Ungraded Lab: Model Analysis with TFX Evaluator

Now that you've used TFMA as a standalone library in the previous lab, you will now see how it is used by TFX with its [Evaluator](https://www.tensorflow.org/tfx/guide/evaluator) component. This component comes after your `Trainer` run and it checks if your trained model meets the minimum required metrics and also compares it with previously generated models.

You will go through a TFX pipeline that prepares and trains the same model architecture you used in the previous lab. As a reminder, this is a binary classifier to be trained on the Census Income dataset. Since you're already familiar with the earlier TFX components, we will just go over them quickly but we've placed notes on where you can modify code if you want to practice or produce a better result.

Let's begin!

Credits: Some of the code and discussions are based on the TensorFlow team's official tutorial.

Setup

Install TFX

```
!pip install -U pip
!pip install -U tfx==1.3

# These are downgraded to work with the packages used by TFX 1
# Please do not delete because it will cause import errors in
!pip install --upgrade tensorflow-estimator==2.6.0
!pip install --upgrade keras==2.6.0
```

Note: In Google Colab, you need to restart the runtime at this point to finalize updating the packages you just installed. You can do so by clicking the `Restart Runtime` at the end of the output cell above (after installation), or by selecting `Runtime > Restart Runtime` in the Menu bar. ***Please do not proceed to the next section without restarting.*** *You can also ignore the errors about version incompatibility of some of the bundled packages because we won't be using those in this notebook.*

Imports

```
import os
import pprint

import tensorflow as tf
import tensorflow_model_analysis as tfma
from tfx import v1 as tfx

from tfx.orchestration.experimental.interactive.interactive_co

tf.get_logger().propagate = False
tf.get_logger().setLevel('ERROR')
pp = pprint.PrettyPrinter()
```

Set up pipeline paths

```
# Location of the pipeline metadata store
_pipeline_root = './pipeline/'

# Directory of the raw data files
_data_root = './data/census'

_data_filepath = os.path.join(_data_root, "data.csv")
```

```
# Create the TFX pipeline files directory
!mkdir {_pipeline_root}

# Create the dataset directory
!mkdir -p {_data_root}
```

```
mkdir: cannot create directory './pipeline/': File exists
```

Download and prepare the dataset

Here, you will download the training split of the [Census Income Dataset](). This is twice as large as the test dataset you used in the previous lab.

```
# Define filename and URL
TAR_NAME = 'C3_W4_Lab_2_data.tar.gz'
DATA_PATH = f'https://storage.googleapis.com/mlep-public/cours
```

```

# Download dataset
!wget -nc {DATA_PATH}

# Extract archive
!tar xvzf {TAR_NAME}

# Delete archive
!rm {TAR_NAME}
```

```
--2022-01-01 20:30:33--  https://storage.googleapis.com/mlep-public/course_3/wee
Resolving storage.googleapis.com (storage.googleapis.com)... 172.217.2.112, 172.
Connecting to storage.googleapis.com (storage.googleapis.com)|172.217.2.112|:443
HTTP request sent, awaiting response... 200 OK
Length: 418898 (409K) [application/x-gzip]
Saving to: 'C3_W4_Lab_2_data.tar.gz'

C3_W4_Lab_2_data.ta 100%[===================>] 409.08K  --.-KB/s    in 0.02s

2022-01-01 20:30:33 (18.8 MB/s) - 'C3_W4_Lab_2_data.tar.gz' saved [418898/418898

./data/census/data.csv
```

Take a quick look at the first few rows.

```
# Preview dataset
!head {_data_filepath}
```

```
age,workclass,fnlwgt,education,education-num,marital-status,occupation,relations
39,State-gov,77516,Bachelors,13,Never-married,Adm-clerical,Not-in-family,White,M
50,Self-emp-not-inc,83311,Bachelors,13,Married-civ-spouse,Exec-managerial,Husban
38,Private,215646,HS-grad,9,Divorced,Handlers-cleaners,Not-in-family,White,Male,
53,Private,234721,11th,7,Married-civ-spouse,Handlers-cleaners,Husband,Black,Male
28,Private,338409,Bachelors,13,Married-civ-spouse,Prof-specialty,Wife,Black,Fema
37,Private,284582,Masters,14,Married-civ-spouse,Exec-managerial,Wife,White,Femal
49,Private,160187,9th,5,Married-spouse-absent,Other-service,Not-in-family,Black,
52,Self-emp-not-inc,209642,HS-grad,9,Married-civ-spouse,Exec-managerial,Husband,
31,Private,45781,Masters,14,Never-married,Prof-specialty,Not-in-family,White,Fem
```

TFX Pipeline

Create the InteractiveContext

As usual, you will initialize the pipeline and use a local SQLite file for the metadata store.

```
# Initialize InteractiveContext
context = InteractiveContext(pipeline_root=_pipeline_root)
```

```
WARNING:absl:InteractiveContext metadata_connection_config not provided: using S
```

ExampleGen

You will start by ingesting the data through `CsvExampleGen`. The code below uses the default 2:1 train-eval split (i.e. 33% of the data goes to eval) but feel free to modify if you want. You can review splitting techniques [here](#).

```
# Run CsvExampleGen
example_gen = tfx.components.CsvExampleGen(input_base=_data_ro
context.run(example_gen)
```

```
WARNING:apache_beam.runners.interactive.interactive_environment:Dependencies req
WARNING:root:Make sure that locally built Python SDK docker image has Python 3.7
WARNING:apache_beam.io.tfrecordio:Couldn't find python-snappy so the implementat
```

▼ExecutionResult at 0x7f8567196b10

.execution_id 1

.component **▼CsvExampleGen** at 0x7f84e2ceef50

```
# Print split names and URI
artifact = example_gen.outputs['examples'].get()[0]
print(artifact.split_names, artifact.uri)
```

```
["train", "eval"] ./pipeline/CsvExampleGen/examples/1
```

[0] **▼Artifact** of type '**Exa**

StatisticsGen

You will then compute the statistics so it can be used by the next components.

```
# Run StatisticsGen
statistics_gen = tfx.components.StatisticsGen(
    examples=example_gen.outputs['examples'])
context.run(statistics_gen)
```

```
WARNING:root:Make sure that locally built Python SDK docker image has Python 3.7
```

▼**ExecutionResult** at 0x7f84e60a1f50

.execution_id 2

.component

▼**StatisticsGen** at 0x7f84e339ff50

.inputs

['examples'] ▼**Channel** of type **'Examples'** (1 artifact)

.type_name Examples

._artifacts

[0] ▼**Artifact** of type **'Exa**
./pipeline/CsvExample(

.type	<class '
.uri	./pipelin
.span	0
.split_names	["train",
.version	0

.outputs

['statistics'] ▼**Channel** of type **'ExampleStatistics'** (1

.type_name ExampleStatistics

._artifacts

[0] ▼**Artifact** of type **'Exar**
./pipeline/StatisticsGen/

.type	<class 'tfx.types
.uri	./pipeline
.span	0
.split_names	["train", '

You can look at the visualizations below if you want to explore the data some more.

```
# Visualize statistics
context.show(statistics_gen.outputs['statistics'])
```

Artifact at ./pipeline/StatisticsGen/statistics/2

'train' split:

Sort by

Feature order ▾ ☐ Reverse order Feature search (regex enabled)

Features: ☐ int(7) ☐ string(8)

Numeric Features (7)

Ch
St

	count	missing	mean	std dev	zeros	min	median	max
age	21.8k	0%	38.61	13.62	0%	17	37	90
capital-gain	21.8k	0%	1,114.65	7,616.76	91.63%	0	0	100k
capital-loss	21.8k	0%	89.15	406.88	95.22%	0	0	4,356
education-num	21.8k	0%	10.08	2.58	0%	1	10	16

1

'eval' split:

