Machine Learning with TensorFlow in Vertex AI

1 hour 30 minutesFree

**GSP273**



**Overview**

In this lab you create a Vertex AI Workbench instance on which you devlop a TensorFlow model in Jupyter notebook. You train the model, create an input data pipeline, deploy it to an endpoint, and get predictions.

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

Vertex AI brings AutoML and AI Platform together into a unified API, client library, and user interface. With Vertex AI, both AutoML training and custom training are available options.

Vertex AI Workbench helps users quickly build end-to-end notebook-based workflows through deep integration with data services (like Dataproc, Dataflow, BigQuery, and Dataplex) and Vertex AI. It enables data scientists to connect to Google Cloud data services, analyze datasets, experiment with different modeling techniques, deploy trained models into production, and manage MLOps through the model lifecycle.

Vertex AI Workbench is a single development environment for the entire data science workflow.

This lab uses a set of code samples and scripts developed for [*Data Science on the Google Cloud Platform, 2nd Edition*](https://www.oreilly.com/library/view/data-science-on/9781098118945/) from O'Reilly Media, Inc.

Objectives

* Deploy Vertex AI Workbench instance
* Create minimal training, validation data
* Create the input data pipeline
* Create TensorFlow model
* Deploy model to Vertex AI
* Deploy Explainable AI model to Vertex AI
* Make predictions from the model endpoint

**Setup and requirements**

Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

To complete this lab, you need:

* Access to a standard internet browser (Chrome browser recommended).

**Note:** Use an Incognito or private browser window to run this lab. This prevents any conflicts between your personal account and the Student account, which may cause extra charges incurred to your personal account.

* Time to complete the lab---remember, once you start, you cannot pause a lab.

**Note:** If you already have your own personal Google Cloud account or project, do not use it for this lab to avoid extra charges to your account.

How to start your lab and sign in to the Google Cloud Console

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is the **Lab Details** panel with the following:
   * The **Open Google Console** button
   * Time remaining
   * The temporary credentials that you must use for this lab
   * Other information, if needed, to step through this lab
2. Click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.

***Tip:*** Arrange the tabs in separate windows, side-by-side.

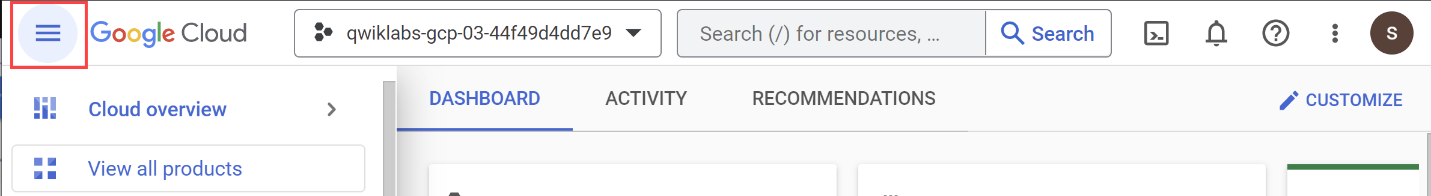
**Note:**If you see the **Choose an account** dialog, click **Use Another Account**.

1. If necessary, copy the **Username** from the **Lab Details** panel and paste it into the **Sign in** dialog. Click **Next**.
2. Copy the **Password** from the **Lab Details** panel and paste it into the **Welcome** dialog. Click **Next**.

**Important:**You must use the credentials from the left panel. Do not use your Google Cloud Skills Boost credentials.**Note:**Using your own Google Cloud account for this lab may incur extra charges.

1. Click through the subsequent pages:
   * Accept the terms and conditions.
   * Do not add recovery options or two-factor authentication (because this is a temporary account).
   * Do not sign up for free trials.

After a few moments, the Cloud Console opens in this tab.

