Feature Engineering

In this notebook, you will learn how to incorporate feature engineering into your pipeline.

- Working with feature columns
- Adding feature crosses in TensorFlow
- Reading data from BigQuery
- Creating datasets using Dataflow
- Using a wide-and-deep model

Feature Engineering

In this notebook, you will learn how to incorporate feature engineering into your pipeline.

- Working with feature columns
- Adding feature crosses in TensorFlow
- Reading data from BigQuery
- Creating datasets using Dataflow
- Using a wide-and-deep model

```
import tensorflow as tf
import apache_beam as beam
import shutil
print(tf.__version__)
```

```
/usr/local/envs/py2env/lib/python2.7/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `n
p.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters

1.8.0
```

1. Environment variables for project and bucket

- Your project id is the *unique* string that identifies your project (not the project name). You can find this from the GCP Console dashboard's Home page. My dashboard reads: **Project ID:** cloud-training-demos

- Cloud training often involves saving and restoring model files. Therefore, we should **create a single-region bucket**. If you don't have a bucket already, I suggest that you create one from the GCP console (because it will dynamically check whether the bucket name you want is available)
 Change the cell below to reflect your Project ID and bucket name.

```
import os
REGION = 'us-central1' # Choose an available region for Cloud MLE from https://cloud.google.com/ml-engine/docs/regions.
BUCKET = 'cloud-training-demos-ml' # REPLACE WITH YOUR BUCKET NAME. Use a regional bucket in the region you selected.
PROJECT = 'cloud-training-demos'    # CHANGE THIS
```

```
# For Python Code
# Model Info
MODEL_NAME = 'taxifare_feateng'  # this name should match entry in trainer/setup.py
# Model Version
MODEL_VERSION = 'v2'
# Training Directory name
TRAINING_DIR = 'taxifare_feateng_trained'
# Preprocessing Directory name
PREPROC_DIR = 'taxifare_feateng_preproc'
```

```
# For Bash Code
os.environ['PROJECT'] = PROJECT
os.environ['BUCKET'] = BUCKET
os.environ['REGION'] = REGION
os.environ['MODEL_NAME'] = MODEL_NAME
os.environ['MODEL_VERSION'] = MODEL_VERSION
os.environ['TRAINING_DIR'] = TRAINING_DIR
os.environ['PREPROC_DIR'] = PREPROC_DIR
os.environ['TFVERSION'] = '1.8'  # Tensorflow version
```

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

```
Updated property [core/project].
Updated property [compute/region].
```

Create the bucket to store model and training data for deploying to Google Cloud Machine Learning Engine Component

```
%%bash
# The bucket needs to exist for the gsutil commands in next cell to work
gsutil mb -p ${PROJECT} gs://${BUCKET}
```

```
Creating gs://bucket-netskink-tda/...
ServiceException: 409 Bucket bucket-netskink-tda already exists.
```

Enable the Cloud Machine Learning Engine API

```
%%bash
# This command will fail if the Cloud Machine Learning Engine API is not enabled using the link above.
echo "Getting the service account email associated with the Cloud Machine Learning Engine API"

AUTH_TOKEN=$(gcloud auth print-access-token)
SVC_ACCOUNT=$(curl -X GET -H "Content-Type: application/json" \
    -H "Authorization: Bearer $AUTH_TOKEN" \
    https://ml.googleapis.com/v1/projects/${PROJECT}:getConfig \
    | python3 -c "import json; import sys; response = json.load(sys.stdin); \
    print (response['serviceAccount'])") # If this command fails, the Cloud Machine Learning Engine API has not been enabled above.

echo "Authorizing the Cloud ML Service account $SVC_ACCOUNT to access files in $BUCKET"
gsutil -m defacl ch -u $SVC_ACCOUNT:R gs://$BUCKET
gsutil -m acl ch -u $SVC_ACCOUNT:R -r gs://$BUCKET  # error message (if bucket is empty) can be ignored.
gsutil -m acl ch -u $SVC_ACCOUNT:W gs://$BUCKET
```

```
Getting the service account email associated with the Cloud Machine Learning Engine API
Authorizing the Cloud ML Service account service-35462102002@cloud-ml.google.com.iam.gserviceaccount.com to access files in bucket-netskink-tda

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   234    0   234    0     0    480      0 --:--:-- --:--:-- --:--:--   480
No changes to gs://bucket-netskink-tda/
No changes to gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/train.csv-00000-of-00005
No changes to gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/train.csv-00001-of-00005
No changes to gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/train.csv-00002-of-00005
No changes to gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/train.csv-00003-of-00005
No changes to gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/train.csv-00004-of-00005
No changes to gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/valid.csv-00000-of-00002
No changes to gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/valid.csv-00001-of-00002
No changes to gs://bucket-netskink-tda/
```

2. Specifying query to pull the data

Let's pull out a few extra columns from the timestamp.

```
def create_query(phase, EVERY_N):
    if EVERY_N == None:
        EVERY_N = 4 #use full dataset

    #select and pre-process fields
    base_query = """
SELECT
  (tolls_amount + fare_amount) AS fare_amount,
  DAYOFWEEK(pickup_datetime) AS dayofweek,
  HOUR(pickup_datetime) AS hourofday,
  pickup_longitude AS pickuplon,
  pickup_latitude AS pickuplat,
  dropoff_longitude AS dropofflon,
  dropoff_latitude AS dropofflat,
  passenger_count*1.0 AS passengers,
  CONCAT(STRING(pickup_datetime), STRING(pickup_longitude), STRING(pickup_latitude), STRING(dropoff_latitude), STRING(dropoff_longitude)) AS key
FROM
  [nyc-tlc:yellow.trips]
WHERE
  trip_distance > 0
  AND 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
"""

    #add subsampling criteria by modding with hashkey
    if phase == 'train':
        query = "{} AND ABS(HASH(pickup_datetime)) % {} < 2".format(base_query,EVERY_N)
    elif phase == 'valid':
        query = "{} AND ABS(HASH(pickup_datetime)) % {} == 2".format(base_query,EVERY_N)
    elif phase == 'test':
        query = "{} AND ABS(HASH(pickup_datetime)) % {} == 3".format(base_query,EVERY_N)
    return query

    print create_query('valid', 100) #example query using 1% of data
```

Try the query above in https://bigquery.cloud.google.com/table/nyc-tlc:yellow.trips (https://bigquery.cloud.google.com/table/nyc-tlc:yellow.trips) if you want to see what it does (ADD LIMIT 10 to the query!)

