Lab: TfTransform

Learning Objectives

1. Preprocess data and engineer new features using TfTransform
2. Create and deploy Apache Beam pipeline
3. Use processed data to train taxifare model locally then serve a prediction

Introduction

While Pandas is fine for experimenting, for operationalization of your workflow it is better to do preprocessing in Apache Beam. This will also help if you need to preprocess data in flight, since Apache Beam allows for streaming. In this lab we will pull data from BigQuery then use Apache Beam TfTransform to process the data.

Only specific combinations of TensorFlow/Beam are supported by tf.transform so make sure to get a combo that works. In this lab we will be using:

- TFT 0.15.0
- TF 2.0
- Apache Beam [GCP] 2.16.0

NOTE: In the output of the next pip commands, you may ignore any WARNINGS or ERRORS related to the incompatibility of the following: "witwidget-gpu", "fairing", "pbr, "hdfscli", "hdfscli-avro", "fastavro", "plasma_store", and/or "gen_client".

In [ ]:

```
!sudo chown -R jupyter:jupyter /home/jupyter/training-data-analyst
```

In [ ]:

```
!pip install tensorflow==2.1.0
```

In [ ]:

```
!pip install --user apache-beam[gcp]==2.16.0
!pip install --user tensorflow-transform==0.15.0
```

Download .whl file for tensorflow-transform. We will pass this file to Beam Pipeline Options so it is installed on the DataFlow workers

In [ ]:

```
!pip download tensorflow-transform==0.15.0 --no-deps
```

Restart the kernel (In the above menu, click Kernel > Restart kernel > Restart)

In [ ]:

```
# Ensure the right version of Tensorflow is installed.
!pip freeze | grep tensorflow==2.1
```

In [ ]:

```
%%bash
pip freeze | grep -e 'flow\|beam'
```

In [ ]:

```
import tensorflow as tf
import tensorflow_transform as tft
import shutil
print(tf.__version__)
```

In [15]:

```
# change these to those of your environment to try this notebook out

BUCKET = 'cloud-training-demos-ml'
PROJECT = 'cloud-training-demos'
REGION = 'us-central1'
```

In [ ]:

```
import os
os.environ['BUCKET'] = BUCKET
os.environ['PROJECT'] = PROJECT
os.environ['REGION'] = REGION
```

In [ ]:

```
%%bash
gcloud config set project $PROJECT
gcloud config set compute/region $REGION
```

In [ ]:

```
%%bash
if ! gsutil ls | grep -q gs://${BUCKET}/; then
  gsutil mb -l ${REGION} gs://${BUCKET}
fi
```

Input source: BigQuery

Get data from BigQuery but defer the majority of filtering etc. to Beam. Note that the dayofweek column is now strings.

In [ ]:

```
from google.cloud import bigquery


def create_query(phase, EVERY_N):
    """Creates a query with the proper splits.

    Args:
        phase: int, 1=train, 2=valid.
        EVERY_N: int, take an example EVERY_N rows.

    Returns:
        Query string with the proper splits.
    """
    base_query = """
    WITH daynames AS
    (SELECT ['Sun', 'Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'] AS daysofweek)
    SELECT
    (tolls_amount + fare_amount) AS fare_amount,
    daysofweek[ORDINAL(EXTRACT(DAYOFWEEK FROM pickup_datetime))] AS dayofweek,
    EXTRACT(HOUR FROM pickup_datetime) AS hourofday,
    pickup_longitude AS pickuplon,
    pickup_latitude AS pickuplat,
    dropoff_longitude AS dropofflon,
    dropoff_latitude AS dropofflat,
    passenger_count AS passengers,
    'notneeded' AS key
    FROM
    `nyc-tlc.yellow.trips`, daynames
    WHERE
    trip_distance > 0 AND fare_amount > 0
    """
    if EVERY_N is None:
        if phase < 2:
            # training
            query = """{0} AND ABS(MOD(FARM_FINGERPRINT(CAST
            (pickup_datetime AS STRING), 4)) < 2""".format(base_query)
        else:
            query = """{0} AND ABS(MOD(FARM_FINGERPRINT(CAST(
            pickup_datetime AS STRING), 4)) = {1}""".format(base_query, phase)
    else:
        query = """{0} AND ABS(MOD(FARM_FINGERPRINT(CAST(
        pickup_datetime AS STRING)), {1})) = {2}""".format(
            base_query, EVERY_N, phase)

    return query

query = create_query(2, 100000)
```