**Note:** You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left. 

**Task 1. Deploy Vertex AI Workbench instance**

1. On the **Navigation Menu** (Navigation menu icon), click > **More Products** > **Vertex AI** > **Workbench**.

**Note:** Notebooks API must be enabled. In this lab, Notebooks API is enabled for you.

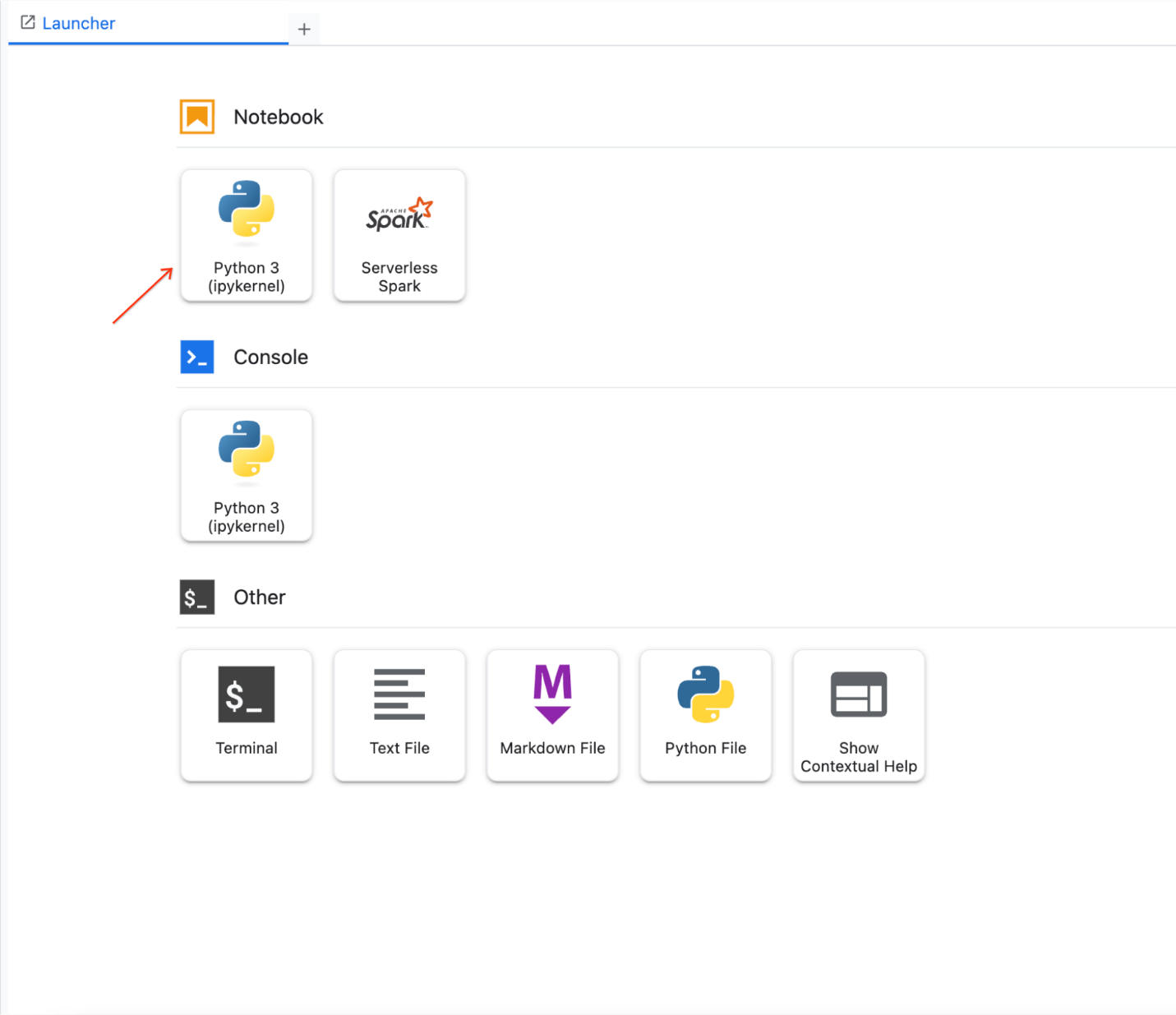
1. On the Notebook instances page, click **New Notebook** > **TensorFlow Enterprise** > **TensorFlow Enterprise 2.6 (with LTS)** > **Without GPUs**.
2. In the **New notebook** dialog, name the Notebook tensorflow.
3. Set **Region** to us-central1.
4. Set **Zone** to a zone within the selected region.
5. Leave all other fields at their default options, and click **Create**.

As the Notebook spins up, you see the tensorflow notebook listed in the Notebook list. When it's complete, Open Jupyterlab appears inline next to the tensorflow Notebook.

1. Click **Open JupyterLab**.

Your notebook is now set up.

1. In the **Notebook** launcher section click **Python 3** to open a new notebook.



To use a Notebook, you enter commands into a cell. Be sure you run the commands in the cell by either pressing **Shift + Enter**, or clicking the triangle on the Notebook top menu to **Run selected cells and advance**.

Please create the AI platform notebook instance with desired configuration.

Deploy Vertex AI Workbench instance

Check my progress

*Please create the AI platform notebook instance with desired configuration.*

**Task 2. Create minimal training and validation data**

* Import python libraries and set environment variables:

import os, json, math, shutil

import numpy as np

import tensorflow as tf

# environment variables used by bash cells

PROJECT=!(gcloud config get-value project)

PROJECT=PROJECT[0]

REGION = 'us-central1'

BUCKET='{}-dsongcp'.format(PROJECT)

os.environ['ENDPOINT\_NAME'] = 'flights'

os.environ['BUCKET'] = BUCKET

os.environ['REGION'] = REGION

os.environ['TF\_VERSION']='2-' + tf.\_\_version\_\_[2:3]

Copied!

content\_copy

**Note:**When pasting commands into the Jupyter notebook cell, remember to run the cell to be sure the last command in any sequence is executed before you proceed to the next step.

Export files that contain training, validation data

When the lab spins up, a few tables are created in the BigQuery dataset. In this section, you use BigQuery to create temporary tables in BigQuery that contain the data we need, and then export the table to CSV files on Google Cloud Storage. You then delete the temporary table. Moving further, read and process those CSV data files to create the training, validation, and full datasets you need for a custom TensorFlow model.

1. Create training dataset flights\_train\_data for model training:

%%bigquery

CREATE OR REPLACE TABLE dsongcp.flights\_train\_data AS

SELECT

IF(arr\_delay < 15, 1.0, 0.0) AS ontime,

dep\_delay,

taxi\_out,

distance,

origin,

dest,

EXTRACT(hour FROM dep\_time) AS dep\_hour,

IF (EXTRACT(dayofweek FROM dep\_time) BETWEEN 2 AND 6, 1, 0) AS is\_weekday,

UNIQUE\_CARRIER AS carrier,

dep\_airport\_lat,

dep\_airport\_lon,

arr\_airport\_lat,

arr\_airport\_lon

FROM dsongcp.flights\_tzcorr f

JOIN dsongcp.trainday t

ON f.FL\_DATE = t.FL\_DATE

WHERE

f.CANCELLED = False AND

f.DIVERTED = False AND

is\_train\_day = 'True'

Copied!