3. Preprocessing Dataflow job from BigQuery

This code reads from BigQuery and saves the data as-is on Google Cloud Storage. We can do additional preprocessing and cleanup inside Dataflow, but then we'll have to remember to repeat that prepreprocessing during inference. It is better to use tf.transform which will do this book-keeping for you, or to do preprocessing within your TensorFlow model. We will look at this in future notebooks. For now, we are simply moving data from BigQuery to CSV using Dataflow.

While we could read from BQ directly from TensorFlow (See: https://www.tensorflow.org/api_docs/python/tf/contrib/cloud/BigQueryReader (https://www.tensorflow.org/api_docs/python/tf/contrib/cloud/BigQueryReader)), it is quite convenient to export to CSV and do the training off CSV. Let's use Dataflow to do this at scale.

Because we are running this on the Cloud, you should go to the GCP Console (https://console.cloud.google.com/dataflow (https://console.cloud.google.com/dataflow)) to look at the status of the job. It will take several minutes for the preprocessing job to launch.

```
%%bash
# Remove the beam output directory
gsutil -m rm -rf gs://${BUCKET}/${MODEL_NAME}/${PREPROC_DIR}/
```

```
Removing gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/train.csv-00003-of-00005#1536865475368792...
Removing gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/train.csv-00002-of-00005#1536865475333605...
Removing gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/train.csv-00001-of-00005#1536865475343836...
Removing gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/train.csv-00004-of-00005#1536865475377058...
Removing gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/train.csv-00000-of-00005#1536865475338131...
Removing gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/valid.csv-00000-of-00002#1536865474125364...
Removing gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/valid.csv-00001-of-00002#1536865474135334...
/ [7/7 objects] 100% Done
Operation completed over 7 objects.
```

```
import datetime

####
# Arguments:
#   -rowdict: Dictionary. The beam bigquery reader returns a PCollection in
#     which each row is represented as a python dictionary
# Returns:
#   -rowstring: a comma separated string representation of the record with dayofweek
#     converted from int to string (e.g. 3 --> Tue)
####
def to_csv(rowdict):
    days = ['null', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat']
    CSV_COLUMNS = 'fare_amount,dayofweek,hourofday,pickuplon,pickuplat,dropofflon,dropofflat,passengers,key'.split(',')
    rowdict['dayofweek'] = days[rowdict['dayofweek']]
    rowstring = ','.join([str(rowdict[k]) for k in CSV_COLUMNS])
    return rowstring


####
# Arguments:
#   -EVERY_N: Integer. Sample one out of every N rows from the full dataset.
#     Larger values will yield smaller sample
#   -RUNNER: 'DirectRunner' or 'DataflowRunner'. Specfy to run the pipeline
#     locally or on Google Cloud respectively.
# Side-effects:
#   -Creates and executes dataflow pipeline.
#     See https://beam.apache.org/documentation/programming-guide/#creating-a-pipeline
####
def preprocess(EVERY_N, RUNNER):
    job_name = 'preprocess-taxifeatures' + '-' + datetime.datetime.now().strftime('%y%m%d-%H%M%S')
    print 'Launching Dataflow job {} ... hang on'.format(job_name)
    OUTPUT_DIR = 'gs://{0}/{1}/{2}/'.format(BUCKET, MODEL_NAME, PREPROC_DIR)
    print(".. using output Dir: {}".format(OUTPUT_DIR) )

    #dictionary of pipeline options
    options = {
        'staging_location': os.path.join(OUTPUT_DIR, 'tmp', 'staging'),
        'temp_location': os.path.join(OUTPUT_DIR, 'tmp'),
        'job_name': 'preprocess-taxifeatures' + '-' + datetime.datetime.now().strftime('%y%m%d-%H%M%S'),
        'project': PROJECT,
        'runner': RUNNER
    }
    #instantiate PipelineOptions object using options dictionary
    opts = beam.pipeline.PipelineOptions(flags=[], **options)
    #instantantiate Pipeline object using PipelineOptions
    p = beam.Pipeline(options=opts)
    for phase in ['train', 'valid']:
        print("Phase: {}".format(phase))
        query = create_query(phase, EVERY_N)
        outfile = os.path.join(OUTPUT_DIR, '{}.csv'.format(phase))
        (
          p | 'read_{}'.format(phase) >> beam.io.Read(beam.io.BigQuerySource(query=query))
            | 'tocsv_{}'.format(phase) >> beam.Map(to_csv)
            | 'write_{}'.format(phase) >> beam.io.Write(beam.io.WriteToText(outfile))
        )

    p.run()
```

Run pipeline locally

```
# This works but be aware it takes about 30 seconds before the preprocessing folder appears in Google Cloud Platform Storage Console UI
preprocess(50*10000, 'DirectRunner')
```

```
Launching Dataflow job preprocess-taxifeatures-180913-192618 ... hang on
.. using output Dir: gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/
Phase: train
Phase: valid

/usr/local/envs/py2env/lib/python2.7/site-packages/oauth2client/contrib/gce.py:99: UserWarning: You have requested explicit scopes to be used with a GCE service account.
Using this argument will have no effect on the actual scopes for tokens
requested. These scopes are set at VM instance creation time and
can't be overridden in the request.

  warnings.warn(_SCOPES_WARNING)
WARNING:root:Dataset netskink-tda-test:temp_dataset_bf3ce4bbb22b4ef28b330c586108fd85 does not exist so we will create it as temporary with location=None
WARNING:root:Dataset netskink-tda-test:temp_dataset_e158683950b74897ab06d27c3954091b does not exist so we will create it as temporary with location=None
```

