Ungraded Lab (Optional): Build, train, and deploy an XGBoost model on Cloud AI Platform

1: Overview

In this lab, you will walk through a complete ML workflow on GCP. From a Cloud AI Platform Notebooks environment, you'll ingest data from a BigQuery public dataset, build and train an XGBoost model, and deploy the model to AI Platform for prediction.

You'll learn how to:

```
Ingest and analyze a BigQuery dataset in AI Platform Notebooks
Build an XGBoost model
Deploy the XGBoost model to AI Platform and get predictions
The total cost to run this lab on Google Cloud is about $1.
```

Tip: It is best to have at least two windows open when going through the instructions in this tutorial: at least one for navigating to the different parts of GCP (e.g. Storage, BigQuery, AI Platform Models) and one for the AI Platform Jupyter Notebook you will open in Step 2 below.

2 : Setup Your Environment

You'll need a Google Cloud Platform project to run this exercise. If you just enabled a GCP free trial, you should already have a project called 'My First Project'. If not, you can follow the instructions here to create a project.

Step 1: Enable the Cloud AI Platform Models API

Navigate to the AI Platform Models section of your Cloud Console and click Enable if it isn't already enabled.
images/models_api

Step 2: Enable the Compute Engine API

Navigate to Compute Engine and select Enable if it isn't already enabled. You'll need this to create your notebook instance. (tip: After clicking Enable and it doesn't automatically refresh, you can just manually refresh the page after a minute to see if the API has been enabled. It should show "API Enabled".

Step 3: Create an AI Platform Notebooks instance

Navigate to AI Platform Notebooks section of your Cloud Console and click New Instance. Then select the latest Python instance type:

Step 4: Install XGBoost

Once your JupyterLab instance has opened, you'll need to add the XGBoost package.

To do this, select Terminal from the launcher:
pip3 install xgboost==1.4.2
From there, you can open a Python 3 Notebook instance. You're ready to get started in your notebook!

Step 5: Import Python packages

For the rest of this codelab, run all the code snippets from your Jupyter notebook.

In the first cell of your notebook, add the following imports and run the cell. You can run it by pressing the right arrow button in the top menu or pressing command-enter:

In [1]:
```
import pandas as pd
import xgboost as xgb
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from google.cloud import bigquery
```

You'll be using the Python client library for BigQuery to download the data into a Pandas DataFrame. The original dataset is 21GB and contains 123M rows. To keep things simple we'll only be using 10,000 rows from the dataset.

Construct the query and preview the resulting DataFrame with the following code. Here we're getting 4 features from the original dataset, along with baby weight (the thing our model will predict). The dataset goes back many years but for this model we'll use only data from after 2000:

3 : Exploring the BigQuery dataset

In [2]:
```
query="""
SELECT
  weight_pounds,
  is_male,
  mother_age,
  plurality,
  gestation_weeks
FROM
  publicdata.samples.natality
WHERE year > 2000
LIMIT 10000
"""
df = bigquery.Client().query(query).to_dataframe()
df.head()
```

Out[2]:

	weight_pounds	is_male	mother_age	plurality	gestation_weeks
0	7.063611	True	32	1	37.0
1	4.687028	True	30	3	33.0
2	7.561856	True	20	1	39.0
3	7.561856	True	31	1	37.0
4	7.312733	True	32	1	40.0

If you get a 403 Forbidden error, it means you will need to enable the BigQuery API for your account. Search for BigQuery in the Search Bar and click Enable API. Kindly wait for it to be enabled before re-running the command above.

This shows the mean, standard deviation, minimum, and other metrics for our numeric columns. Finally, let's get some data on our boolean column indicating the baby's gender. We can do this with Pandas' value_counts method:

In [4]:
```
df['is_male'].value_counts()
```

Out[4]:
```
True     5096
False    4904
Name: is_male, dtype: int64
```

4 : Prepare Data for Training

In this section, we'll divide the data into train and test sets to prepare it for training our model.

Step 1: Extract the label column

First drop rows with null values from the dataset and shuffle the data:

In [6]:
```
df = df.dropna()
df = shuffle(df, random_state=2)
```

Next, extract the label column into a separate variable and create a DataFrame with only our features:

In [7]:
```
labels = df['weight_pounds']
data = df.drop(columns=['weight_pounds'])
```

Now if you preview our dataset by running data.head(), you should see the four features we'll be using for training.