Let's pull this query down into a Pandas DataFrame and take a look at some of the statistics.

In [ ]:

```
df_valid = bigquery.Client().query(query).to_dataframe()
display(df_valid.head())
df_valid.describe()
```

In [ ]:

```
OUTDIR = './trained_model'
shutil.rmtree(OUTDIR, ignore_errors = True)
tf.compat.v1.summary.FileWriterCache.clear()
```

Create ML dataset using tf.transform and Dataflow

Let's use Cloud Dataflow to read in the BigQuery data and write it out as TFRecord files. Along the way, let's use tf.transform to do scaling and transforming. Using tf.transform allows us to save the metadata to ensure that the appropriate transformations get carried out during prediction as well.

NOTE: You may ignore any WARNING related to "tensorflow" in the output after executing the code cell below.

`transformed_data` is type `pcollection` .

In [ ]:

```
import datetime
import tensorflow as tf
import apache_beam as beam
import tensorflow_transform as tft
import tensorflow_metadata as tfmd
from tensorflow_transform.beam import impl as beam_impl


def is_valid(inputs):
    """Check to make sure the inputs are valid.

    Args:
        inputs: dict, dictionary of TableRow data from BigQuery.

    Returns:
        True if the inputs are valid and False if they are not.
    """
    try:
        pickup_longitude = inputs['pickuplon']
        dropoff_longitude = inputs['dropofflon']
        pickup_latitude = inputs['pickuplat']
        dropoff_latitude = inputs['dropofflat']
        hourofday = inputs['hourofday']
        dayofweek = inputs['dayofweek']
        passenger_count = inputs['passengers']
        fare_amount = inputs['fare_amount']
        return fare_amount >= 2.5 and pickup_longitude > -78 \
            and pickup_longitude < -70 and dropoff_longitude > -78 \
            and dropoff_longitude < -70 and pickup_latitude > 37 \
            and pickup_latitude < 45 and dropoff_latitude > 37 \
            and dropoff_latitude < 45 and passenger_count > 0
    except:
        return False


def preprocess_tft(inputs):
    """Preprocess the features and add engineered features with tf transform.

    Args:
        dict, dictionary of TableRow data from BigQuery.

    Returns:
        Dictionary of preprocessed data after scaling and feature engineering.
    """
    import datetime
    print(inputs)
    result = {}
    result['fare_amount'] = tf.identity(inputs['fare_amount'])
    # build a vocabulary
    result['dayofweek'] = tft.string_to_int(inputs['dayofweek'])
    result['hourofday'] = tf.identity(inputs['hourofday'])  # pass through
    # scaling numeric values
    result['pickuplon'] = (tft.scale_to_0_1(inputs['pickuplon']))
    result['pickuplat'] = (tft.scale_to_0_1(inputs['pickuplat']))
    result['dropofflon'] = (tft.scale_to_0_1(inputs['dropofflon']))
    result['dropofflat'] = (tft.scale_to_0_1(inputs['dropofflat']))
    result['passengers'] = tf.cast(inputs['passengers'], tf.float32)  # a cast
    # arbitrary TF func
    result['key'] = tf.as_string(tf.ones_like(inputs['passengers']))
    # engineered features
    latdiff = inputs['pickuplat'] - inputs['dropofflat']
    londiff = inputs['pickuplon'] - inputs['dropofflon']
    result['latdiff'] = tft.scale_to_0_1(latdiff)
    result['londiff'] = tft.scale_to_0_1(londiff)
    dist = tf.sqrt(latdiff * latdiff + londiff * londiff)
    result['euclidean'] = tft.scale_to_0_1(dist)
    return result


def preprocess(in_test_mode):
    """Sets up preprocess pipeline.

    Args:
        in_test_mode: bool, False to launch DataFlow job, True to run locally.
    """
    import os
    import os.path
    import tempfile
    from apache_beam.io import tfrecordio
    from tensorflow_transform.coders import example_proto_coder
    from tensorflow_transform.tf_metadata import dataset_metadata
    from tensorflow_transform.tf_metadata import dataset_schema
    from tensorflow_transform.beam import tft_beam_io
    from tensorflow_transform.beam.tft_beam_io import transform_fn_io

    job_name = 'preprocess-taxi-features' + '-'
    job_name += datetime.datetime.now().strftime('%y%m%d-%H%M%S')
    if in_test_mode:
        import shutil
        print('Launching local job ... hang on')
        OUTPUT_DIR = './preproc_tft'
        shutil.rmtree(OUTPUT_DIR, ignore_errors=True)
        EVERY_N = 100000
    else:
        print('Launching Dataflow job {} ... hang on'.format(job_name))
        OUTPUT_DIR = 'gs://{0}/taxifare/preproc_tft/'.format(BUCKET)
        import subprocess
        subprocess.call('gsutil rm -r {}'.format(OUTPUT_DIR).split())
        EVERY_N = 10000

    options = {
        'staging_location': os.path.join(OUTPUT_DIR, 'tmp', 'staging'),
        'temp_location': os.path.join(OUTPUT_DIR, 'tmp'),
        'job_name': job_name,
        'project': PROJECT,
        'num_workers': 1,
        'max_num_workers': 1,
        'teardown_policy': 'TEARDOWN_ALWAYS',
        'no_save_main_session': True,
        'direct_num_workers': 1,
        'extra_packages': ['tensorflow-transform-0.15.0.tar.gz']
        }
    opts = beam.pipeline.PipelineOptions(flags=[], **options)
    if in_test_mode:
        RUNNER = 'DirectRunner'
    else:
        RUNNER = 'DataflowRunner'

    # Set up raw data metadata
    raw_data_schema = {
        colname: dataset_schema.ColumnSchema(
            tf.string, [], dataset_schema.FixedColumnRepresentation())
        for colname in 'dayofweek,key'.split(',')
    }

    raw_data_schema.update({
        colname: dataset_schema.ColumnSchema(
            tf.float32, [], dataset_schema.FixedColumnRepresentation())
        for colname in
        'fare_amount,pickuplon,pickuplat,dropofflon,dropofflat'.split(',')
    })

    raw_data_schema.update({
        colname: dataset_schema.ColumnSchema(
            tf.int64, [], dataset_schema.FixedColumnRepresentation())
        for colname in 'hourofday,passengers'.split(',')
    })

    raw_data_metadata = dataset_metadata.DatasetMetadata(
        dataset_schema.Schema(raw_data_schema))

    # Run Beam
    with beam.Pipeline(RUNNER, options=opts) as p:
        with beam_impl.Context(temp_dir=os.path.join(OUTPUT_DIR, 'tmp')):
            # Save the raw data metadata
            (raw_data_metadata |
                'WriteInputMetadata' >> tft_beam_io.WriteMetadata(
                    os.path.join(
                        OUTPUT_DIR, 'metadata/rawdata_metadata'), pipeline=p))

            # Read training data from bigquery and filter rows
            raw_data = (p | 'train_read' >> beam.io.Read(
                    beam.io.BigQuerySource(
                        query=create_query(1, EVERY_N),
                        use_standard_sql=True)) |
                        'train_filter' >> beam.Filter(is_valid))

            raw_dataset = (raw_data, raw_data_metadata)

            # Analyze and transform training data
            transformed_dataset, transform_fn = (
                raw_dataset | beam_impl.AnalyzeAndTransformDataset(
                    preprocess_tft))
            transformed_data, transformed_metadata = transformed_dataset

            # Save transformed train data to disk in efficient tfrecord format
            transformed_data | 'WriteTrainData' >> tfrecordio.WriteToTFRecord(
                os.path.join(OUTPUT_DIR, 'train'), file_name_suffix='.gz',
                coder=example_proto_coder.ExampleProtoCoder(
                    transformed_metadata.schema))

            # Read eval data from bigquery and filter rows
            raw_test_data = (p | 'eval_read' >> beam.io.Read(
                beam.io.BigQuerySource(
                    query=create_query(2, EVERY_N),
                    use_standard_sql=True)) | 'eval_filter' >> beam.Filter(
                        is_valid))

            raw_test_dataset = (raw_test_data, raw_data_metadata)

            # Transform eval data
            transformed_test_dataset = (
                (raw_test_dataset, transform_fn) | beam_impl.TransformDataset()
                )
            transformed_test_data, _ = transformed_test_dataset

            # Save transformed train data to disk in efficient tfrecord format
            (transformed_test_data |
                'WriteTestData' >> tfrecordio.WriteToTFRecord(
                    os.path.join(OUTPUT_DIR, 'eval'), file_name_suffix='.gz',
                    coder=example_proto_coder.ExampleProtoCoder(
                        transformed_metadata.schema)))

            # Save transformation function to disk for use at serving time
            (transform_fn |
                'WriteTransformFn' >> transform_fn_io.WriteTransformFn(
                    os.path.join(OUTPUT_DIR, 'metadata')))

# Change to True to run locally
preprocess(in_test_mode=False)
```