Run pipleline on cloud on a larger sample size.

```
# TODO: This fails even though I have the Cloud Dataflow API enabled.
#preprocess(50*100, 'DataflowRunner')
#change first arg to None to preprocess full dataset
```

```
/usr/local/envs/py2env/lib/python2.7/site-packages/apache_beam/io/gcp/gcsio.py:113: DeprecationWarning: object() takes no parameters
  super(GcsIO, cls).__new__(cls, storage_client))
```

Once the job completes, observe the files created in Google Cloud Storage for training and validation

```
%%bash
gsutil ls -l gs://${BUCKET}/${MODEL_NAME}/${PREPROC_DIR}/
```

```
    114721  2018-09-13T19:26:48Z  gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/train.csv-00000-of-00005
    114927  2018-09-13T19:26:48Z  gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/train.csv-00001-of-00005
    113891  2018-09-13T19:26:48Z  gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/train.csv-00002-of-00005
    114336  2018-09-13T19:26:48Z  gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/train.csv-00003-of-00005
    108806  2018-09-13T19:26:48Z  gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/train.csv-00004-of-00005
    113600  2018-09-13T19:26:47Z  gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/valid.csv-00000-of-00002
    106741  2018-09-13T19:26:47Z  gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/valid.csv-00001-of-00002
TOTAL: 7 objects, 787022 bytes (768.58 KiB)
```

```
%%bash
#print first 10 lines of first shard of train.csv
# Notice the file has nine columns.
# Fare Amount, Day of Week, Hour of Day, Pick Up Long, Pick Up Lat, Drop Off Lon, Drop Off Lat, Passengers and Key which
# is a combination of pickup time and the lat long pairs
gsutil cat "gs://${BUCKET}/${MODEL_NAME}/${PREPROC_DIR}/train.csv-00000-of-*" | head
```

```
14.1,Thu,0,-73.973489,40.752214,-73.963907,40.807841,1.0,2012-08-02 00:33:03.000000-73.973540.752240.8078-73.9639
16.1,Sun,0,-73.964025,40.710115,-74.007547,40.741085,2.0,2012-04-01 00:55:49.000000-73.96440.710140.7411-74.0075
8.6,Sat,0,-73.947914,40.740628,-73.947982,40.743065,4.0,2009-01-17 00:04:32.000000-73.947940.740640.7431-73.948
30.5,Fri,0,-73.9804077148,40.7457847595,-74.0038757324,40.6466941833,3.0,2015-03-20 00:37:15.000000-73.980440.745840.6467-74.0039
7.4,Sat,0,-73.972259,40.752311,-73.987083,40.728431,3.0,2009-01-17 00:04:32.000000-73.972340.752340.7284-73.9871
15.3,Tue,0,-73.996457,40.742907,-73.96584,40.677997,1.0,2011-11-29 00:46:00.000000-73.996540.742940.678-73.9658
17.3,Thu,0,-73.989998,40.761777,-73.985523,40.688012,1.0,2009-05-28 00:09:17.000000-73.9940.761840.688-73.9855
24.1,Tue,0,-73.999955,40.732977,-73.956437,40.824042,1.0,2011-11-29 00:46:00.000000-7440.73340.824-73.9564
22.1,Tue,0,-73.979457,40.764863,-73.926718,40.863352,3.0,2011-11-29 00:46:00.000000-73.979540.764940.8634-73.9267
28.0,Sat,0,-73.996712,40.72562,-73.91038,40.703752,1.0,2013-03-23 00:47:20.000000-73.996740.725640.7038-73.9104
```

4. Develop model with new inputs

Download the first shard of the preprocessed data to enable local development.

```
%%bash
mkdir sample
gsutil cp "gs://${BUCKET}/${MODEL_NAME}/${PREPROC_DIR}/train.csv-00000-of-*" sample/train.csv
gsutil cp "gs://${BUCKET}/${MODEL_NAME}/${PREPROC_DIR}/valid.csv-00000-of-*" sample/valid.csv
```

```
mkdir: cannot create directory ‘sample’: File exists
Copying gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/train.csv-00000-of-00005...
/ [1 files][112.0 KiB/112.0 KiB]
Operation completed over 1 objects/112.0 KiB.
Copying gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/valid.csv-00000-of-00002...
/ [1 files][110.9 KiB/110.9 KiB]
Operation completed over 1 objects/110.9 KiB.
```