Sort by

Feature order ▾ ☐ Reverse order Feature search (regex enabled)

Features: ☐ int(7) ☐ string(8)

Numeric Features (7)

Ch
St

	count	missing	mean	std dev	zeros	min	median	max
age	10.8k	0%	38.53	13.68	0%	17	37	90
capital-gain	10.8k	0%	1,003.14	6,894.74	91.74%	0	0	100k
capital-loss								

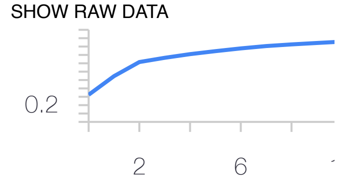
4

	10.8k	0%	83.58	394.91	95.57%	0	0	4,356
education-num	10.8k	0%	10.09	2.55	0%	1	10	16
fnlwgt	10.8k	0%	191k	105k	0%	19.3k	179k	1.23M
hours-per-week	10.8k	0%	40.33	12.39	0%	1	40	99
label	10.8k	0%	0.24	0.43	76.01%	0	0	1

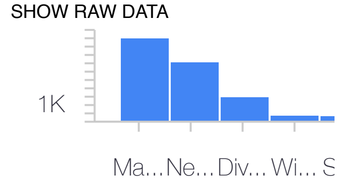
Categorical Features (8)	count	missing	unique	top	freq top	avg str len
education	10.8k	0%	16	HS-grad	3,498	8.43
marital-status	10.8k	0%	7	Married-…	4,972	14.43
native-country	10.8k	0%	41	United-S…	9,684	12.3
occupation	10.8k	0%	15	Prof-spe…	1,392	12.16

Chart to show
Standard
log expand

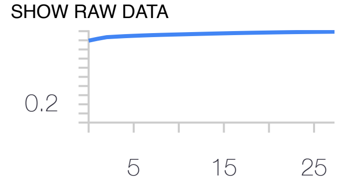
SHOW RAW DATA



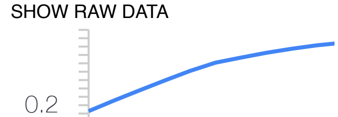
SHOW RAW DATA

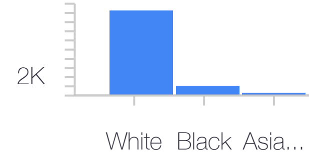


SHOW RAW DATA



SHOW RAW DATA



race	10.8k	0%	5	White	9,274	5.5	SHOW RAW DATA 
relationship	10.8k	0%	6	Husband	4,372	9.11	SHOW RAW DATA 
sex	10.8k	0%	2	Male	7,232	4.66	SHOW RAW DATA 

SchemaGen

You can then infer the dataset schema with SchemaGen. This will be used to validate incoming data to ensure that it is formatted correctly.

```
# Run SchemaGen
schema_gen = tfx.components.SchemaGen(
    statistics=statistics_gen.outputs['statistics'])
context.run(schema_gen)
```

▼ExecutionResult at 0x7f84e1309950

.execution_id 3

.component

▼SchemaGen at 0x7f84e129a990

.inputs **['statistics']**

▼Channel of type **'ExampleStatistics'** (1

.type_name ExampleStatistics

._artifacts **[0]**

▼Artifact of type **'Exar**
./pipeline/StatisticsGen/

.type	<class 'tfx.types
.uri	./pipeline
.span	0
.split_names	["train", '

.outputs **['schema']**

▼Channel of type **'Schema'** (1 artifact) at 0

.type_name Schema

._artifacts **[0]**

▼Artifact of type **'Scher**
./pipeline/SchemaGen/sc

.type	<class 'tfx.types.s
.uri	./pipeline/Schema

.exec_properties

['infer_feature_shape']	1
['exclude_splits']	[]

For simplicity, you will just accept the inferred schema but feel free to modify with the [TFDV API](#) if you want.

```
# Visualize the inferred Schema
context.show(schema_gen.outputs['schema'])
```

Artifact at ./pipeline/SchemaGen/schema/3

Feature name	Type	Presence	Valency	Domain
'age'	INT	required		-
'capital-gain'	INT	required		-
'capital-loss'	INT	required		-
'education'	STRING	required		'education'
'education-num'	INT	required		-
'fnlwgt'	INT	required		-
'hours-per-week'	INT	required		-
'label'	INT	required		-
'marital-status'	STRING	required		'marital-status'
'native-country'	STRING	required		'native-country'
'occupation'	STRING	required		'occupation'
'race'	STRING	required		'race'
'relationship'	STRING	required		'relationship'
'sex'	STRING	required		'sex'
'workclass'	STRING	required		'workclass'

```
/usr/local/lib/python3.7/dist-packages/tensorflow_data_validation/utils/display_
  pd.set_option('max_colwidth', -1)
```

Domain	Values
'education'	'10th', '11th', '12th', '1st-4th', '5th-6th', '7th-8th', '9th', 'Assoc-acdm', 'Assoc-voc', 'Bachelors', 'Doctorate', 'HS-grad', 'Masters', 'Preschool', 'Prof-school', 'Some-college'
'marital-status'	'Divorced', 'Married-AF-spouse', 'Married-civ-spouse', 'Married-spouse-absent', 'Never-married', 'Separated', 'Widowed'
'native-country'	'?', 'Cambodia', 'Canada', 'China', 'Columbia', 'Cuba', 'Dominican-Republic', 'Ecuador', 'El-Salvador', 'England', 'France', 'Germany', 'Greece', 'Guatemala', 'Haiti', 'Holand-Netherlands', 'Honduras', 'Hong', 'Hungary', 'India', 'Iran', 'Ireland', 'Italy', 'Jamaica', 'Japan', 'Laos', 'Mexico', 'Nicaragua', 'Outlying-US(Guam-USVI-etc)', 'Peru', 'Philippines', 'Poland', 'Portugal', 'Puerto-Rico', 'Scotland', 'South', 'Taiwan', 'Thailand',

ExampleValidator

Next, run `ExampleValidator` to check if there are anomalies in the data.

```
# Run ExampleValidator
example_validator = tfx.components.ExampleValidator(
    statistics=statistics_gen.outputs['statistics'],
    schema=schema_gen.outputs['schema'])
context.run(example_validator)
```

▼**ExecutionResult** at 0x7f84e133d110

.execution_id 4

.component

▼**ExampleValidator** at 0x7f84e133d850

.inputs

['statistics']

▼**Channel** of type **'ExampleStatistics'** (1

.type_name ExampleStatistics

._artifacts **[0]**

▼**Artifact** of type **'Exar**
./pipeline/StatisticsGen/

.type <class 'tfx.types

.uri ./pipeline

.span 0

.split_names ["train", '

['schema']

▼**Channel** of type **'Schema'** (1 artifact) at

.type_name Schema

._artifacts **[0]**

▼**Artifact** of type **'Sche**
./pipeline/SchemaGen/s

.type <class 'tfx.types.

.uri ./pipeline/Schem

.outputs

['anomalies']

▼**Channel** of type **'ExampleAnomalies'**

.type_name ExampleAnomalies

._artifacts **[0]**

▼**Artifact** of type **'Exa**
./pipeline/ExampleVali

If you just used the inferred schema, then there should not be any anomalies detected. If you modified the schema, then there might be some results here and you can again use TFDV to modify or relax constraints as needed.

In actual deployments, this component will also help you understand how your data evolves over time and identify data errors. For example, the first batches of data that you get from your users might conform to the schema but it might not be the case after several months. This component will detect that and let you know that your model might need to be updated.

```
# Check results
context.show(example_validator.outputs['anomalies'])
```

Artifact at ./pipeline/ExampleValidator/anomalies/4

'train' split:

```
/usr/local/lib/python3.7/dist-packages/tensorflow_data_validation/utils/display_
  pd.set_option('max_colwidth', -1)
```

No anomalies found.