This will take **10-15 minutes**. You cannot go on in this lab until your DataFlow job has successfully completed.

You may monitor the progress of the Dataflow job in the GCP console on the **Navigation menu > Dataflow** page.

When you see the Jupyter notebook status has returned to "Idle" you may proceed to the next step.

In [ ]:

```
%%bash
# ls preproc_tft
gsutil ls gs://${BUCKET}/taxifare/preproc_tft/
```

Train off preprocessed data

Now that we have our data ready and verified it is in the correct location we can train our taxifare model locally.

NOTE: You may ignore any WARNING related to "tensorflow" in any of the outputs that follow from this point.

In [ ]:

```
%%bash
rm -r ./taxi_trained
export PYTHONPATH=${PYTHONPATH}:$PWD
python3 -m tft_trainer.task \
    --train_data_path="gs://${BUCKET}/taxifare/preproc_tft/train*" \
    --eval_data_path="gs://${BUCKET}/taxifare/preproc_tft/eval*"  \
    --output_dir=./taxi_trained \
```

NOTE: If you get any directory not found error then you may need to rerun the above cell.

In [ ]:

```
!ls $PWD/taxi_trained/export/exporter
```

Now let's create fake data in JSON format and use it to serve a prediction with gcloud ai-platform local predict

In [ ]:

```
%%writefile /tmp/test.json
{"dayofweek":0, "hourofday":17, "pickuplon": -73.885262, "pickuplat": 40.773008, "dropofflon": -73.987232, "dropofflat": 40.732403, "passengers": 2.0}
```

In [ ]:

```
%%bash
sudo find "/usr/lib/google-cloud-sdk/lib/googlecloudsdk/command_lib/ml_engine" -name '*.pyc' -delete
```

In [ ]:

```
%%bash
model_dir=$(ls $PWD/taxi_trained/export/exporter/)
gcloud ai-platform local predict \
    --model-dir=./taxi_trained/export/exporter/${model_dir} \
    --json-instances=/tmp/test.json
```