We have two new inputs in the INPUT_COLUMNS, three engineered features, and the estimator involves bucketization and feature crosses.

```
%%bash
grep -A 20 "INPUT_COLUMNS =" ${MODEL_NAME}/trainer/model.py
```

```
INPUT_COLUMNS = [
    # Define features
    tf.feature_column.categorical_column_with_vocabulary_list('dayofweek', vocabulary_list = ['Sun', 'Mon', 'Tues', 'Wed', 'Thu', 'Fri', 'Sat']),
    tf.feature_column.categorical_column_with_identity('hourofday', num_buckets = 24),

    # Numeric columns
    tf.feature_column.numeric_column('pickuplat'),
    tf.feature_column.numeric_column('pickuplon'),
    tf.feature_column.numeric_column('dropofflat'),
    tf.feature_column.numeric_column('dropofflon'),
    tf.feature_column.numeric_column('passengers'),

    # Engineered features that are created in the input_fn
    tf.feature_column.numeric_column('latdiff'),
    tf.feature_column.numeric_column('londiff'),
    tf.feature_column.numeric_column('euclidean')
]

# Build the estimator
def build_estimator(model_dir, nbuckets, hidden_units):
    """
```

```
%%bash
grep -A 50 "build_estimator" ${MODEL_NAME}/trainer/model.py
```

```
def build_estimator(model_dir, nbuckets, hidden_units):
    """
     Build an estimator starting from INPUT COLUMNS.
     These include feature transformations and synthetic features.
     The model is a wide-and-deep model.
  """

    # Input columns
    (dayofweek, hourofday, plat, plon, dlat, dlon, pcount, latdiff, londiff, euclidean) = INPUT_COLUMNS

    # Bucketize the lats & lons
    latbuckets = np.linspace(38.0, 42.0, nbuckets).tolist()
    lonbuckets = np.linspace(-76.0, -72.0, nbuckets).tolist()
    b_plat = tf.feature_column.bucketized_column(plat, latbuckets)
    b_dlat = tf.feature_column.bucketized_column(dlat, latbuckets)
    b_plon = tf.feature_column.bucketized_column(plon, lonbuckets)
    b_dlon = tf.feature_column.bucketized_column(dlon, lonbuckets)

    # Feature cross
    ploc = tf.feature_column.crossed_column([b_plat, b_plon], nbuckets * nbuckets)
    dloc = tf.feature_column.crossed_column([b_dlat, b_dlon], nbuckets * nbuckets)
    pd_pair = tf.feature_column.crossed_column([ploc, dloc], nbuckets ** 4 )
    day_hr =  tf.feature_column.crossed_column([dayofweek, hourofday], 24 * 7)

    # Wide columns and deep columns.
    wide_columns = [
        # Feature crosses
        dloc, ploc, pd_pair,
        day_hr,

        # Sparse columns
        dayofweek, hourofday,

        # Anything with a linear relationship
        pcount
    ]

    deep_columns = [
        # Embedding_column to "group" together ...
        tf.feature_column.embedding_column(pd_pair, 10),
        tf.feature_column.embedding_column(day_hr, 10),

        # Numeric columns
        plat, plon, dlat, dlon,
        latdiff, londiff, euclidean
    ]

    estimator = tf.estimator.DNNLinearCombinedRegressor(
        model_dir = model_dir,
        linear_feature_columns = wide_columns,
        dnn_feature_columns = deep_columns,
--
    estimator = build_estimator(args['output_dir'], args['nbuckets'], args['hidden_units'].split(' '))
    train_spec = tf.estimator.TrainSpec(
        input_fn = read_dataset(
            filename = args['train_data_paths'],
            mode = tf.estimator.ModeKeys.TRAIN,
            batch_size = args['train_batch_size']),
        max_steps = args['train_steps'])
    exporter = tf.estimator.LatestExporter('exporter', serving_input_fn)
    eval_spec = tf.estimator.EvalSpec(
        input_fn = read_dataset(
            filename = args['eval_data_paths'],
            mode = tf.estimator.ModeKeys.EVAL,
            batch_size = args['eval_batch_size']),
        steps = 100,
        exporters = exporter)
    tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)

# If we want to use TFRecords instead of CSV
def gzip_reader_fn():
    return tf.TFRecordReader(options=tf.python_io.TFRecordOptions(
            compression_type = tf.python_io.TFRecordCompressionType.GZIP))

def generate_tfrecord_input_fn(data_paths, num_epochs = None, batch_size = 512, mode = tf.estimator.ModeKeys.TRAIN):
    def get_input_features():
        # Read the tfrecords. Same input schema as in preprocess
        input_schema = {}
        if mode != tf.estimator.ModeKeys.INFER:
            input_schema[LABEL_COLUMN] = tf.FixedLenFeature(shape = [1], dtype = tf.float32, default_value = 0.0)
        for name in ['dayofweek', 'key']:
            input_schema[name] = tf.FixedLenFeature(shape = [1], dtype = tf.string, default_value = 'null')
        for name in ['hourofday']:
            input_schema[name] = tf.FixedLenFeature(shape = [1], dtype = tf.int64, default_value = 0)
        for name in SCALE_COLUMNS:
            input_schema[name] = tf.FixedLenFeature(shape = [1], dtype = tf.float32, default_value = 0.0)

        # How?
        keys, features = tf.contrib.learn.io.read_keyed_batch_features(
            data_paths[0] if len(data_paths) == 1 else data_paths,
            batch_size,
            input_schema,
            reader = gzip_reader_fn,
            reader_num_threads = 4,
            queue_capacity = batch_size * 2,
            randomize_input = (mode != tf.estimator.ModeKeys.EVAL),
            num_epochs = (1 if mode == tf.estimator.ModeKeys.EVAL else num_epochs))
```

```
            target = features.pop(LABEL_COLUMN)
            features[KEY_FEATURE_COLUMN] = keys
            return add_engineered(features), target

        # Return a function to input the features into the model from a data path.
        return get_input_features
```

```
%%bash
grep -A 15 "add_engineered(" ${MODEL_NAME}/trainer/model.py
```

```
    def add_engineered(features):
        # this is how you can do feature engineering in TensorFlow
        lat1 = features['pickuplat']
        lat2 = features['dropofflat']
        lon1 = features['pickuplon']
        lon2 = features['dropofflon']
        latdiff = (lat1 - lat2)
        londiff = (lon1 - lon2)

        # set features for distance with sign that indicates direction
        features['latdiff'] = latdiff
        features['londiff'] = londiff
        dist = tf.sqrt(latdiff * latdiff + londiff * londiff)
        features['euclidean'] = dist
        return features

    --
        return tf.estimator.export.ServingInputReceiver(add_engineered(features), feature_placeholders)

    # Create input function to load data into datasets
    def read_dataset(filename, mode, batch_size = 512):
        def _input_fn():
            def decode_csv(value_column):
                columns = tf.decode_csv(value_column, record_defaults = DEFAULTS)
                features = dict(zip(CSV_COLUMNS, columns))
                label = features.pop(LABEL_COLUMN)
                return add_engineered(features), label

            # Create list of files that match pattern
            file_list = tf.gfile.Glob(filename)

            # Create dataset from file list
            dataset = tf.data.TextLineDataset(file_list).map(decode_csv)

            if mode == tf.estimator.ModeKeys.TRAIN:
                num_epochs = None # indefinitely
                dataset = dataset.shuffle(buffer_size = 10 * batch_size)
            else:
                num_epochs = 1 # end-of-input after this

            dataset = dataset.repeat(num_epochs).batch(batch_size)
            batch_features, batch_labels = dataset.make_one_shot_iterator().get_next()
    --
            return add_engineered(features), target

        # Return a function to input the features into the model from a data path.
        return get_input_features

    def add_eval_metrics(labels, predictions):
        pred_values = predictions['predictions']
        return {
            'rmse': tf.metrics.root_mean_squared_error(labels, pred_values)
        }
```

Running the Python module from the command-line

Try out the new model on the local sample to make sure it works fine.