content\_copy

1. Create the evaluation dataset flights\_eval\_data for model evaluation:

%%bigquery

CREATE OR REPLACE TABLE dsongcp.flights\_eval\_data AS

SELECT

IF(arr\_delay < 15, 1.0, 0.0) AS ontime,

dep\_delay,

taxi\_out,

distance,

origin,

dest,

EXTRACT(hour FROM dep\_time) AS dep\_hour,

IF (EXTRACT(dayofweek FROM dep\_time) BETWEEN 2 AND 6, 1, 0) AS is\_weekday,

UNIQUE\_CARRIER AS carrier,

dep\_airport\_lat,

dep\_airport\_lon,

arr\_airport\_lat,

arr\_airport\_lon

FROM dsongcp.flights\_tzcorr f

JOIN dsongcp.trainday t

ON f.FL\_DATE = t.FL\_DATE

WHERE

f.CANCELLED = False AND

f.DIVERTED = False AND

is\_train\_day = 'False'

Copied!

content\_copy

1. Create the full dataset flights\_all\_data using the following code:

%%bigquery

CREATE OR REPLACE TABLE dsongcp.flights\_all\_data AS

SELECT

IF(arr\_delay < 15, 1.0, 0.0) AS ontime,

dep\_delay,

taxi\_out,

distance,

origin,

dest,

EXTRACT(hour FROM dep\_time) AS dep\_hour,

IF (EXTRACT(dayofweek FROM dep\_time) BETWEEN 2 AND 6, 1, 0) AS is\_weekday,

UNIQUE\_CARRIER AS carrier,

dep\_airport\_lat,

dep\_airport\_lon,

arr\_airport\_lat,

arr\_airport\_lon,

IF (is\_train\_day = 'True',

IF(ABS(MOD(FARM\_FINGERPRINT(CAST(f.FL\_DATE AS STRING)), 100)) < 60, 'TRAIN', 'VALIDATE'),

'TEST') AS data\_split

FROM dsongcp.flights\_tzcorr f

JOIN dsongcp.trainday t

ON f.FL\_DATE = t.FL\_DATE

WHERE

f.CANCELLED = False AND

f.DIVERTED = False

Copied!

content\_copy

1. Export the training, validation, and full datasets to CSV file format the Google Cloud Storage bucket:

This will take about 2 minutes to complete.

1. Wait until you receive output from running the following bash script in your notebook cell:

%%bash

PROJECT=$(gcloud config get-value project)

for dataset in "train" "eval" "all"; do

TABLE=dsongcp.flights\_${dataset}\_data

CSV=gs://${BUCKET}/ch9/data/${dataset}.csv

echo "Exporting ${TABLE} to ${CSV} and deleting table"

bq --project\_id=${PROJECT} extract --destination\_format=CSV $TABLE $CSV

bq --project\_id=${PROJECT} rm -f $TABLE

done

Copied!

content\_copy

1. List exported objects to Google Cloud Storage bucket using the following code:

!gsutil ls -lh gs://{BUCKET}/ch9/data

Copied!

content\_copy

Create training and validation dataset

Check my progress

**Task 3. Create the input data**

Setup in notebook

1. For development purposes, train for a few epochs. That's why the NUM\_EXAMPLES is so low.

DEVELOP\_MODE = True

NUM\_EXAMPLES = 5000\*1000

Copied!

content\_copy

1. Assign your training and validation data URI to training\_data\_uri and validation\_data\_uri respectively:

training\_data\_uri = 'gs://{}/ch9/data/train\*'.format(BUCKET)

validation\_data\_uri = 'gs://{}/ch9/data/eval\*'.format(BUCKET)

Copied!

content\_copy

1. Set up Model Parameters using the following code-block:

NBUCKETS = 5

NEMBEDS = 3

TRAIN\_BATCH\_SIZE = 64

DNN\_HIDDEN\_UNITS = '64,32'

Copied!

content\_copy

Reading data into TensorFlow

1. To read the CSV files from Google Cloud Storage into TensorFlow, use a method from the tf.data package:

if DEVELOP\_MODE:

train\_df = tf.data.experimental.make\_csv\_dataset(training\_data\_uri, batch\_size=5)

for n, data in enumerate(train\_df):

numpy\_data = {k: v.numpy() for k, v in data.items()}

print(n, numpy\_data)

if n==1: break

Copied!

content\_copy

Write features\_and\_labels() and read\_dataset() functions. The read\_dataset() function reads the training data, yielding batch\_size examples each time, and allows you to stop iterating once a certain number of examples have been read.

The dataset contains all the columns in the CSV file, named according to the header line. The data consists of both features and the label. It’s better to separate them by writing the features\_and\_labels() function to make the later code easier to read. Hence, we’ll apply a pop() function to the dictionary and return a tuple of features and labels).

1. Enter and run the following code:

def features\_and\_labels(features):

label = features.pop('ontime')

return features, label

def read\_dataset(pattern, batch\_size, mode=tf.estimator.ModeKeys.TRAIN, truncate=None):

dataset = tf.data.experimental.make\_csv\_dataset(pattern, batch\_size, num\_epochs=1)

dataset = dataset.map(features\_and\_labels)

if mode == tf.estimator.ModeKeys.TRAIN:

dataset = dataset.shuffle(batch\_size\*10)

dataset = dataset.repeat()

dataset = dataset.prefetch(1)

if truncate is not None:

dataset = dataset.take(truncate)

return dataset

if DEVELOP\_MODE:

print("Checking input pipeline")

one\_item = read\_dataset(training\_data\_uri, batch\_size=2, truncate=1)

print(list(one\_item)) # should print one batch of 2 items

Copied!

content\_copy

**Task 4. Create, train and evaluate TensorFlow model**

Typically you create one feature for every column in our tabular data. Keras supports feature columns, opening up the ability to represent structured data using standard feature engineering techniques like embedding, bucketizing, and feature crosses. As numeric data can be passed in directly to the ML model, keep the real-valued columns separate from the sparse (or string) columns:

1. Enter and run the following code:

import tensorflow as tf

real = {

colname : tf.feature\_column.numeric\_column(colname)

for colname in

(

'dep\_delay,taxi\_out,distance,dep\_hour,is\_weekday,' +

'dep\_airport\_lat,dep\_airport\_lon,' +

'arr\_airport\_lat,arr\_airport\_lon'

).split(',')

}

sparse = {

'carrier': tf.feature\_column.categorical\_column\_with\_vocabulary\_list('carrier',

vocabulary\_list='AS,VX,F9,UA,US,WN,HA,EV,MQ,DL,OO,B6,NK,AA'.split(',')),

'origin' : tf.feature\_column.categorical\_column\_with\_hash\_bucket('origin', hash\_bucket\_size=1000),

'dest' : tf.feature\_column.categorical\_column\_with\_hash\_bucket('dest', hash\_bucket\_size=1000),

}

Copied!

content\_copy

All these features come directly from the input file (and are provided by any client that wants a prediction for a flight). Input layers map 1:1 to the input features and their types, so rather than repeat the column names, you create an input layer for each of these columns, specifying the right data type (either a float or a string).

1. Enter and run the following code:

inputs = {

colname : tf.keras.layers.Input(name=colname, shape=(), dtype='float32')

for colname in real.keys()

}

inputs.update({

colname : tf.keras.layers.Input(name=colname, shape=(), dtype='string')

for colname in sparse.keys()

})

Copied!

content\_copy

Bucketing

Real-valued columns whose precision is overkill (thus, likely to cause overfitting) can be discretized and made into categorical columns. For example, if you have a column for the age of the aircraft, you might discretize into just three bins—less than 5 years old, 5 to 20 years old, and more than 20 years old. Use the discretization shortcut: you can discretize the latitudes and longitudes and cross the buckets—this results in breaking up the country into grids and yield the grid point into which a specific latitude and longitude falls.

* Enter and run the following code:

latbuckets = np.linspace(20.0, 50.0, NBUCKETS).tolist() # USA

lonbuckets = np.linspace(-120.0, -70.0, NBUCKETS).tolist() # USA

disc = {}

disc.update({

'd\_{}'.format(key) : tf.feature\_column.bucketized\_column(real[key], latbuckets)

for key in ['dep\_airport\_lat', 'arr\_airport\_lat']

})

disc.update({

'd\_{}'.format(key) : tf.feature\_column.bucketized\_column(real[key], lonbuckets)

for key in ['dep\_airport\_lon', 'arr\_airport\_lon']

})

# cross columns that make sense in combination

sparse['dep\_loc'] = tf.feature\_column.crossed\_column(

[disc['d\_dep\_airport\_lat'], disc['d\_dep\_airport\_lon']], NBUCKETS\*NBUCKETS)

sparse['arr\_loc'] = tf.feature\_column.crossed\_column(

[disc['d\_arr\_airport\_lat'], disc['d\_arr\_airport\_lon']], NBUCKETS\*NBUCKETS)

sparse['dep\_arr'] = tf.feature\_column.crossed\_column([sparse['dep\_loc'], sparse['arr\_loc']], NBUCKETS \*\* 4)

# embed all the sparse columns

embed = {

'embed\_{}'.format(colname) : tf.feature\_column.embedding\_column(col, NEMBEDS)

for colname, col in sparse.items()

}

real.update(embed)

# one-hot encode the sparse columns

sparse = {

colname : tf.feature\_column.indicator\_column(col)

for colname, col in sparse.items()

}

if DEVELOP\_MODE:

print(sparse.keys())

print(real.keys())

Copied!

content\_copy

Train and evaluate the model

1. Save the checkpoint:

output\_dir='gs://{}/ch9/trained\_model'.format(BUCKET)

os.environ['OUTDIR'] = output\_dir # needed for deployment

print('Writing trained model to {}'.format(output\_dir))

Copied!

content\_copy

1. Delete the model checkpoints already present in the storage bucket:

!gsutil -m rm -rf $OUTDIR

Copied!

content\_copy

This reports an error stating CommandException: 1 files/objects could not be removed because the model has not yet been saved. The error indicates that there are no files present in the target location. You must be certain that this location is empty before attempting to save the model and this command guarantees that.

1. With the sparse and real feature columns thus enhanced beyond the raw inputs, you can create a wide\_and\_deep\_classifier passing in the linear and deep feature columns separately:

# Build a wide-and-deep model.

def wide\_and\_deep\_classifier(inputs, linear\_feature\_columns, dnn\_feature\_columns, dnn\_hidden\_units):

deep = tf.keras.layers.DenseFeatures(dnn\_feature\_columns, name='deep\_inputs')(inputs)

layers = [int(x) for x in dnn\_hidden\_units.split(',')]

for layerno, numnodes in enumerate(layers):

deep = tf.keras.layers.Dense(numnodes, activation='relu', name='dnn\_{}'.format(layerno+1))(deep)

wide = tf.keras.layers.DenseFeatures(linear\_feature\_columns, name='wide\_inputs')(inputs)

both = tf.keras.layers.concatenate([deep, wide], name='both')

output = tf.keras.layers.Dense(1, activation='sigmoid', name='pred')(both)

model = tf.keras.Model(inputs, output)