'eval' split:

No anomalies found.

Transform

Now you will perform feature engineering on the training data. As shown when you previewed the CSV earlier, the data is still in raw format and cannot be consumed by the model just yet. The transform code in the following cells will take care of scaling your numeric features and one-hot encoding your categorical variables.

Note: If you're running this exercise for the first time, we advise that you leave the transformation code as is. After you've gone through the entire notebook, then you can modify these for practice and see what results you get. Just make sure that your model builder code in the `Trainer` component will also reflect those changes if needed. For example, removing a feature here should also remove an input layer for that feature in the model.

```
# Set the constants module filename
_census_constants_module_file = 'census_constants.py'
```

```
%%writefile {_census_constants_module_file}

# Features with string data types that will be converted to in
VOCAB_FEATURE_DICT = {
    'education': 16, 'marital-status': 7, 'occupation': 15, 'r
    'relationship': 6, 'workclass': 9, 'sex': 2, 'native-count
}

# Numerical features that are marked as continuous
NUMERIC_FEATURE_KEYS = ['fnlwgt', 'education-num', 'capital-ga

# Feature that can be grouped into buckets
BUCKET_FEATURE_DICT = {'age': 4}

```

```
# Number of out-of-vocabulary buckets
NUM_OOV_BUCKETS = 5

# Feature that the model will predict
LABEL_KEY = 'label'
```

```
Writing census_constants.py
```

```
# Set the transform module filename
_census_transform_module_file = 'census_transform.py'
```

```
%%writefile {_census_transform_module_file}

import tensorflow as tf
import tensorflow_transform as tft

# import constants from cells above
import census_constants

# Unpack the contents of the constants module
_NUMERIC_FEATURE_KEYS = census_constants.NUMERIC_FEATURE_KEYS
_VOCAB_FEATURE_DICT = census_constants.VOCAB_FEATURE_DICT
_BUCKET_FEATURE_DICT = census_constants.BUCKET_FEATURE_DICT
_NUM_OOV_BUCKETS = census_constants.NUM_OOV_BUCKETS
_LABEL_KEY = census_constants.LABEL_KEY

# Define the transformations
def preprocessing_fn(inputs):
    """tf.transform's callback function for preprocessing inpu
    Args:
        inputs: map from feature keys to raw not-yet-transform
    Returns:
        Map from string feature key to transformed feature ope
    """

    # Initialize outputs dictionary
    outputs = {}

    # Scale these features to the range [0,1]
    for key in _NUMERIC_FEATURE_KEYS:
        scaled = tft.scale_to_0_1(inputs[key])
        outputs[key] = tf.reshape(scaled, [-1])
```

```

    # Convert strings to indices and convert to one-hot vector
    for key, vocab_size in _VOCAB_FEATURE_DICT.items():
        indices = tft.compute_and_apply_vocabulary(inputs[key]
        one_hot = tf.one_hot(indices, vocab_size + _NUM_OOV_BU
        outputs[key] = tf.reshape(one_hot, [-1, vocab_size + _

    # Bucketize this feature and convert to one-hot vectors
    for key, num_buckets in _BUCKET_FEATURE_DICT.items():
        indices = tft.bucketize(inputs[key], num_buckets)
        one_hot = tf.one_hot(indices, num_buckets)
        outputs[key] = tf.reshape(one_hot, [-1, num_buckets])

    # Cast label to float
    outputs[_LABEL_KEY] = tf.cast(inputs[_LABEL_KEY], tf.float

    return outputs
```

```
Writing census_transform.py
```

Now, we pass in this feature engineering code to the `Transform` component and run it to transform your data.

```
# Run the Transform component
transform = tfx.components.Transform(
    examples=example_gen.outputs['examples'],
    schema=schema_gen.outputs['schema'],
    module_file=os.path.abspath(_census_transform_module_file)
context.run(transform, enable_cache=False)
```

```
WARNING:root:This output type hint will be ignored and not used for type-checkin
WARNING:absl:Tables initialized inside a tf.function  will be re-initialized on
WARNING:absl:Tables initialized inside a tf.function  will be re-initialized on
WARNING:absl:Tables initialized inside a tf.function  will be re-initialized on
WARNING:absl:Tables initialized inside a tf.function  will be re-initialized on
WARNING:absl:Tables initialized inside a tf.function  will be re-initialized on
WARNING:absl:Tables initialized inside a tf.function  will be re-initialized on
WARNING:absl:Tables initialized inside a tf.function  will be re-initialized on
WARNING:absl:Tables initialized inside a tf.function  will be re-initialized on
WARNING:absl:Tables initialized inside a tf.function  will be re-initialized on
WARNING:absl:Tables initialized inside a tf.function  will be re-initialized on
WARNING:absl:Tables initialized inside a tf.function  will be re-initialized on
WARNING:absl:Tables initialized inside a tf.function  will be re-initialized on
WARNING:absl:Tables initialized inside a tf.function  will be re-initialized on
WARNING:absl:Tables initialized inside a tf.function  will be re-initialized on
WARNING:absl:Tables initialized inside a tf.function  will be re-initialized on
WARNING:absl:Tables initialized inside a tf.function  will be re-initialized on
WARNING:root:This output type hint will be ignored and not used for type-checkin
WARNING:root:Make sure that locally built Python SDK docker image has Python 3.7
```

▼**ExecutionResult** at 0x7f84e138d350

.execution_id 5

.component

▼**Transform** at 0x7f84e138ded0

.inputs

['examples']

▼**Channel** of type **'Examples'** (1 artifact) a

.type_name Examples

._artifacts **[0]**

▼**Artifact** of type **'Exar**
0x7f85467ff210

.type <class 't
.uri ./pipeline
.span 0
.split_names ["train", '
.version 0

['schema']

▼**Channel** of type **'Schema'** (1 artifact) at

.type_name Schema

._artifacts **[0]**

▼**Artifact** of type **'Sche**
0x7f84e0513050

.type <class 'tfx.types.
.uri ./pipeline/Schem

.outputs

['transform_graph']

▼**Channel** of type **'Tran**

.type_name Transform

._artifacts **[0]**

▼**Ar**

./pipe

.typ

.uri

['transformed_examples']

▼**Channel** of type **'Exa**

.type_name Examples

._artifacts

[0]

▼**Ar**

./pipe

0x7f

.typ

.uri

.spa

.spl

.ver

['updated_analyzer_cache']

▼**Channel** of type **'Tran**

.type_name Transform

._artifacts

[0]

▼**Ar**

./pipe

0x7f

.typ

.uri

['pre_transform_schema']

▼**Channel** of type **'Sch**

.type_name Schema

._artifacts

[0]

▼**Ar**

./pipe

0x7f

.typ

.uri

['pre_transform_stats']

▼**Channel** of type **'Exa**

.type_name ExampleS

._artifacts

[0]

▼**Ar**

./pipe

.typ

.uri

.spa

.spl

.spl

['post_transform_schema']

▼**Channel** of type **'Sch**

.type_name Schema

._artifacts

[0] ▼**Ar**

./pipe

0x7f

.typ

.uri

['post_transform_stats']

▼**Channel** of type **'Exa**

.type_name ExampleS

._artifacts

[0] ▼**Ar**

./pipe

.typ

.uri

.spa

.spl

['post_transform_anomalies']

▼**Channel** of type **'Exa**

.type_name ExampleA

._artifacts

[0] ▼**Ar**

./pipe

0x7f

.typ

.uri

.spa

.spl

.exec_properties

['module_file']	None
['preprocessing_fn']	None
['stats_options_updater_fn']	None
['force_tf_compat_v1']	0
['custom_config']	null
['splits_config']	None
['disable_statistics']	0
['module_path']	census_transform@./pipeli 0.0+b4cdffd259c88473f714 any.whl