Clean model training dir/output dir

```
%%bash
# This is so that the trained model is started fresh each time. However, this needs to be done before
# tensorboard is started
rm -rf $PWD/${TRAINING_DIR}
```

Monitor using Tensorboard

```
from google.datalab.ml import TensorBoard
print('Using dir ' + './'+ TRAINING_DIR)
TensorBoard().start('./'+ TRAINING_DIR)
```

```
/usr/local/envs/py2env/lib/python2.7/site-packages/sklearn/utils/__init__.py:10: RuntimeWarning: numpy.dtype size changed, may indicate binary incompatibility. Expected 96, got 88
  from .murmurhash import murmurhash3_32
/usr/local/envs/py2env/lib/python2.7/site-packages/sklearn/utils/extmath.py:24: RuntimeWarning: numpy.dtype size changed, may indicate binary incompatibility. Expected 96, got 88
  from ._logistic_sigmoid import _log_logistic_sigmoid
/usr/local/envs/py2env/lib/python2.7/site-packages/sklearn/metrics/cluster/supervised.py:23: RuntimeWarning: numpy.dtype size changed, may indicate binary incompatibility. Expected 96, got 88
  from .expected_mutual_info_fast import expected_mutual_information
/usr/local/envs/py2env/lib/python2.7/site-packages/sklearn/metrics/pairwise.py:30: RuntimeWarning: numpy.dtype size changed, may indicate binary incompatibility. Expected 96, got 88
  from .pairwise_fast import _chi2_kernel_fast, _sparse_manhattan

Using dir ./taxifare_feateng_trained

TensorBoard was started successfully with pid 9290. Click here (/_proxy/52441/) to access it.

9290
```

Train the model locally

```
%%bash
export PYTHONPATH=${PYTHONPATH}:${PWD}/${MODEL_NAME}
python -m trainer.task \
  --train_data_paths=${PWD}/sample/train.csv \
  --eval_data_paths=${PWD}/sample/valid.csv  \
  --output_dir=${PWD}/${TRAINING_DIR} \
  --train_steps=1000 \
  --job-dir=/tmp
```

```
%%bash
ls ${TRAINING_DIR}/export/exporter/
```

```
%%writefile /tmp/test.json
{"dayofweek": "Sun", "hourofday": 17, "pickuplon": -73.885262, "pickuplat": 40.773008, "dropofflon": -73.987232, "dropofflat": 40.732403, "passengers": 2}
```

```
%%bash
model_dir=$(ls ${PWD}/${TRAINING_DIR}/export/exporter | tail -1)
gcloud ml-engine local predict \
  --model-dir=${PWD}/${TRAINING_DIR}/export/exporter/${model_dir} \
  --json-instances=/tmp/test.json
```

```
#if gcloud ml-engine local predict fails, might need to update glcoud
#!gcloud --quiet components update
```

Stop Tensorboard

The training directory will be deleted. Stop the exising tensorboard before removing the directory its using.

```
pids_df = TensorBoard.list()
if not pids_df.empty:
    for pid in pids_df['pid']:
        TensorBoard().stop(pid)
        print('Stopped TensorBoard with pid {}'.format(pid))
```

```
Stopped TensorBoard with pid 9290
```

Clean model training dir/output dir

```
%%bash
# This is so that the trained model is started fresh each time. However, this needs to be done before
# tensorboard is started
rm -rf $PWD/${TRAINING_DIR}
```

Restart tensorboard for monitoring

```
TensorBoard().start('./'+ TRAINING_DIR)
```

```
TensorBoard was started successfully with pid 9312. Click here (/_proxy/40493/) to access it.

9312
```