model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

return model

model = wide\_and\_deep\_classifier(

inputs,

linear\_feature\_columns = sparse.values(),

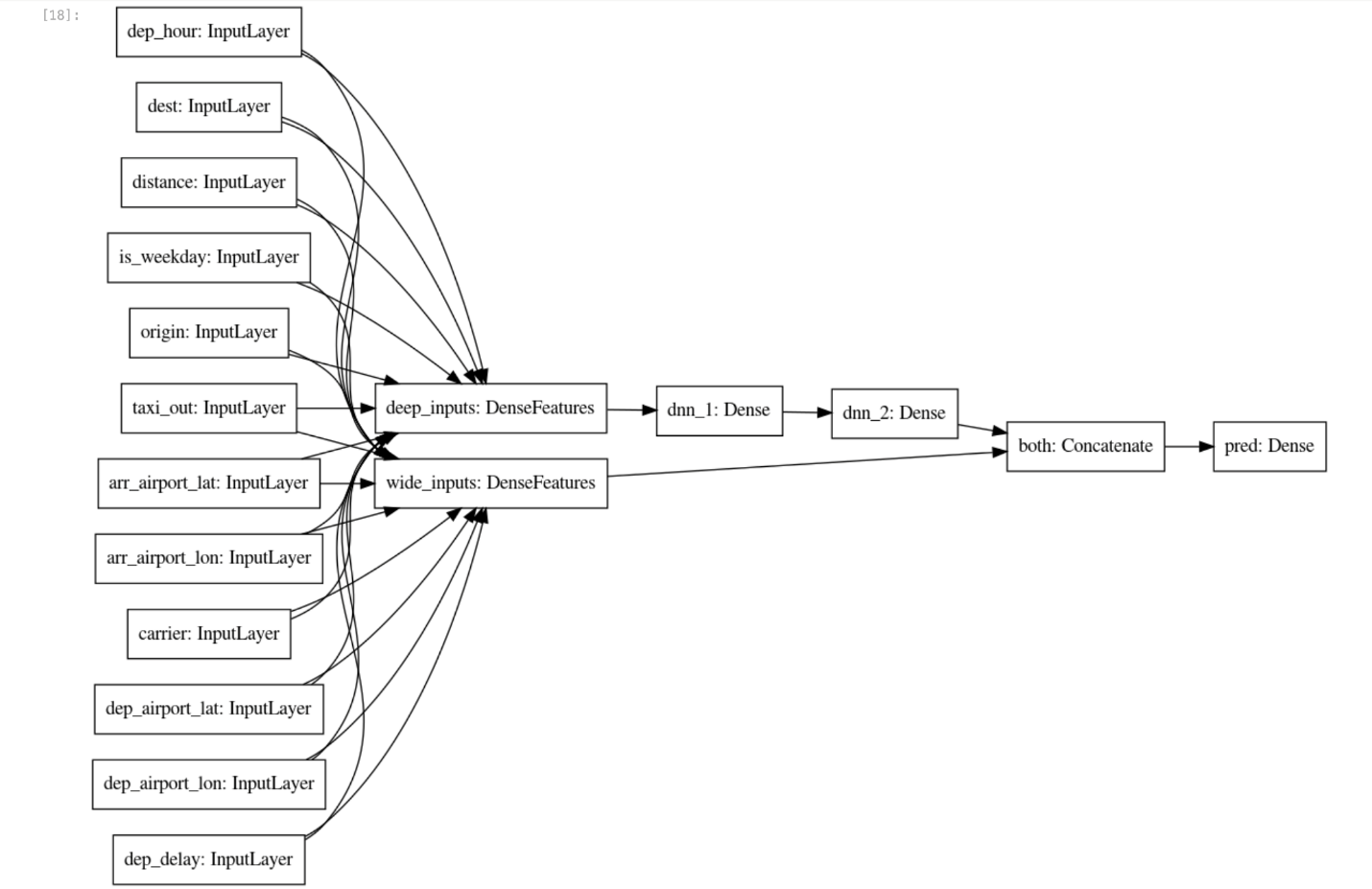
dnn\_feature\_columns = real.values(),

dnn\_hidden\_units = DNN\_HIDDEN\_UNITS)

tf.keras.utils.plot\_model(model, 'flights\_model.png', show\_shapes=False, rankdir='LR')

Copied!

content\_copy



Use train\_dataset for model training and eval\_dataset for model evaluation.

1. Create the model using the following code-blocks:

# training and evaluation dataset

train\_batch\_size = TRAIN\_BATCH\_SIZE

if DEVELOP\_MODE:

eval\_batch\_size = 100

steps\_per\_epoch = 3

epochs = 2

num\_eval\_examples = eval\_batch\_size\*10

else:

eval\_batch\_size = 100

steps\_per\_epoch = NUM\_EXAMPLES // train\_batch\_size

epochs = 10

num\_eval\_examples = eval\_batch\_size \* 100

train\_dataset = read\_dataset(training\_data\_uri, train\_batch\_size)

eval\_dataset = read\_dataset(validation\_data\_uri, eval\_batch\_size, tf.estimator.ModeKeys.EVAL, num\_eval\_examples)

checkpoint\_path = '{}/checkpoints/flights.cpt'.format(output\_dir)

shutil.rmtree(checkpoint\_path, ignore\_errors=True)

cp\_callback = tf.keras.callbacks.ModelCheckpoint(checkpoint\_path,

save\_weights\_only=True,

verbose=1)

history = model.fit(train\_dataset,

validation\_data=eval\_dataset,

epochs=epochs,

steps\_per\_epoch=steps\_per\_epoch,

callbacks=[cp\_callback])

Copied!

content\_copy

1. Visualize the model loss and model accuracy using matplotlib.pyplot:

import matplotlib.pyplot as plt

nrows = 1

ncols = 2

fig = plt.figure(figsize=(10, 5))

for idx, key in enumerate(['loss', 'accuracy']):

ax = fig.add\_subplot(nrows, ncols, idx+1)

plt.plot(history.history[key])

plt.plot(history.history['val\_{}'.format(key)])

plt.title('model {}'.format(key))

plt.ylabel(key)