.component.inputs

['examples'] ▼**Channel** of type **'Examples'** (1 artifact) at 0x7f84e3471e10

.type_name Examples

._artifacts [0] ▼**Artifact** of type **'Examples'** (uri: ./pipeline

.type	<class 'tfx.types.standard_art
.uri	./pipeline/CsvExampleGen/ex
.span	0
.split_names	["train", "eval"]
.version	0

['schema'] ▶**Channel** of type **'Schema'** (1 artifact) at 0x7f84e129a210

.component.outputs

['transform_graph'] ▼**Channel** of type **'TransformGraph'** (1 artif

.type_name TransformGraph

._artifacts [0] ▶**Artifact** of type **'Transfo**
0x7f84e131fdd0

['transformed_examples'] ▼**Channel** of type **'Examples'** (1 artifact) at

.type_name Examples

._artifacts [0] ▶**Artifact** of type **'Exampl**
0x7f84e132ced0

['updated_analyzer_cache'] ▼**Channel** of type **'TransformCache'** (1 artif

.type_name TransformCache

._artifacts [0] ▶**Artifact** of type **'Transfo**
at 0x7f84e136f2d0

['pre_transform_schema'] ▼**Channel** of type **'Schema'** (1 artifact) at 0x

.type_name Schema

._artifacts [0] ▶**Artifact** of type **'Schema**
0x7f84e136f150

['pre_transform_stats'] ▼**Channel** of type **'ExampleStatistics'** (1 ar

.type_name ExampleStatistics

._artifacts [0] ▶**Artifact** of type **'Exampl**
0x7f84e136f310

['post_transform_schema'] ▼**Channel** of type **'Schema'** (1 artifact) at 0x

.type_name Schema

._artifacts [0] ▶**Artifact** of type **'Schema**

0x7f84e136f050

You can see a sample result for one row with the code below. Notice that the numeric features are indeed scaled and the categorical features are now one-hot encoded.

[0] ▶ **Artifact** of type **'Exampl**

```
# Get the URI of the output artifact representing the transfor
train_uri = os.path.join(transform.outputs['transformed_exampl

# Get the list of files in this directory (all compressed TFRe
tfrecord_filenames = [os.path.join(train_uri, name)
                      for name in os.listdir(train_uri)]

# Create a `TFRecordDataset` to read these files
dataset = tf.data.TFRecordDataset(tfrecord_filenames, compress

# Decode the first record and print output
for tfrecord in dataset.take(1):
  serialized_example = tfrecord.numpy()
  example = tf.train.Example()
  example.ParseFromString(serialized_example)
  pp.pprint(example)
```

```
          value: 0.0
          value: 0.0
        }
      }
    }
    feature {
      key: "relationship"
      value {
        float_list {
          value: 0.0
          value: 1.0
          value: 0.0
          value: 0.0
          value: 0.0
          value: 0.0
          value: 0.0
          value: 0.0
          value: 0.0
          value: 0.0
          value: 0.0
        }
      }
    }
    feature {
      key: "sex"
      value {
        float_list {
          value: 1.0
```

```
          value: 0.0
          value: 0.0
          value: 0.0
          value: 0.0
          value: 0.0
          value: 0.0
        }
      }
    }
    feature {
      key: "workclass"
      value {
        float_list {
          value: 0.0
          value: 0.0
          value: 0.0
          value: 0.0
          value: 1.0
          value: 0.0
          value: 0.0
          value: 0.0
          value: 0.0
          value: 0.0
          value: 0.0
          value: 0.0
          value: 0.0
          value: 0.0
        }
      }
    }
```

As you already know, the `Transform` component not only outputs the transformed examples but also the transformation graph. This should be used on all inputs when your model is deployed to ensure that it is transformed the same way as your training data. Else, it can produce training-serving skew which leads to noisy predictions.

The `Transform` component stores related files in its `transform_graph` output and it would be good to quickly review its contents before we move on to the next component. As shown below, the URI of this output points to a directory containing three subdirectories.

```
# Get URI and list subdirectories
graph_uri = transform.outputs['transform_graph'].get()[0].uri
os.listdir(graph_uri)
```

```
['transformed_metadata', 'metadata', 'transform_fn']
```

- The `transformed_metadata` subdirectory contains the schema of the preprocessed data.
- The `transform_fn` subdirectory contains the actual preprocessing graph.

- The `metadata` subdirectory contains the schema of the original data.

Trainer

Next, you will now build the model to make your predictions. As mentioned earlier, this is a binary classifier where the label is 1 if a person earns more than 50k USD and 0 if less than or equal. The model used here uses the [wide and deep model](#) as reference but feel free to modify after you've completed the exercise. Also for simplicity, the hyperparameters (e.g. number of hidden units) have been hardcoded but feel free to use a `Tuner` component as you did in Week 1 if you want to get some practice.

As a reminder, it is best to start from `run_fn()` when you're reviewing the module file below. The `Trainer` component looks for that function first. All other functions defined in the module are just utility functions for `run_fn()`.

Another thing you will notice below is the `_get_serve_tf_examples_fn()` function. This is tied to the `serving_default` [signature](#) which makes it possible for you to just pass in raw features when the model is served for inference. You have seen this in action in the previous lab. This is done by decorating the enclosing function, `serve_tf_examples_fn()`, with [tf.function](#). This indicates that the computation will be done by first tracing a [Tensorflow graph](#). You will notice that this function uses `model.tft_layer` which comes from `transform_graph` output. Now when you call the `.get_concrete_function()` method on this tf.function in `run_fn()`, you are creating the graph that will be used in later computations. This graph is used whenever you pass in an `examples` argument pointing to a Tensor with `tf.string` dtype. That matches the format of the serialized batches of data you used in the previous lab.

```
# Declare trainer module file
_census_trainer_module_file = 'census_trainer.py'
```

```
%%writefile {_census_trainer_module_file}

from typing import List, Text

import tensorflow as tf
import tensorflow_transform as tft
from tensorflow_transform.tf_metadata import schema_utils

from tfx.components.trainer.fn_args_utils import DataAccessor,
from tfx_bsl.public.tfxio import TensorFlowDatasetOptions

# import same constants from transform module
```

```
import census_constants

# Unpack the contents of the constants module
_NUMERIC_FEATURE_KEYS = census_constants.NUMERIC_FEATURE_KEYS
_VOCAB_FEATURE_DICT = census_constants.VOCAB_FEATURE_DICT
_BUCKET_FEATURE_DICT = census_constants.BUCKET_FEATURE_DICT
_NUM_OOV_BUCKETS = census_constants.NUM_OOV_BUCKETS
_LABEL_KEY = census_constants.LABEL_KEY


def _gzip_reader_fn(filenames):
  '''Load compressed dataset

  Args:
    filenames - filenames of TFRecords to load

  Returns:
    TFRecordDataset loaded from the filenames
  '''

  # Load the dataset. Specify the compression type since it is
  return tf.data.TFRecordDataset(filenames, compression_type='


def _input_fn(file_pattern,
              tf_transform_output,
              num_epochs=None,
              batch_size=32) -> tf.data.Dataset:
  '''Create batches of features and labels from TF Records

  Args:
    file_pattern - List of files or patterns of file paths con
    tf_transform_output - transform output graph
    num_epochs - Integer specifying the number of times to rea
              If None, cycles through the dataset forever.
    batch_size - An int representing the number of records to

  Returns:
    A dataset of dict elements, (or a tuple of dict elements a
    Each dict maps feature keys to Tensor or SparseTensor obje
  '''