Running locally using gcloud

```
%%bash
# Use Cloud Machine Learning Engine to train the model in local file system
gcloud ml-engine local train \
  --module-name=trainer.task \
  --package-path=${PWD}/${MODEL_NAME}/trainer \
  -- \
  --train_data_paths=${PWD}/sample/train.csv \
  --eval_data_paths=${PWD}/sample/valid.csv  \
  --train_steps=1000 \
  --output_dir=${PWD}/${TRAINING_DIR}
```

```
/usr/local/envs/py2env/lib/python2.7/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
INFO:tensorflow:TF_CONFIG environment variable: {u'environment': u'cloud', u'cluster': {}, u'job': {u'args': [u'--train_data_paths=/content/datalab/notebooks/training-data-analyst/courses/machine_learning/deepdive/04_features/feateng/sample/train.csv', u'--eval_data_paths=/content/datalab/notebooks/training-data-analyst/courses/machine_learning/deepdive/04_features/feateng/sample/valid.csv', u'--train_steps=1000', u'--output_dir=/content/datalab/notebooks/training-data-analyst/courses/machine_learning/deepdive/04_features/feateng/taxifare_feateng_trained'], u'job_name': u'trainer.task'}, u'task': {}}
INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_save_checkpoints_secs': 600, '_session_config': None, '_keep_checkpoint_max': 5, '_task_type': 'worker', '_train_distribute': None, '_is_chief': True, '_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec object at 0x7fb997f68c50>, '_evaluation_master': '', '_save_checkpoints_steps': None, '_keep_checkpoint_every_n_hours': 10000, '_service': None, '_num_ps_replicas': 0, '_tf_random_seed': None, '_master': '', '_num_worker_replicas': 1, '_task_id': 0, '_log_step_count_steps': 100, '_model_dir': '/content/datalab/notebooks/training-data-analyst/courses/machine_learning/deepdive/04_features/feateng/taxifare_feateng_trained/', '_global_id_in_cluster': 0, '_save_summary_steps': 100}
INFO:tensorflow:Using config: {'_save_checkpoints_secs': 600, '_session_config': None, '_keep_checkpoint_max': 5, '_task_type': 'worker', '_train_distribute': None, '_is_chief': True, '_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec object at 0x7fb997f68c10>, '_evaluation_master': '', '_save_checkpoints_steps': None, '_keep_checkpoint_every_n_hours': 10000, '_service': None, '_num_ps_replicas': 0, '_tf_random_seed': None, '_master': '', '_num_worker_replicas': 1, '_task_id': 0, '_log_step_count_steps': 100, '_model_dir': '/content/datalab/notebooks/training-data-analyst/courses/machine_learning/deepdive/04_features/feateng/taxifare_feateng_trained/', '_global_id_in_cluster': 0, '_save_summary_steps': 100}
INFO:tensorflow:Running training and evaluation locally (non-distributed).
INFO:tensorflow:Start train and evaluate loop. The evaluate will happen after 600 secs (eval_spec.throttle_secs) or training is finished.
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
2018-09-13 19:28:24.964543: I tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into /content/datalab/notebooks/training-data-analyst/courses/machine_learning/deepdive/04_features/feateng/taxifare_feateng_trained/model.ckpt.
INFO:tensorflow:loss = 461923.84, step = 1
INFO:tensorflow:global_step/sec: 18.4932
INFO:tensorflow:loss = 110909.76, step = 101 (5.408 sec)
INFO:tensorflow:global_step/sec: 23.0033
INFO:tensorflow:loss = 120676.945, step = 201 (4.347 sec)
INFO:tensorflow:global_step/sec: 22.2755
INFO:tensorflow:loss = 112583.14, step = 301 (4.489 sec)
INFO:tensorflow:global_step/sec: 21.6274
INFO:tensorflow:loss = 109312.0, step = 401 (4.624 sec)
INFO:tensorflow:global_step/sec: 22.9675
INFO:tensorflow:loss = 101399.05, step = 501 (4.354 sec)
INFO:tensorflow:global_step/sec: 22.3444
INFO:tensorflow:loss = 109581.61, step = 601 (4.475 sec)
INFO:tensorflow:global_step/sec: 22.9131
INFO:tensorflow:loss = 114956.17, step = 701 (4.364 sec)
INFO:tensorflow:global_step/sec: 22.3908
INFO:tensorflow:loss = 102907.17, step = 801 (4.466 sec)
INFO:tensorflow:global_step/sec: 23.0858
INFO:tensorflow:loss = 108584.14, step = 901 (4.331 sec)
INFO:tensorflow:Saving checkpoints for 1000 into /content/datalab/notebooks/training-data-analyst/courses/machine_learning/deepdive/04_features/feateng/taxifare_feateng_trained/model.ckpt.
INFO:tensorflow:Loss for final step: 118065.47.
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-09-13-19:29:14
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from /content/datalab/notebooks/training-data-analyst/courses/machine_learning/deepdive/04_features/feateng/taxifare_feateng_trained/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2018-09-13-19:29:16
INFO:tensorflow:Saving dict for global step 1000: average_loss = 179.10988, global_step = 1000, loss = 89554.94, rmse = 13.383194
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Signatures INCLUDED in export for Classify: None
INFO:tensorflow:Signatures INCLUDED in export for Regress: None
INFO:tensorflow:Signatures INCLUDED in export for Predict: ['predict']
INFO:tensorflow:Signatures EXCLUDED from export because they cannot be be served via TensorFlow Serving APIs:
INFO:tensorflow:'serving_default' : Regression input must be a single string Tensor; got {'dayofweek': <tf.Tensor 'Placeholder_8:0' shape=(?,) dtype=string>, 'passengers': <tf.Tensor 'Placeholder_4:0' shape=(?,) dtype=float32>, 'euclidean': <tf.Tensor 'Placeholder_7:0' shape=(?,) dtype=float32>, 'latdiff': <tf.Tensor 'Placeholder_5:0' shape=(?,) dtype=float32>, 'pickuplat': <tf.Tensor 'Placeholder:0' shape=(?,) dtype=float32>, 'dropofflat': <tf.Tensor 'Placeholder_2:0' shape=(?,) dtype=float32>, 'londiff': <tf.Tensor 'Placeholder_6:0' shape=(?,) dtype=float32>, 'hourofday': <tf.Tensor 'Placeholder_9:0' shape=(?,) dtype=int32>, 'pickuplon': <tf.Tensor 'Placeholder_1:0' shape=(?,) dtype=float32>, 'dropofflon': <tf.Tensor 'Placeholder_3:0' shape=(?,) dtype=float32>}
INFO:tensorflow:'regression' : Regression input must be a single string Tensor; got {'dayofweek': <tf.Tensor 'Placeholder_8:0' shape=(?,) dtype=string>, 'passengers': <tf.Tensor 'Placeholder_4:0' shape=(?,) dtype=float32>, 'euclidean': <tf.Tensor 'Placeholder_7:0' shape=(?,) dtype=float32>, 'latdiff': <tf.Tensor 'Placeholder_5:0' shape=(?,) dtype=float32>, 'pickuplat': <tf.Tensor 'Placeholder:0' shape=(?,) dtype=float32>, 'dropofflat': <tf.Tensor 'Placeholder_2:0' shape=(?,) dtype=float32>, 'londiff': <tf.Tensor 'Placeholder_6:0' shape=(?,) dtype=float32>, 'hourofday': <tf.Tensor 'Placeholder_9:0' shape=(?,) dtype=int32>, 'pickuplon': <tf.Tensor 'Placeholder_1:0' shape=(?,) dtype=float32>, 'dropofflon': <tf.Tensor 'Placeholder_3:0' shape=(?,) dtype=float32>}
WARNING:tensorflow:Export includes no default signature!
INFO:tensorflow:Restoring parameters from /content/datalab/notebooks/training-data-analyst/courses/machine_learning/deepdive/04_features/feateng/taxifare_feateng_trained/model.ckpt-1000
INFO:tensorflow:Assets added to graph.
INFO:tensorflow:No assets to write.
INFO:tensorflow:SavedModel written to: /content/datalab/notebooks/training-data-analyst/courses/machine_learning/deepdive/04_features/feateng/taxifare_feateng_trained/export/exporter/temp-1536866957/saved_model.pb
```

Stop Tensorboard

```
pids_df = TensorBoard.list()
if not pids_df.empty:
    for pid in pids_df['pid']:
        TensorBoard().stop(pid)
        print('Stopped TensorBoard with pid {}'.format(pid))
```

```
Stopped TensorBoard with pid 9312
```

Submit training job using gcloud

Since the training data already exists in cloud storage, no need to copy the training data to the cloud. Please take note of the input data location.

