-specific risk factors to account for in system design. These include boundary erosion, entanglement, hidden feedback loops, undeclared consumers, data dependencies, configuration issues, changes in the external world, and a variety of system-level anti-patterns.

When Would You Recommend TFX?

- Not binary: I love some components and dislike others.
- Overall, **<u>probably a bad fit for the vast majority of people.</u>**

- Might be a good fit if:
  - Very committed to TF
  - Running on GCP (beam, vertex ai pipelines)
  - Need to deploy to edge devices
  - Scale: Optimization is more expensive vs developer time
  - **Even then, not convinced is a great fit for many people.**

Don’t Become A Tool Zealot

- It will **narrow the way you think** about problems.
- Introduces **bias towards working on certain tasks** over others (i.e. data cleaning vs crafting model architectures).
- Will **prevent you from hiring diverse talent.**
- Can lead to blindspots (ex: data drift, fairness ,etc)**.**
- Try new tools often, especially ones that use a different approach.
- These are the same reasons you should learn other programming languages

… Even if you have no time to try other things, keep an open mind