Step 2: Convert categorical features to integers

Since XGBoost requires all data to be numeric, we'll need to change how we're representing the data in the is_male column, which is currently True / False strings. We can do that simply by changing the type of that column:

In [8]:
```
data['is_male'] = data['is_male'].astype(int)
```

Step 3: Split data into train and test sets

We'll use Scikit Learn's train_test_split utility which we imported at the beginning of the notebook to split our data into train and test sets:

In [9]:
```
x,y = data,labels
x_train,x_test,y_train,y_test = train_test_split(x,y)
```

Now we're ready to build and train our model!

5 : A quick XGBoost primer

```
XGBoost is a machine learning framework that uses decision trees and gradient
boosting to build predictive models. It works by ensembling multiple decision trees
together based on the score associated with different leaf nodes in a tree.
```

The diagram below is a simplified visualization of an ensemble tree network for a model that evaluates whether or not someone will like a specific computer game (this is from the XGBoost docs):

Why are we using XGBoost for this model? While traditional neural networks have been shown to perform best on unstructured data like images and text, decision trees often perform extremely well on structured data like the birth weight dataset we'll be using.

6 : Build, train, and evaluate an XGBoost model

Step 1:

Define and train the XGBoost model Creating a model in XGBoost is simple. We'll use the XGBRegressor class to create the model, and just need to pass the right objective parameter for our specific task. Here we're using a regression model since we're predicting a numerical value (baby's weight). If we were instead bucketing our data to determine if a baby weighed more or less than 6 pounds, we'd use a classification model.

In this case we'll use reg:squarederror as our model's objective.

The following code will create an XGBoost model:

In [10]:
```
model = xgb.XGBRegressor(
    objective='reg:squarederror'
)
```

you can train the model with one line of code, calling the fit() method and passing it the training data and labels.

In [11]:
```
model.fit(x_train, y_train)
```

Out[11]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

Step 2: Evaluate your model on test data

We can now use our trained model to generate predictions on our test data with the predict() function:

In [12]:
```
y_pred = model.predict(x_test)
```

Let's see how the model performed on the first 20 values from our test set. Below we'll print the predicted baby weight along with the actual baby weight for each test example:

In [13]:
```
for i in range(20):
    print('Predicted weight: ', y_pred[i])
    print('Actual weight: ', y_test.iloc[i])
    print()
```

```
Predicted weight:  7.9053507
Actual weight:  9.43798943622

Predicted weight:  7.5190206
Actual weight:  8.56275425608

Predicted weight:  6.360227
Actual weight:  7.936641432

Predicted weight:  7.8039613
Actual weight:  7.87491199864

Predicted weight:  7.218525
Actual weight:  6.1883756943399995

Predicted weight:  7.791048
Actual weight:  9.62538235892

Predicted weight:  7.593407
Actual weight:  10.37495404972

Predicted weight:  7.4427757
Actual weight:  6.9996768185

Predicted weight:  7.440815
Actual weight:  7.7492485093

Predicted weight:  7.926063
Actual weight:  7.1319541757

Predicted weight:  7.584514
Actual weight:  7.68751907594

Predicted weight:  7.761119
Actual weight:  7.50012615324

Predicted weight:  7.858591
Actual weight:  5.1367707046

Predicted weight:  2.1454005
Actual weight:  2.68743497378

Predicted weight:  7.747493
Actual weight:  6.935742762519999

Predicted weight:  6.832648
```

```
Actual weight:  7.30391474006

Predicted weight:  4.3678555
Actual weight:  7.06361087448

Predicted weight:  6.3810005
Actual weight:  7.3744626639

Predicted weight:  7.780418
Actual weight:  6.6248909731

Predicted weight:  8.02287
Actual weight:  7.71617917
```

Step 3:

Save your model In order to deploy the model, run the following code to save it to a local file:

In [14]:
```
model.save_model('model.bst')
```

7 : Deploy model to Cloud AI Platform

We've got our model working locally, but it would be nice if we could make predictions on it from anywhere (not just this notebook!). In this step we'll deploy it to the cloud.