Clean model training dir/output dir

```
%%bash
# This is so that the trained model is started fresh each time. However, this needs to be done before
# tensorboard is started
OUTDIR=gs://${BUCKET}/${MODEL_NAME}/${TRAINING_DIR}
# Clear the Cloud Storage Bucket used for the training job. (if it exists already)
echo "Clearing DIRECTORY: $OUTDIR"
gsutil -m rm -rf $OUTDIR
```

```
Clearing DIRECTORY: gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_trained

CommandException: 1 files/objects could not be removed.
```

Restart tensorboard for monitoring

```
OUTPUT_DIR = 'gs://{0}/{1}/{2}/'.format(BUCKET, MODEL_NAME, TRAINING_DIR)
print('Tensorboard using bucket {}'.format(OUTPUT_DIR))
TensorBoard().start(OUTPUT_DIR)
```

```
Tensorboard using bucket gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_trained/

TensorBoard was started successfully with pid 10153. Click here (/_proxy/54619/) to access it.

10153
```

Train the model using Cloud

```
%%bash
OUTDIR=gs://${BUCKET}/${MODEL_NAME}/${TRAINING_DIR}
JOBNAME=${MODEL_NAME}_$(date -u +%y%m%d_%H%M%S)
echo "DIRECTORY: $OUTDIR       REGION: $REGION      JOBNAME: $JOBNAME"

echo "Dirs used by this job"
echo "train data dir --> gs://${BUCKET}/${MODEL_NAME}/${PREPROC_DIR}/train*"
echo "eval data dir --> gs://${BUCKET}/${MODEL_NAME}/${PREPROC_DIR}/eval*"
echo "output dir --> $OUTDIR"
gcloud ml-engine jobs submit training $JOBNAME \
  --region=$REGION \
  --module-name=trainer.task \
  --package-path=${PWD}/${MODEL_NAME}/trainer \
  --job-dir=$OUTDIR \
  --staging-bucket=gs://$BUCKET \
  --scale-tier=BASIC \
  --runtime-version=$TFVERSION \
  -- \
  --train_data_paths="gs://${BUCKET}/${MODEL_NAME}/${PREPROC_DIR}/train*" \
  --eval_data_paths="gs://${BUCKET}/${MODEL_NAME}/${PREPROC_DIR}/valid*"  \
  --train_steps=1000 \
  --output_dir=$OUTDIR
```

```
DIRECTORY: gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_trained       REGION: us-east1       JOBNAME: taxifare_feateng_180913_194045
Dirs used by this job
train data dir --> gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/train*
eval data dir --> gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_preproc/eval*
output dir --> gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_trained
jobId: taxifare_feateng_180913_194045
state: QUEUED

Job [taxifare_feateng_180913_194045] submitted successfully.
Your job is still active. You may view the status of your job with the command

  $ gcloud ml-engine jobs describe taxifare_feateng_180913_194045

or continue streaming the logs with the command

  $ gcloud ml-engine jobs stream-logs taxifare_feateng_180913_194045
```

Deploy model

Find out the actual name of the subdirectory where the model is stored and use it to deploy the model. Deploying model will take up to 5 minutes.

```
%%bash
gsutil ls gs://${BUCKET}/${MODEL_NAME}/${TRAINING_DIR}/export/exporter
```

```
gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_trained/export/exporter/
gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_trained/export/exporter/1536867776/
gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_trained/export/exporter/1536867866/
```

Deploy model : step 1 - remove version info

Before an existing cloud model can be removed, it must have any version info removed. If an existing model with the specified version does not exist, this command will generate an error but that is ok. It just means the model does not exist or a model with the specified version is not available to remove version info.

```
%%bash
gcloud ml-engine versions delete ${MODEL_VERSION} --model ${MODEL_NAME}
```

```
This will delete version [v2]...

Do you want to continue (Y/n)?
Deleting version [v2]......
.....done.
```

Deploy model: step 2 - remove existing model

Now that the version info is removed from an existing model, the actual model can be removed. If an existing model is not deployed, this command will generate an error but that is ok. It just means the model with the given name is not deployed.

```
%%bash
gcloud ml-engine models delete ${MODEL_NAME}
```

```
This will delete model [taxifare_feateng]...

Do you want to continue (Y/n)?
Deleting model [taxifare_feateng]...
done.
```

Deploy model: step 3 - deploy new model

```
%%bash
gcloud ml-engine models create ${MODEL_NAME} --regions ${REGION}
```

```
Created ml engine model [projects/netskink-tda-test/models/taxifare_feateng].
```

Deploy model: step 4 - add version info to the new model

```
%%bash
MODEL_LOCATION=$(gsutil ls gs://${BUCKET}/${MODEL_NAME}/${TRAINING_DIR}/export/exporter | tail -1)

echo "MODEL_LOCATION = ${MODEL_LOCATION}"

gcloud ml-engine versions create ${MODEL_VERSION} --model ${MODEL_NAME} --origin ${MODEL_LOCATION} --runtime-version $TFVERSION
```

```
MODEL_LOCATION = gs://bucket-netskink-tda/taxifare_feateng/taxifare_feateng_trained/export/exporter/1536867866/

Creating version (this might take a few minutes)......
.....done.
```

Prediction

Using the same data format as the locally trained model

Using gcloud ml-engine predict

```
%%bash
gcloud ml-engine predict --model=${MODEL_NAME} --version=${MODEL_VERSION} --json-instances=/tmp/test.json
```

```
{
  "error": "Prediction failed: Error during model execution: AbortionError(code=StatusCode.INVALID_ARGUMENT, details=\"input size does not match signature\")"
}
```