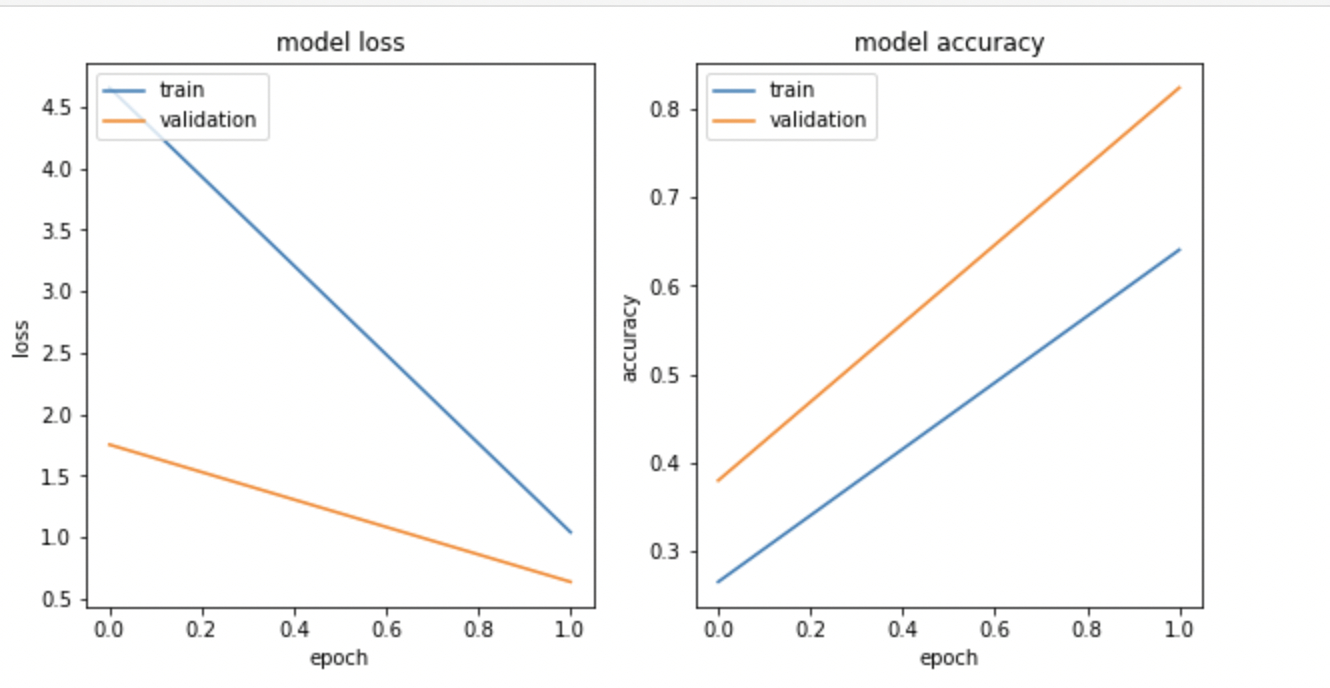
plt.xlabel('epoch')

plt.legend(['train', 'validation'], loc='upper left');

Copied!

content\_copy

The output looks similar to the following:



**Note:**Training loss and the model accuracy graph may not match because you are training on a very small random sample.

Export the trained model

* Save the model artifacts to the Google Cloud Storage bucket:

import time

export\_dir = '{}/export/flights\_{}'.format(output\_dir, time.strftime("%Y%m%d-%H%M%S"))

print('Exporting to {}'.format(export\_dir))

tf.saved\_model.save(model, export\_dir)

Copied!

content\_copy

Assessment Completed!

Create the TensorFlow Model

Check my progress

*Assessment Completed!*

**Task 5. Deploy flights model to Vertex AI**

Vertex AI provides a fully managed, autoscaling, serverless environment for Machine Learning models. You get the benefits of paying for any compute resources (such as CPUs or GPUs) only when you are using them. Because the models are containerized, dependency management is taken care of. The Endpoints take care of traffic splits, allowing you to do A/B testing in a convenient way.

The benefits go beyond not having to manage infrastructure. Once you deploy the model to Vertex AI, you get a lot of neat capabilities without any additional code — explainability, drift detection, monitoring, etc.

1. Create the model endpoint flights using the following code cell and delete any existing models with the same name:

%%bash

# note TF\_VERSION and ENDPOINT\_NAME set in 1st cell

# TF\_VERSION=2-6

# ENDPOINT\_NAME=flights

TIMESTAMP=$(date +%Y%m%d-%H%M%S)

MODEL\_NAME=${ENDPOINT\_NAME}-${TIMESTAMP}

EXPORT\_PATH=$(gsutil ls ${OUTDIR}/export | tail -1)

echo $EXPORT\_PATH

# create the model endpoint for deploying the model

if [[ $(gcloud beta ai endpoints list --region=$REGION \

--format='value(DISPLAY\_NAME)' --filter=display\_name=${ENDPOINT\_NAME}) ]]; then

echo "Endpoint for $MODEL\_NAME already exists"

else

echo "Creating Endpoint for $MODEL\_NAME"

gcloud beta ai endpoints create --region=${REGION} --display-name=${ENDPOINT\_NAME}

fi

ENDPOINT\_ID=$(gcloud beta ai endpoints list --region=$REGION \

--format='value(ENDPOINT\_ID)' --filter=display\_name=${ENDPOINT\_NAME})

echo "ENDPOINT\_ID=$ENDPOINT\_ID"

# delete any existing models with this name

for MODEL\_ID in $(gcloud beta ai models list --region=$REGION --format='value(MODEL\_ID)' --filter=display\_name=${MODEL\_NAME}); do

echo "Deleting existing $MODEL\_NAME ... $MODEL\_ID "

gcloud ai models delete --region=$REGION $MODEL\_ID

done

# create the model using the parameters docker conatiner image and artifact uri

gcloud beta ai models upload --region=$REGION --display-name=$MODEL\_NAME \

--container-image-uri=us-docker.pkg.dev/vertex-ai/prediction/tf2-cpu.${TF\_VERSION}:latest \