```

```
  # Get post-transfrom feature spec
  transformed_feature_spec = (
      tf_transform_output.transformed_feature_spec().copy())

  # Create batches of data
  dataset = tf.data.experimental.make_batched_features_dataset
      file_pattern=file_pattern,
      batch_size=batch_size,
      features=transformed_feature_spec,
      reader=_gzip_reader_fn,
      num_epochs=num_epochs,
      label_key=_LABEL_KEY
      )

  return dataset


def _get_serve_tf_examples_fn(model, tf_transform_output):
  """Returns a function that parses a serialized tf.Example an

  # Get transformation graph
  model.tft_layer = tf_transform_output.transform_features_lay

  @tf.function
  def serve_tf_examples_fn(serialized_tf_examples):
    """Returns the output to be used in the serving signature.
    # Get pre-transform feature spec
    feature_spec = tf_transform_output.raw_feature_spec()

    # Pop label since serving inputs do not include the label
    feature_spec.pop(_LABEL_KEY)

    # Parse raw examples into a dictionary of tensors matching
    parsed_features = tf.io.parse_example(serialized_tf_exampl

    # Transform the raw examples using the transform graph
    transformed_features = model.tft_layer(parsed_features)

    # Get predictions using the transformed features
    return model(transformed_features)

  return serve_tf_examples_fn
```

```


def _build_keras_model(hidden_units: List[int] = None) -> tf.k
  """Creates a DNN Keras model for classifying income data.

  Args:
    hidden_units: [int], the layer sizes of the DNN (input lay

  Returns:
    A keras Model.
  """

  # Use helper function to create the model
  model = _wide_and_deep_classifier(
      dnn_hidden_units=hidden_units or [100, 70, 50, 25])

  return model


def _wide_and_deep_classifier(dnn_hidden_units):
  """Build a simple keras wide and deep model using the Functi

  Args:
    wide_columns: Feature columns wrapped in indicator_column
      part of the model.
    deep_columns: Feature columns for deep part of the model.
    dnn_hidden_units: [int], the layer sizes of the hidden DNN

  Returns:
    A Wide and Deep Keras model
  """

  # Define input layers for numeric keys
  input_numeric = [
      tf.keras.layers.Input(name=colname, shape=(1,), dtype=tf
      for colname in _NUMERIC_FEATURE_KEYS
  ]

  # Define input layers for vocab keys
  input_categorical = [
      tf.keras.layers.Input(name=colname, shape=(vocab_size +
      for colname, vocab_size in _VOCAB_FEATURE_DICT.items()
```

```
]

# Define input layers for bucket key
input_categorical += [
    tf.keras.layers.Input(name=colname, shape=(num_buckets,)
    for colname, num_buckets in _BUCKET_FEATURE_DICT.items()
]

# Concatenate numeric inputs
deep = tf.keras.layers.concatenate(input_numeric)

# Create deep dense network for numeric inputs
for numnodes in dnn_hidden_units:
  deep = tf.keras.layers.Dense(numnodes)(deep)

# Concatenate categorical inputs
wide = tf.keras.layers.concatenate(input_categorical)

# Create shallow dense network for categorical inputs
wide = tf.keras.layers.Dense(128, activation='relu')(wide)

# Combine wide and deep networks
combined = tf.keras.layers.concatenate([deep, wide])

# Define output of binary classifier
output = tf.keras.layers.Dense(
    1, activation='sigmoid')(combined)

# Setup combined input
input_layers = input_numeric + input_categorical

# Create the Keras model
model = tf.keras.Model(input_layers, output)

# Define training parameters
model.compile(
    loss='binary_crossentropy',
    optimizer=tf.keras.optimizers.Adam(lr=0.001),
    metrics='binary_accuracy')

# Print model summary
model.summary()
```

```

  return model


# TFX Trainer will call this function.
def run_fn(fn_args: FnArgs):
  """Defines and trains the model.

  Args:
    fn_args: Holds args as name/value pairs. Refer here for th
    https://www.tensorflow.org/tfx/api_docs/python/tfx/compone
  """

  # Number of nodes in the first layer of the DNN
  first_dnn_layer_size = 100
  num_dnn_layers = 4
  dnn_decay_factor = 0.7

  # Get transform output (i.e. transform graph) wrapper
  tf_transform_output = tft.TFTransformOutput(fn_args.transfor

  # Create batches of train and eval sets
  train_dataset = _input_fn(fn_args.train_files[0], tf_transfo
  eval_dataset = _input_fn(fn_args.eval_files[0], tf_transform

  # Build the model
  model = _build_keras_model(
      # Construct layers sizes with exponential decay
      hidden_units=[
          max(2, int(first_dnn_layer_size * dnn_decay_factor**
          for i in range(num_dnn_layers)
      ])

  # Callback for TensorBoard
  tensorboard_callback = tf.keras.callbacks.TensorBoard(
      log_dir=fn_args.model_run_dir, update_freq='batch')


  # Train the model
  model.fit(
      train_dataset,
      steps_per_epoch=fn_args.train_steps,
```

```
      validation_data=eval_dataset,
      validation_steps=fn_args.eval_steps,
      callbacks=[tensorboard_callback])


  # Define default serving signature
  signatures = {
      'serving_default':
          _get_serve_tf_examples_fn(model,
                                    tf_transform_output).get_c
                                        tf.TensorSpec(
                                            shape=[None],
                                            dtype=tf.string,
                                            name='examples')),
  }
```

```
Writing census_trainer.py
```

Now, we pass in this model code to the `Trainer` component and run it to train the model.

```
trainer = tfx.components.Trainer(
    module_file=os.path.abspath(_census_trainer_module_file),
    examples=transform.outputs['transformed_examples'],
    transform_graph=transform.outputs['transform_graph'],
    schema=schema_gen.outputs['schema'],
    train_args=tfx.proto.TrainArgs(num_steps=50),
    eval_args=tfx.proto.EvalArgs(num_steps=50))
context.run(trainer, enable_cache=False)
```

```
WARNING:absl:Examples artifact does not have payload_format custom property. Fal
WARNING:absl:Examples artifact does not have payload_format custom property. Fal
WARNING:absl:Examples artifact does not have payload_format custom property. Fal
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/optimizer_v2.py:356: U
  "The `lr` argument is deprecated, use `learning_rate` instead.")
Model: "model"
________________________________________________________________________________
Layer (type)                    Output Shape          Param #     Connected to
================================================================================
fnlwgt (InputLayer)             [(None, 1)]           0

________________________________________________________________________________
education-num (InputLayer)      [(None, 1)]           0

________________________________________________________________________________
capital-gain (InputLayer)       [(None, 1)]           0

________________________________________________________________________________
capital-loss (InputLayer)       [(None, 1)]           0

________________________________________________________________________________
hours-per-week (InputLayer)     [(None, 1)]           0

________________________________________________________________________________
concatenate (Concatenate)       (None, 5)             0           fnlwgt[0][0]
                                                                  education-num[0
                                                                  capital-gain[0]
                                                                  capital-loss[0]
                                                                  hours-per-week[

________________________________________________________________________________
dense (Dense)                   (None, 100)           600         concatenate[0][

________________________________________________________________________________
dense_1 (Dense)                 (None, 70)            7070        dense[0][0]

________________________________________________________________________________
education (InputLayer)          [(None, 21)]          0

________________________________________________________________________________
marital-status (InputLayer)     [(None, 12)]          0

________________________________________________________________________________
occupation (InputLayer)         [(None, 20)]          0

________________________________________________________________________________
race (InputLayer)               [(None, 10)]          0

________________________________________________________________________________
relationship (InputLayer)       [(None, 11)]          0

________________________________________________________________________________
workclass (InputLayer)          [(None, 14)]          0

________________________________________________________________________________
sex (InputLayer)                [(None, 7)]           0

________________________________________________________________________________
native-country (InputLayer)     [(None, 47)]          0

________________________________________________________________________________
age (InputLayer)                [(None, 4)]           0

________________________________________________________________________________
dense_2 (Dense)                 (None, 48)            3408        dense_1[0][0]

________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 146)           0           education[0][0]
                                                                  marital-status[
                                                                  occupation[0][0
                                                                  race[0][0]
                                                                  relationship[0]
                                                                  workclass[0][0]
                                                                  sex[0][0]
```

```
                                                                      native-country[
                                                                      age[0][0]
______________________________________________________________________________________
dense_3 (Dense)                (None, 34)          1666          dense_2[0][0]
______________________________________________________________________________________
dense_4 (Dense)                (None, 128)         18816         concatenate_1[0
______________________________________________________________________________________
concatenate_2 (Concatenate)    (None, 162)         0             dense_3[0][0]
                                                                 dense_4[0][0]
______________________________________________________________________________________
dense_5 (Dense)                (None, 1)           163           concatenate_2[0
======================================================================================
Total params: 31,723
Trainable params: 31,723
```