Using gcloud with the rest api

```
from googleapiclient import discovery
from oauth2client.client import GoogleCredentials
import json

credentials = GoogleCredentials.get_application_default()
api = discovery.build('ml', 'v1', credentials=credentials,
          discoveryServiceUrl='https://storage.googleapis.com/cloud-ml/discovery/ml_v1_discovery.json')

request_data = {'instances':
  [
    {
      'dayofweek': 'Sun',
      'hourofday': 17,
      'pickuplon': -73.885262,
      'pickuplat': 40.773008,
      'dropofflon': -73.987232,
      'dropofflat': 40.732403,
      'passengers': 2,
    }
  ]
}

parent = 'projects/%s/models/%s/versions/%s' % (PROJECT, MODEL_NAME, MODEL_VERSION)
response = api.projects().predict(body=request_data, name=parent).execute()
print "response={0}".format(response)
```

```
response={u'error': u'Prediction failed: Error during model execution: AbortionError(code=StatusCode.INVALID_ARGUMENT, details="input size does not match signature")'}
```

```
from googleapiclient import discovery
from oauth2client.client import GoogleCredentials
import json

credentials = GoogleCredentials.get_application_default()
api = discovery.build('ml', 'v1', credentials=credentials,
          discoveryServiceUrl='https://storage.googleapis.com/cloud-ml/discovery/ml_v1_discovery.json')

request_data = {'instances':
  [
      {
        'dayofweek': 'Sun',
        'hourofday': 17,
        'pickuplon': -73.885262,
        'pickuplat': 40.773008,
        'dropofflon': -73.987232,
        'dropofflat': 40.732403,
        'passengers': 2,
        'latdiff': 20.0,
        'londiff': 21.0,
        'euclidean': 15.0
      }
  ]
}

parent = 'projects/%s/models/%s/versions/%s' % (PROJECT, MODEL_NAME, MODEL_VERSION)
response = api.projects().predict(body=request_data, name=parent).execute()
print "response={0}".format(response)
```

```
response={u'predictions': [{u'predictions': [11.386539459228516]}]}
```

The RMSE is now 8.33249, an improvement over the 9.3 that we were getting ... of course, we won't know until we train/validate on a larger dataset. Still, this is promising. But before we do that, let's do hyper-parameter tuning.

6. Hyper-parameter tune

Look at hyper-parameter tuning notebook (hyperparam.ipynb) to decide what parameters to use for model. Based on that run, I ended up choosing:

1. train_batch_size: 512
2. nbuckets: 16
3. hidden_units: "64 64 64 8"

This gives an RMSE of 5, a considerable improvement from the 8.3 we were getting earlier ... Let's try this over a larger dataset.

Run Cloud training on 2 million row dataset

This run uses as input 2 million rows and takes ~20 minutes with 10 workers (STANDARD_1 pricing tier). The model is exactly the same as above. The only changes are to the input (to use the larger dataset) and to the Cloud MLE tier (to use STANDARD_1 instead of BASIC -- STANDARD_1 is approximately 10x more powerful than BASIC). Because the Dataflow preprocessing takes about 15 minutes, we train here using CSV files in a public bucket.

When doing distributed training, use train_steps instead of num_epochs. The distributed workers don't know how many rows there are, but we can calculate train_steps = num_rows * num_epochs / train_batch_size. In this case, we have 2141023 * 100 / 512 = 418168 train steps.

```
%%bash

#WARNING -- this uses significant resources and is optional. Remove this line to run the block.
#   --train_steps=418168

# NOTE: this is not using the same directory for the csv files.

OUTDIR=gs://${BUCKET}/${MODEL_NAME}/feateng2m
JOBNAME=${MODEL_NAME}_a_$(date -u +%y%m%d_%H%M%S)
TIER=STANDARD_1
echo $OUTDIR $REGION $JOBNAME
gsutil -m rm -rf $OUTDIR
gcloud ml-engine jobs submit training $JOBNAME \
  --region=$REGION \
  --module-name=trainer.task \
  --package-path=${PWD}/${MODEL_NAME}/trainer \
  --job-dir=$OUTDIR \
  --staging-bucket=gs://$BUCKET \
  --scale-tier=$TIER \
  --runtime-version=$TFVERSION \
  -- \
  --train_data_paths="gs://${BUCKET}/${PREPROC_DIR}/train*" \
  --eval_data_paths="gs://${BUCKET}/${PREPROC_DIR}/valid*"  \
  --output_dir=$OUTDIR \
  --train_steps=2000 \
  --train_batch_size=512 --nbuckets=16 --hidden_units="64 64 64 8"
```

Start Tensorboard

```
from google.datalab.ml import TensorBoard
OUTDIR='gs://{0}/taxifare_feat/feateng2m'.format(BUCKET)
print OUTDIR
TensorBoard().start(OUTDIR)
```

Stop Tensorboard

```
pids_df = TensorBoard.list()
if not pids_df.empty:
    for pid in pids_df['pid']:
        TensorBoard().stop(pid)
        print 'Stopped TensorBoard with pid {}'.format(pid)
```

The RMSE after training on the 2-million-row dataset is $3.03. This graph shows the improvements so far ...

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

df = pd.DataFrame({'Lab' : pd.Series(['1a', '2-3', '4a', '4b', '4c']),
            'Method' : pd.Series(['Heuristic Benchmark', 'tf.learn', '+Feature Eng.', '+ Hyperparam', '+ 2m rows']),
            'RMSE': pd.Series([8.026, 9.4, 8.3, 5.0, 3.03]) })

ax = sns.barplot(data = df, x = 'Method', y = 'RMSE')
ax.set_ylabel('RMSE (dollars)')
ax.set_xlabel('Labs/Methods')
plt.plot(np.linspace(-20, 120, 1000), [5] * 1000, 'b');
```

```
%%bash
gsutil -m mv gs://${BUCKET}/taxifare/ch4/  gs://${BUCKET}/taxifare/ch4_1m/
```