--artifact-uri=$EXPORT\_PATH

MODEL\_ID=$(gcloud beta ai models list --region=$REGION --format='value(MODEL\_ID)' --filter=display\_name=${MODEL\_NAME})

echo "MODEL\_ID=$MODEL\_ID"

# deploy the model to the endpoint

gcloud beta ai endpoints deploy-model $ENDPOINT\_ID \

--region=$REGION \

--model=$MODEL\_ID \

--display-name=$MODEL\_NAME \

--machine-type=n1-standard-2 \

--min-replica-count=1 \

--max-replica-count=1 \

--traffic-split=0=100

Copied!

content\_copy

**Note:**An error can occasionally occur around 5 minutes into this process. If you encounter a model building error, like the service account doesn't have sufficient permissions for writing objects to the Google Cloud Storage bucket, **try to run the code cell again**. Also, enable the Vertex AI API, if not enabled.**Note:**It will take around 15-20 minutes to create the model, model endpoint, and deploy the model to the endpoint. If you are unable to access the generated endpoint link, please ignore it. To see the progress in your Cloud Console, click on **Navigation menu > Vertex AI > Endpoints**.

Please follow the lab instructions to deploy the flights model on Vertex AI.

Deploy flights model to Vertex AI

Check my progress

*Please follow the lab instructions to deploy the flights model on Vertex AI.*

1. Create a test input file example\_input.json using the following code:

%%writefile example\_input.json

{"instances": [

{"dep\_hour": 2, "is\_weekday": 1, "dep\_delay": 40, "taxi\_out": 17, "distance": 41, "carrier": "AS", "dep\_airport\_lat": 58.42527778, "dep\_airport\_lon": -135.7075, "arr\_airport\_lat": 58.35472222, "arr\_airport\_lon": -134.57472222, "origin": "GST", "dest": "JNU"},

{"dep\_hour": 22, "is\_weekday": 0, "dep\_delay": -7, "taxi\_out": 7, "distance": 201, "carrier": "HA", "dep\_airport\_lat": 21.97611111, "dep\_airport\_lon": -159.33888889, "arr\_airport\_lat": 20.89861111, "arr\_airport\_lon": -156.43055556, "origin": "LIH", "dest": "OGG"}

]}

Copied!

content\_copy

1. Make a prediction from the model endpoint. Here you have input data in a JSON file called example\_input.json:

%%bash

ENDPOINT\_ID=$(gcloud beta ai endpoints list --region=$REGION \

--format='value(ENDPOINT\_ID)' --filter=display\_name=${ENDPOINT\_NAME})

echo $ENDPOINT\_ID

gcloud beta ai endpoints predict $ENDPOINT\_ID --region=$REGION --json-request=example\_input.json

Copied!

content\_copy

Here’s how client programs can invoke the model that you deployed.

Assume that they have the input data in a JSON file called example\_input.json.

1. Now, send an HTTP POST request and you will get the result back as JSON:

%%bash

PROJECT=$(gcloud config get-value project)

ENDPOINT\_ID=$(gcloud beta ai endpoints list --region=$REGION \

--format='value(ENDPOINT\_ID)' --filter=display\_name=${ENDPOINT\_NAME})

curl -X POST \

-H "Authorization: Bearer "$(gcloud auth application-default print-access-token) \

-H "Content-Type: application/json; charset=utf-8" \

-d @example\_input.json \

"https://${REGION}-aiplatform.googleapis.com/v1/projects/${PROJECT}/locations/${REGION}/endpoints/${ENDPOINT\_ID}:predict"

Copied!

content\_copy

**Task 6. Model explainability**

Model explainability is one of the most important problems in machine learning. It's a broad concept of analyzing and understanding the results provided by machine learning models. Explainability in machine learning means you can explain what happens in your model from input to output. It makes models transparent and solves the black box problem. Explainable AI (XAI) is the more formal way to describe this.

1. Run the following code:

%%bash

model\_dir=$(gsutil ls ${OUTDIR}/export | tail -1)

echo $model\_dir

saved\_model\_cli show --tag\_set serve --signature\_def serving\_default --dir $model\_dir

Copied!

content\_copy

1. Create a JSON file explanation-metadata.json that contains the metadata describing the Model's input and output for explanation. Here, you use sampled-shapley method used for explanation:

cols = ('dep\_delay,taxi\_out,distance,dep\_hour,is\_weekday,' +

'dep\_airport\_lat,dep\_airport\_lon,' +

'arr\_airport\_lat,arr\_airport\_lon,' +

'carrier,origin,dest')

inputs = {x: {"inputTensorName": "{}".format(x)}

for x in cols.split(',')}

expl = {

"inputs": inputs,

"outputs": {

"pred": {

"outputTensorName": "pred"

}

}

}

print(expl)

with open('explanation-metadata.json', 'w') as ofp:

json.dump(expl, ofp, indent=2)

Copied!

content\_copy

1. View the explanation-metadata.json file using the cat command:

!cat explanation-metadata.json

Copied!

content\_copy



Create and deploy another model flights\_xai to Vertex AI

* Create the model endpoint flights\_xai, upload the model, and deploy it at the model endpoint using the following code:

%%bash

# note TF\_VERSION set in 1st cell, but ENDPOINT\_NAME is being changed

# TF\_VERSION=2-6

ENDPOINT\_NAME=flights\_xai

TIMESTAMP=$(date +%Y%m%d-%H%M%S)