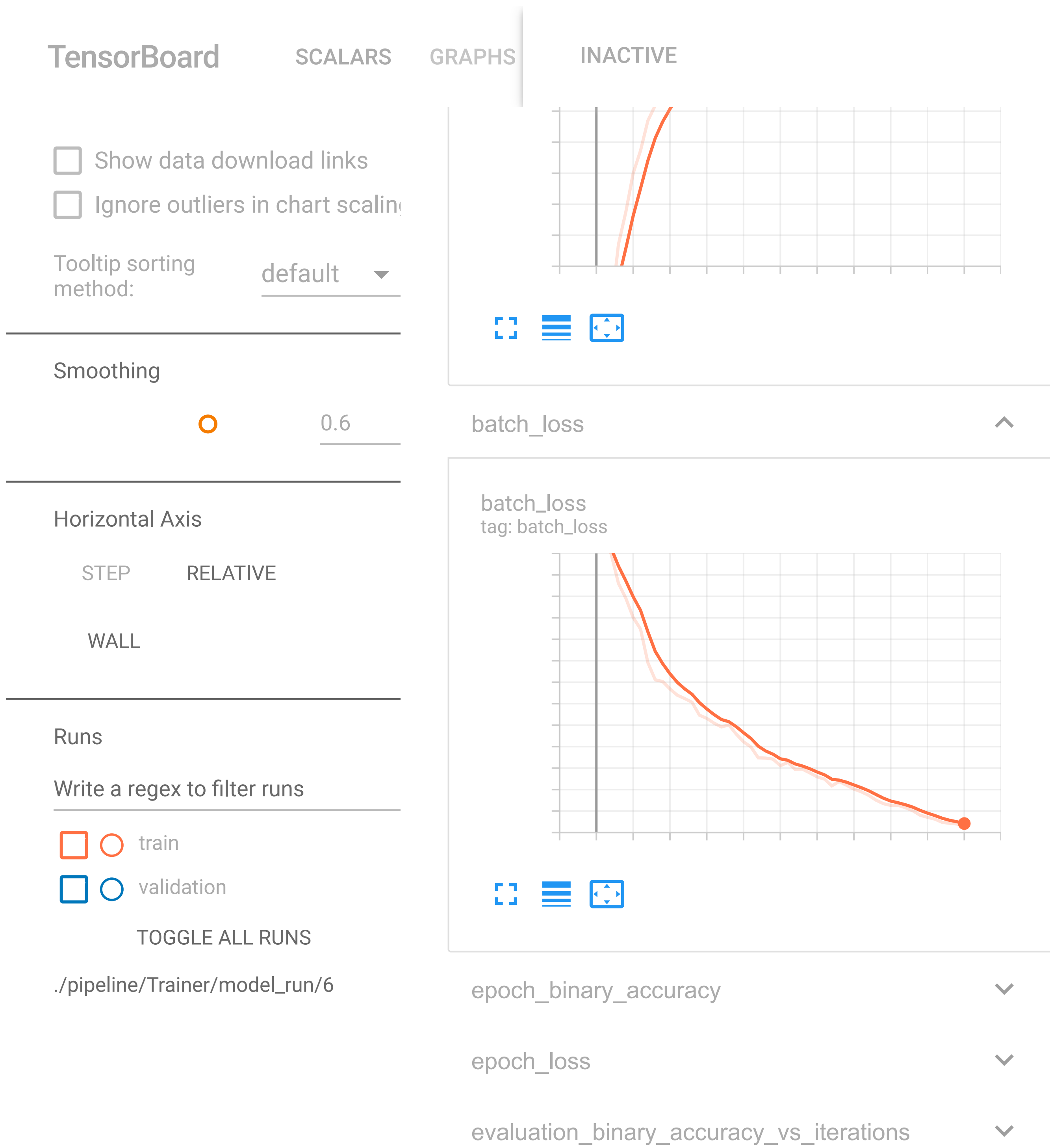
Let's review the outputs of this component. The `model` output points to the model output itself.

```
# Get `model` output of the component
model_artifact_dir = trainer.outputs['model'].get()[0].uri

# List top-level directory
pp.pprint(os.listdir(model_artifact_dir))

# Get model directory
model_dir = os.path.join(model_artifact_dir, 'Format-Serving')

# List subdirectories
pp.pprint(os.listdir(model_dir))
```

```
['Format-Serving']
['variables', 'keras_metadata.pb', 'saved_model.pb', 'assets']
```

The `model_run` output acts as the working directory and can be used to output non-model related output (e.g., TensorBoard logs).

```
# Get `model_run` output URI
model_run_artifact_dir = trainer.outputs['model_run'].get()[0]

# Load results to Tensorboard
%load_ext tensorboard
%tensorboard --logdir {model_run_artifact_dir}
```



Evaluator

The `Evaluator` component computes model performance metrics over the evaluation set using the [TensorFlow Model Analysis](#) library. The `Evaluator` can also optionally validate that a newly trained model is better than the previous model. This is useful in a production pipeline setting where you may automatically train and validate a model every day.

There's a few steps needed to setup this component and you will do it in the next cells.

Define EvalConfig

First, you will define the `EvalConfig` message as you did in the previous lab. You can also set thresholds so you can compare subsequent models to it. The module below should look familiar. One minor difference is you don't have to define the candidate and baseline model names in the `model_specs`. That is automatically detected.

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

eval_config = text_format.Parse("""
  ## Model information
  model_specs {
    # This assumes a serving model with signature 'serving_def
    signature_name: "serving_default",
    label_key: "label"
  }

  ## Post training metric information
  metrics_specs {
    metrics { class_name: "ExampleCount" }
    metrics {
      class_name: "BinaryAccuracy"
      threshold {
        # Ensure that metric is always > 0.5
        value_threshold {
          lower_bound { value: 0.5 }
        }
        # Ensure that metric does not drop by more than a smal
        # e.g. (candidate - baseline) > -1e-10 or candidate >
        change_threshold {
          direction: HIGHER_IS_BETTER
          absolute { value: -1e-10 }
        }
      }
    }
    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
  slicing_specs {}  # overall slice
  slicing_specs {
    feature_keys: ["race"]
  }
  slicing_specs {
    feature_keys: ["sex"]
  }
""", tfma.EvalConfig())
```

Resolve latest blessed model

If you remember in the last lab, you were able to validate a candidate model against a baseline by modifying the `EvalConfig` and `EvalSharedModel` definitions. That is also possible using the `Evaluator` component and you will see how it is done in this section.

First thing to note is that the `Evaluator` marks a model as `BLESSED` if it meets the metrics thresholds you set in the eval config module. You can load the latest blessed model by using the [`LatestBlessedModelStrategy`]() with the [Resolver]() component. This component takes care of finding the latest blessed model for you so you don't have to remember it manually. The syntax is shown below.