Step 1: Create a Cloud Storage bucket for our model

Let's first define some environment variables that we'll be using throughout the rest of the tutorial. Fill in the values below with your PROJECT ID, the name of the cloud storage bucket you'd like to create (must be globally unique, you can use the project id as well), and the version name for the first version of your model.

Tip: You can get the Project ID as shown by Laurence in the screencast or by running this command in a cell: !gcloud config list project --format "value(core.project)". You can use the result to fill in below:

In [15]:
```
# Update these to your own GCP project, model, and version names
GCP_PROJECT = 'coursera-mlops-c4w1l3'
MODEL_BUCKET = 'gs://coursera-mlops-c4w1l3'
VERSION_NAME = 'v1'
MODEL_NAME = 'baby_weight'
```

Now we're ready to create a storage bucket to store our XGBoost model file. We'll point Cloud AI Platform at this file when we deploy.

Run this gsutil command from within your notebook to create a bucket:

In [16]:
```
!gsutil mb $MODEL_BUCKET
```

```
Creating gs://coursera-mlops-c4w1l3/...
```

Step 2: Copy the model file to Cloud Storage

Next, we'll copy our XGBoost saved model file to Cloud Storage. Run the following gsutil command:

In [17]:
```
!gsutil cp ./model.bst $MODEL_BUCKET
```

```
Copying file://./model.bst [Content-Type=application/octet-stream]...
/ [1 files][294.1 KiB/294.1 KiB]
Operation completed over 1 objects/294.1 KiB.
```

If you get errors about creating buckets, you may need to enable the Cloud Storage API before retrying the command above. Just search for Cloud Storage using the Search Bar then click Enable API.

Head over to the storage browser in your Cloud Console to confirm the file has been copied:

Step 3: Create and deploy the model

The following ai-platform gcloud command will create a new model in your project:

In [18]:
```
!gcloud ai-platform models create $MODEL_NAME --region=us-central1
```

```
Using endpoint [https://us-central1-ml.googleapis.com/]
Created ai platform model [projects/coursera-mlops-c4w1l3/models/baby_weight].
```

Now it's time to deploy the model. We can do that with this gcloud command:

In [19]:
```
!gcloud ai-platform versions create $VERSION_NAME \
--model=$MODEL_NAME \
--framework='XGBOOST' \
--runtime-version=2.5 \
--origin=$MODEL_BUCKET \
--python-version=3.7 \
--project=$GCP_PROJECT \
--region=us-central1
```

```
Using endpoint [https://us-central1-ml.googleapis.com/]
Creating version (this might take a few minutes)......done.
```

While this is running, check the models section of your AI Platform console. You should see your new version deploying there. When the deploy completes successfully you'll see a green check mark where the loading spinner is. The deployment can take up to 5 minutes.

Step 4: Test the deployed model

To make sure your deployed model is working, test it out using gcloud to make a prediction. First, save a JSON file with two examples from our test set:

In [20]:
```
%%writefile predictions.json
[0.0, 33.0, 1.0, 27.0]
[1.0, 26.0, 1.0, 40.0]
```

```
Writing predictions.json
```

Test your model by saving the output of the following gcloud command to a variable and printing it:

In [21]:
```
prediction = !gcloud ai-platform predict --model=$MODEL_NAME --json-instances=predictions.js
print(prediction.s)
```

```
Using endpoint [https://us-central1-ml.googleapis.com/] [2.051780939102173, 7.87857484817504
9]
```

You should see your model's prediction in the output. The actual baby weight for these two examples is around 2 and 8 pounds respectively (results may differ slightly because we shuffled our dataset).

8 : Cleanup

If you'd like to continue using this notebook, it is recommended that you turn it off when not in use. From the Notebooks UI in your Cloud Console, select the notebook and then select Stop: images/cleanup

If you'd like to delete all resources you've created in this lab, simply delete the notebook instance instead of stopping it.

Using the Navigation menu in your Cloud Console, browse to Cloud Storage and delete both buckets you created to store your model assets. Similarly, you can also go to the dashboard of AI Platform -> Models to delete the model manually.

In [ ]:
```
```