MODEL\_NAME=${ENDPOINT\_NAME}-${TIMESTAMP}

EXPORT\_PATH=$(gsutil ls ${OUTDIR}/export | tail -1)

echo $EXPORT\_PATH

# create the model endpoint for deploying the model

if [[ $(gcloud beta ai endpoints list --region=$REGION \

--format='value(DISPLAY\_NAME)' --filter=display\_name=${ENDPOINT\_NAME}) ]]; then

echo "Endpoint for $MODEL\_NAME already exists"

else

# create model endpoint

echo "Creating Endpoint for $MODEL\_NAME"

gcloud beta ai endpoints create --region=${REGION} --display-name=${ENDPOINT\_NAME}

fi

ENDPOINT\_ID=$(gcloud beta ai endpoints list --region=$REGION \

--format='value(ENDPOINT\_ID)' --filter=display\_name=${ENDPOINT\_NAME})

echo "ENDPOINT\_ID=$ENDPOINT\_ID"

# delete any existing models with this name

for MODEL\_ID in $(gcloud beta ai models list --region=$REGION --format='value(MODEL\_ID)' --filter=display\_name=${MODEL\_NAME}); do

echo "Deleting existing $MODEL\_NAME ... $MODEL\_ID "

gcloud ai models delete --region=$REGION $MODEL\_ID

done

# upload the model using the parameters docker conatiner image, artifact URI, explanation method,

# explanation path count and explanation metadata JSON file `explanation-metadata.json`.

# Here, you keep number of feature permutations to `10` when approximating the Shapley values for explanation.

gcloud beta ai models upload --region=$REGION --display-name=$MODEL\_NAME \

--container-image-uri=us-docker.pkg.dev/vertex-ai/prediction/tf2-cpu.${TF\_VERSION}:latest \

--artifact-uri=$EXPORT\_PATH \

--explanation-method=sampled-shapley --explanation-path-count=10 --explanation-metadata-file=explanation-metadata.json

MODEL\_ID=$(gcloud beta ai models list --region=$REGION --format='value(MODEL\_ID)' --filter=display\_name=${MODEL\_NAME})

echo "MODEL\_ID=$MODEL\_ID"

# deploy the model to the endpoint

gcloud beta ai endpoints deploy-model $ENDPOINT\_ID \

--region=$REGION \

--model=$MODEL\_ID \

--display-name=$MODEL\_NAME \

--machine-type=n1-standard-2 \

--min-replica-count=1 \

--max-replica-count=1 \

--traffic-split=0=100

Copied!

content\_copy

**Note:**It will take around 15-20 minutes to create the model, model endpoint, and deploy the model to the endpoint. If you are unable to access the generated endpoint link, please ignore it. To see the progress in your Cloud Console, click on **Navigation menu > Vertex AI > Endpoints**.

Deploy flights\_xai model to Vertex AI

Check my progress

**Task 7. Invoke the deployed model**

Here’s how client programs can invoke the model you deployed. Assume that they have the input data in a JSON file called example\_input.json. Now, send an HTTP POST request and you will get the result back as JSON.

* Run the following code:

%%bash

PROJECT=$(gcloud config get-value project)

ENDPOINT\_NAME=flights\_xai

ENDPOINT\_ID=$(gcloud beta ai endpoints list --region=$REGION \

--format='value(ENDPOINT\_ID)' --filter=display\_name=${ENDPOINT\_NAME})

curl -X POST \

-H "Authorization: Bearer "$(gcloud auth application-default print-access-token) \

-H "Content-Type: application/json; charset=utf-8" \

-d @example\_input.json \

"https://${REGION}-aiplatform.googleapis.com/v1/projects/${PROJECT}/locations/${REGION}/endpoints/${ENDPOINT\_ID}:explain"

Copied!

content\_copy

**Congratulations!**

You explored the machine learning approach by using the TensorFlow library to carry out GPU-accelerated training. TensorFlow also allows a computer scientist to go as low-level as they need to, which is why so many machine learning research innovations are implemented in TensorFlow. TensorFlow allows machine learning practitioners to use innovative machine learning research soon after it is published, rather than wait for a reimplementation in some other framework. Finally, TensorFlow is portable across a wide variety of hardware platforms, therefore using TensorFlow allows you to easily deploy the model into your data pipelines regardless of where they are run.

You trained a logistic regression model on all of the input values and learned that the model was unable to effectively use the new features like airport locations.

Finish your quest

This self-paced lab is part of the [Data Science on Google Cloud: Machine Learning](https://cloudskillsboost.google/quests/50) quest. A quest is a series of related labs that form a learning path. Completing this quest earns you a badge to recognize your achievement. You can make your badge or badges public and link to them in your online resume or social media account. Enroll in any quest that contains this lab and get immediate completion credit. See the [Google Cloud Skills Boost catalog](http://cloudskillsboost.google/catalog) to see all available quests

Take your next lab

Continue your Quest with [MLOps with Vertex AI](https://www.cloudskillsboost.google/catalog_lab/1310), or [Real Time Machine Learning with Cloud Dataflow and Vertex AI](https://www.cloudskillsboost.google/catalog_lab/1311).

Next Steps / Learn more

* [Data Science on the Google Cloud Platform, 2nd Edition: O'Reilly Media, Inc](https://www.oreilly.com/library/view/data-science-on/9781098118945/).

Google Cloud training and certification

...helps you make the most of Google Cloud technologies. [Our classes](https://cloud.google.com/training/courses) include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and virtual options to suit your busy schedule. [Certifications](https://cloud.google.com/certification/) help you validate and prove your skill and expertise in Google Cloud technologies.

**Manual Last Updated April 19, 2023**

**Lab Last Tested April 19, 2023**

Copyright 2023 Google LLC All rights reserved. Google and the Google logo are trademarks of Google LLC. All other company and product names may be trademarks of the respective companies with which they are associated.