```
# Setup the Resolver node to find the latest blessed model
model_resolver = tfx.dsl.Resolver(
      strategy_class=tfx.dsl.experimental.LatestBlessedModelSt
      model=tfx.dsl.Channel(type=tfx.types.standard_artifacts.
      model_blessing=tfx.dsl.Channel(
          type=tfx.types.standard_artifacts.ModelBlessing)).wi
              'latest_blessed_model_resolver')

# Run the Resolver node
context.run(model_resolver)
```

▼ExecutionResult at 0x7f84db2368d0

.execution_id	7	
.component	<tfx.dsl.components.common.resolver.Resolver object at 0x7f84d869f910>	
.component.inputs	**['model']**	**▼Channel** of type **'Model'** (0 artifacts) at 0x7f84dc425e50 **.type_name** Model **._artifacts** []
	['model_blessing']	**▼Channel** of type **'ModelBlessing'** (0 artifacts) at 0x7f84d9ea7f90 **.type_name** ModelBlessing **._artifacts** []
.component.outputs	**['model']**	**▼Channel** of type **'Model'** (0 artifacts) at 0x7f84daec1490 **.type_name** Model **._artifacts** []

As expected, the search yielded 0 artifacts because you haven't evaluated any models yet. You will run this component again in later parts of this notebook and you will see a different result.

```
# Load Resolver outputs
model_resolver.outputs['model']
```

▼Channel of type **'Model'** (0 artifacts) at 0x7f84daec1490

.type_name Model
._artifacts []

Run the Evaluator component

With the `EvalConfig` defined and code to load the latest blessed model available, you can proceed to run the `Evaluator` component.

You will notice that two models are passed into the component. The `Trainer` output will serve as the candidate model while the output of the `Resolver` will be the baseline model. While you can indeed run the `Evaluator` without comparing two models, it would likely be required in production environments so it's best to include it. Since the `Resolver` doesn't have any results yet, the `Evaluator` will just mark the candidate model as `BLESSED` in this run.

Aside from the eval config and models (i.e. Trainer and Resolver output), you will also pass in the *raw* examples from `ExampleGen`. By default, the component will look for the `eval` split of these examples and since you've defined the serving signature, these will be transformed internally before feeding to the model inputs.

```
# Setup and run the Evaluator component
evaluator = tfx.components.Evaluator(
    examples=example_gen.outputs['examples'],
    model=trainer.outputs['model'],
    baseline_model=model_resolver.outputs['model'],
    eval_config=eval_config)
context.run(evaluator, enable_cache=False)
```

```
WARNING:root:Make sure that locally built Python SDK docker image has Python 3.7
```

▼ExecutionResult at 0x7f84e1348390

.execution_id	8
.component	**▼Evaluator** at 0x7f84e06aa710

.inputs

['examples'] **▼Channel** of type **'Examples'** (1 ar

.type_name Examples

._artifacts **[0]** **▼Artifact** of typ ./pipeline/CsvEx 0x7f85467ff210

.type	< 't
.uri	./
.span	0
.split_names	['
.version	0

['model'] **▼Channel** of type **'Model'** (1 artifac

.type_name Model

._artifacts **[0]** **▼Artifact** of typ ./pipeline/Traine

.type	<class 'tfx
.uri	./pipeline/

['baseline_model'] **▼Channel** of type **'Model'** (0 artifac

.type_name Model

._artifacts []

.outputs

['evaluation'] **▼Channel** of type **'ModelEvaluation'** (1

.type_name ModelEvaluation

._artifacts **[0]** **▼Artifact** of type **'Mo** ./pipeline/Evaluator/ev

.type	<class 'tfx.types
.uri	./pipeline/Evalu

['blessing'] **▼Channel** of type **'ModelBlessing'** (1 ar

.type_name ModelBlessing

._artifacts **[0]** **▼Artifact** of type **'Mo** ./pipeline/Evaluator/bl

.type <class 'tfx.types
.uri ./pipeline/Evalu

.exec_properties ['eval_config'] { "metrics_specs": [{ "
"ExampleCount" }, { "
"threshold": { "change
"direction": "HIGHER_
"lower_bound": 0.5 } }
{ "class_name": "AUC
}, { "class_name": "Pr
"class_name": "Mean
}, { "class_name": "Ca
"CalibrationPlot" }, { "
"model_specs": [{ "la
"serving_default" }], "

Now let's examine the output artifacts of `Evaluator`.

```
# Print component output keys
evaluator.outputs.keys()
```

```
dict_keys(['evaluation', 'blessing'])
```

The `blessing` output simply states if the candidate model was blessed. The artifact URI will have a `BLESSED` or `NOT_BLESSED` file depending on the result. As mentioned earlier, this first run will pass the evaluation because there is no baseline model yet.

```
# Get `Evaluator` blessing output URI
blessing_uri = evaluator.outputs['blessing'].get()[0].uri

# List files under URI
os.listdir(blessing_uri)
```

```
['BLESSED']
```

The `evaluation` output, on the other hand, contains the evaluation logs and can be used to visualize the global metrics on the entire evaluation set.

```
# Visualize the evaluation results
context.show(evaluator.outputs['evaluation'])
```

Artifact at ./pipeline/Evaluator/evaluation/8

Examples (Weighted) Threshold

0

Visualization

Slices Overview

Show

auc_precision_recall

Sort by

Slice

feature	auc	auc_precision_recall	binary_accu
Overall	0.86546	0.66025	0.80

To see the individual slices, you will need to import TFMA and use the commands you learned in the previous lab.

```
import tensorflow_model_analysis as tfma

# Get the TFMA output result path and load the result.
PATH_TO_RESULT = evaluator.outputs['evaluation'].get()[0].uri
tfma_result = tfma.load_eval_result(PATH_TO_RESULT)

# Show data sliced along feature column trip_start_hour.
tfma.view.render_slicing_metrics(
    tfma_result, slicing_column='sex')
```

Examples (Weighted) Threshold

0

Visualization

Slices Overview

Show

example_count

Sort by

Slice

feature	auc	auc_precision_recall	binary_
sex:Male	0.84052	0.68632	
sex:Female	0.89632	0.56684	

You can also use TFMA to load the validation results as before by specifying the output URI of the evaluation output. This would be more useful if your model was not blessed because you can see the metric failure prompts. Try to simulate this later by training with fewer epochs (or raising the threshold) and see the results you get here.

```
# Get `evaluation` output URI
PATH_TO_RESULT = evaluator.outputs['evaluation'].get()[0].uri

# Print validation result
print(tfma.load_validation_result(PATH_TO_RESULT))
```

```
validation_ok: true
```

```
    validation_details {
      slicing_details {
        slicing_spec {
        }
        num_matching_slices: 8
      }
    }
```

Now that your `Evaluator` has finished running, the `Resolver` component should be able to detect the latest blessed model. Let's run the component again.

```
# Re-run the Resolver component
context.run(model_resolver)
```

▼**ExecutionResult** at 0x7f84d99fbe10

.execution_id	9	
.component	<tfx.dsl.components.common.resolver.Resolver object at 0x7f84d869f910>	
.component.inputs	**['model']**	▼**Channel** of type **'Model'** (0 artifacts) at 0x7f84dc425e50 **.type_name** Model **._artifacts** []
	['model_blessing']	▼**Channel** of type **'ModelBlessing'** (0 artifacts) at 0x7f84d9ea7f90 **.type_name** ModelBlessing **._artifacts** []
.component.outputs	**['model']**	▼**Channel** of type **'Model'** (1 artifact) at 0x7f84d9ce0d10 **.type_name** Model **._artifacts** **[0]** ▶**Artifact** of type **'Model'** (uri: ./pipeline/Trainer/model/6) at 0x7f84d9ce0590
	['model_blessing']	▼**Channel** of type **'ModelBlessing'** (1 artifact) at 0x7f84d9ce0850

You should now see an artifact in the component outputs. You can also get it programmatically as shown below.

```
# Get path to latest blessed model
```

```
model_resolver.outputs['model'].get()[0].uri
```

```
'./pipeline/Trainer/model/6'
```

Comparing two models

Now let's see how `Evaluator` compares two models. You will train the same model with more epochs and this should hopefully result in higher accuracy and overall metrics.

```
# Setup trainer to train with more epochs
trainer = tfx.components.Trainer(
    module_file=os.path.abspath(_census_trainer_module_file),
    examples=transform.outputs['transformed_examples'],
    transform_graph=transform.outputs['transform_graph'],
    schema=schema_gen.outputs['schema'],
    train_args=tfx.proto.TrainArgs(num_steps=500),
    eval_args=tfx.proto.EvalArgs(num_steps=200))

# Run trainer
context.run(trainer, enable_cache=False)
```

```
WARNING:absl:Examples artifact does not have payload_format custom property. Fal
WARNING:absl:Examples artifact does not have payload_format custom property. Fal
WARNING:absl:Examples artifact does not have payload_format custom property. Fal
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/optimizer_v2.py:356: U
  "The `lr` argument is deprecated, use `learning_rate` instead.")
Model: "model_1"
________________________________________________________________________________
Layer (type)                    Output Shape          Param #     Connected to
================================================================================
fnlwgt (InputLayer)             [(None, 1)]           0
________________________________________________________________________________
education-num (InputLayer)      [(None, 1)]           0
________________________________________________________________________________
capital-gain (InputLayer)       [(None, 1)]           0
________________________________________________________________________________
capital-loss (InputLayer)       [(None, 1)]           0
________________________________________________________________________________
hours-per-week (InputLayer)     [(None, 1)]           0
________________________________________________________________________________
concatenate_3 (Concatenate)     (None, 5)             0           fnlwgt[0][0]
                                                                  education-num[0
                                                                  capital-gain[0]
                                                                  capital-loss[0]
                                                                  hours-per-week[
________________________________________________________________________________
dense_6 (Dense)                 (None, 100)           600         concatenate_3[0
________________________________________________________________________________
dense_7 (Dense)                 (None, 70)            7070        dense_6[0][0]
________________________________________________________________________________
education (InputLayer)          [(None, 21)]          0
________________________________________________________________________________
marital-status (InputLayer)     [(None, 12)]          0
________________________________________________________________________________
occupation (InputLayer)         [(None, 20)]          0
________________________________________________________________________________
race (InputLayer)               [(None, 10)]          0
________________________________________________________________________________
relationship (InputLayer)       [(None, 11)]          0
________________________________________________________________________________
workclass (InputLayer)          [(None, 14)]          0
________________________________________________________________________________
sex (InputLayer)                [(None, 7)]           0
________________________________________________________________________________
native-country (InputLayer)     [(None, 47)]          0
________________________________________________________________________________
age (InputLayer)                [(None, 4)]           0
________________________________________________________________________________
dense_8 (Dense)                 (None, 48)            3408        dense_7[0][0]
________________________________________________________________________________
concatenate_4 (Concatenate)     (None, 146)           0           education[0][0]
                                                                  marital-status[
                                                                  occupation[0][0
                                                                  race[0][0]
                                                                  relationship[0]
                                                                  workclass[0][0]
                                                                  sex[0][0]
```

```
                                                                  native-country[
                                                                  age[0][0]

dense_9 (Dense)                    (None, 34)          1666       dense_8[0][0]

dense_10 (Dense)                   (None, 128)         18816      concatenate_4[0

concatenate_5 (Concatenate)        (None, 162)         0          dense_9[0][0]
                                                                  dense_10[0][0]

dense_11 (Dense)                   (None, 1)           163        concatenate_5[0
===================================================================================
Total params: 31,723
Trainable params: 31,723
Non-trainable params: 0

500/500 [==============================] - 4s 6ms/step - loss: 0.3545 - binary_a
```

▼ExecutionResult at 0x7f84d8895e90

.execution_id 10

.component **▼Trainer** at 0x7f84d9cdfd90

.inputs

['examples'] **▼Channel** of type **'Examples'** (1 a

.type_name Examples

._artifacts **[0]** **▶Artifact** of typ

./pipeline/Transfo

0x7f84e132ced0

['transform_graph'] **▼Channel** of type **'TransformGra**

.type_name TransformGraph

._artifacts **[0]** **▶Artifact** of typ

You will re-run the evaluator but you will specify the latest trainer output as the candidate model. The baseline is automatically found with the Resolver node.

```
# Setup and run the Evaluator component
evaluator = tfx.components.Evaluator(
    examples=example_gen.outputs['examples'],
    model=trainer.outputs['model'],
    baseline_model=model_resolver.outputs['model'],
    eval_config=eval_config)
context.run(evaluator, enable_cache=False)
```

```
WARNING:root:Make sure that locally built Python SDK docker image has Python 3.7
```

▼**ExecutionResult** at 0x7f84defe2810

.execution_id	11		
.component	▶**Evaluator** at 0x7f84d22b86d0		
.component.inputs	**['examples']**	▼**Channel** of type **'Examples'** (1 artifact) at 0x7f84e3471e10	
		.type_name Examples	
		._artifacts	**[0]** ▶**Artifact** of type **'Examples'** (uri: ./pipeline/CsvExampleGen/examples/1 at 0x7f85467ff210
	['model']	▼**Channel** of type **'Model'** (1 artifact) at 0x7f84d9cdff90	
		.type_name Model	
		._artifacts	**[0]** ▶**Artifact** of type **'Model'** (uri: ./pipeline/Trainer/model/10) at 0x7f84d9ce0dd0
	['baseline_model']	▼**Channel** of type **'Model'** (1 artifact) at 0x7f84d9ce0d1	

Depending on the result, the Resolver should reflect the latest blessed model. Since you trained with more epochs, it is most likely that your candidate model will pass the thresholds you set in the eval config. If so, the artifact URI should be different here compared to your earlier runs.

```
# Re-run the resolver
context.run(model_resolver, enable_cache=False)
```

▼**ExecutionResult** at 0x7f84d58d6e50

.execution_id 12
.component <tfx.dsl.components.common.resolver.Resolver object at 0x7f84d869f910>
.component.inputs **['model']** ▼**Channel** of type **'Model'** (0 artifacts) at 0x7f84dc425e50
.type_name Model

```
# Get path to latest blessed model
model_resolver.outputs['model'].get()[0].uri
```

```
'./pipeline/Trainer/model/10'
```

.artifacts []

Finally, the `evaluation` output of the `Evaluator` component will now be able to produce the `diff` results you saw in the previous lab. This will signify if the metrics from the candidate model has indeed improved compared to the baseline. Unlike when using TFMA as a standalone library, visualizing this will just show the results for the candidate (i.e. only one row instead of two in the tabular output in the graph below).

Note: You can ignore the warning about failing to find plots.

```
context.show(evaluator.outputs['evaluation'])
```

Artifact at ./pipeline/Evaluator/evaluation/11

```
WARNING:absl:Fail to find plots for model name: None . Available model names are
WARNING:absl:Fail to find plots for model name: None . Available model names are
WARNING:absl:Fail to find plots for model name: None . Available model names are
WARNING:absl:Fail to find plots for model name: None . Available model names are
WARNING:absl:Fail to find plots for model name: None . Available model names are
WARNING:absl:Fail to find plots for model name: None . Available model names are
WARNING:absl:Fail to find plots for model name: None . Available model names are
WARNING:absl:Fail to find plots for model name: None . Available model names are
```

Examples (Weighted) Threshold

0

Visualization

Slices Overview

Show

Sort by

Congratulations! You can now successfully evaluate your models in a TFX pipeline! This is a critical part of production ML because you want to make sure that subsequent deployments are indeed improving your results. Moreover, you can extract the evaluation results from your pipeline directory for further investigation with TFMA. In the next sections, you will continue to study techniques related to model evaluation and ensuring fairness.

feature	auc	auc_diff	auc_precision_recall
Overall	0.91107	0.04560	0.76560