time_series_analysis_gcp_vertex_ai_cloud_training_job

December 15, 2021

0.0.1 Training on cloud using Vertex AI SDK

0.0.2 Import libraries

[1]:

```
import datetime
import os
import time

import numpy as np
import pandas as pd
import tensorflow as tf

from google.cloud import aiplatform, storage
from google.cloud.aiplatform import gapic as aip
from sklearn.preprocessing import StandardScaler
```

[2]:

```
# Check the TensorFlow version installed

tf.__version__
```

[2]:

```
'2.7.0'
```

[3]:

```
# initialising contansts
PROJECT = 'vertex-ai-projects'
BUCKET = 'vertex-ai-projects'
REGION = 'us-west1'
BUCKET_URI = 'gs://' + BUCKET
```

Setting up the vertex ai environment with region name and project to use.

[4]:

```
# starting the Vertex AI SDK

aiplatform.init(project=PROJECT, location=REGION, staging_bucket=BUCKET)
```

[5]:

```
# Dataset parameters

# target variable column
target_col = 'total_rides'
```

```
# Date field
ts_col = 'service_date'
```

[6]:

```
# Parameters for Model

# Daily frequency
freq = 'D'

# Lookback window
n_input_steps = 30

# How many steps to predict forward
n_output_steps = 7

# Periodicity month wise
n_seasons = 7

# % Split between train/test data for our dataset
train_split = 0.8

# Epochs
epochs = 1000

# this parameters stops the training if the loss stops decreasing after no. of␣
 ↪steps.
patience = 5

# lstm units
lstm_units = 64
input_layer_name = 'lstm_input'

# Model name
MODEL_NAME = 'cta_ridership_lstm_model'
```

[7]:

```
# Building a cloud storage bucket on GCP

# initiating a storage session client
storage_client = storage.Client()

# to create bucket if it doesn't exist
try:
    bucket = storage_client.get_bucket(BUCKET)
    print('Bucket already exists')
except:
    bucket = storage_client.create_bucket(BUCKET)
    print('Created bucket: ' + BUCKET)
```

```
Bucket already exists
```

0.1 Ingest and view the dataset

[8]:

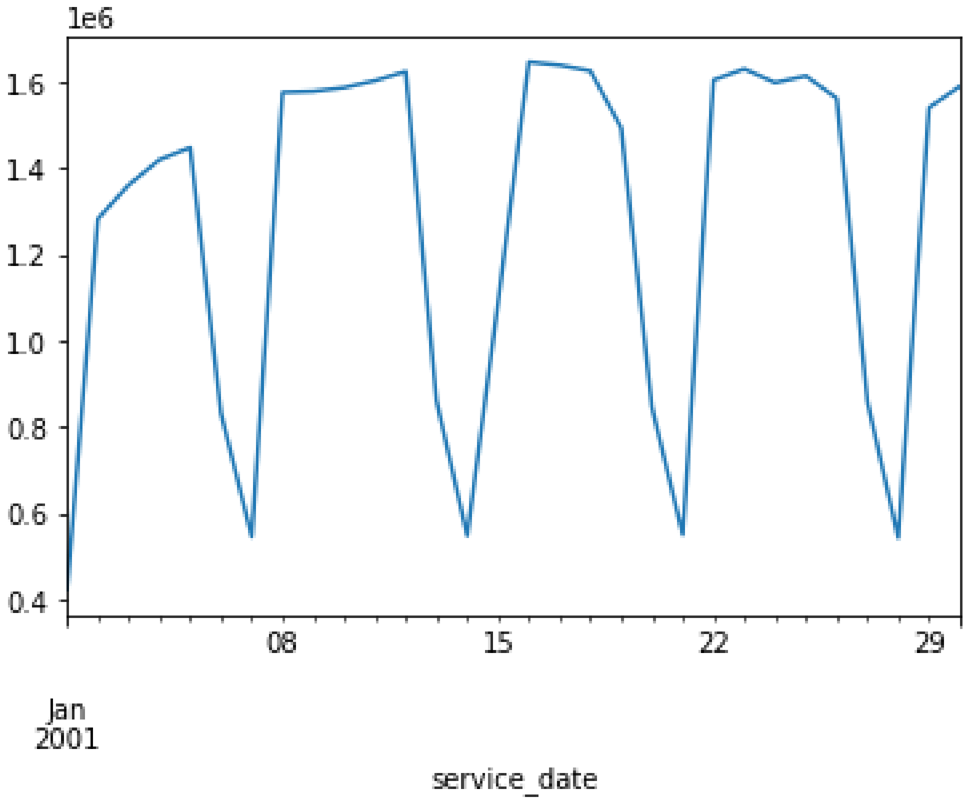
```
import os
processed_file = 'cta_ridership.csv'

# loading data if it doesnt exists in the sys
if os.path.exists(processed_file):
    input_file = processed_file
else:
    input_file = f'data/{processed_file}'


df = pd.read_csv(input_file, index_col=ts_col, parse_dates=True)
df.index.freq = freq

df.head()
```

[8]:

```
              total_rides
service_date
2001-01-01         423647
2001-01-02        1282779
2001-01-03        1361355
2001-01-04        1420032
2001-01-05        1448343
```

[9]:

```
# Plot 30 days from ridership data
_ = df[target_col][:30].plot()
```



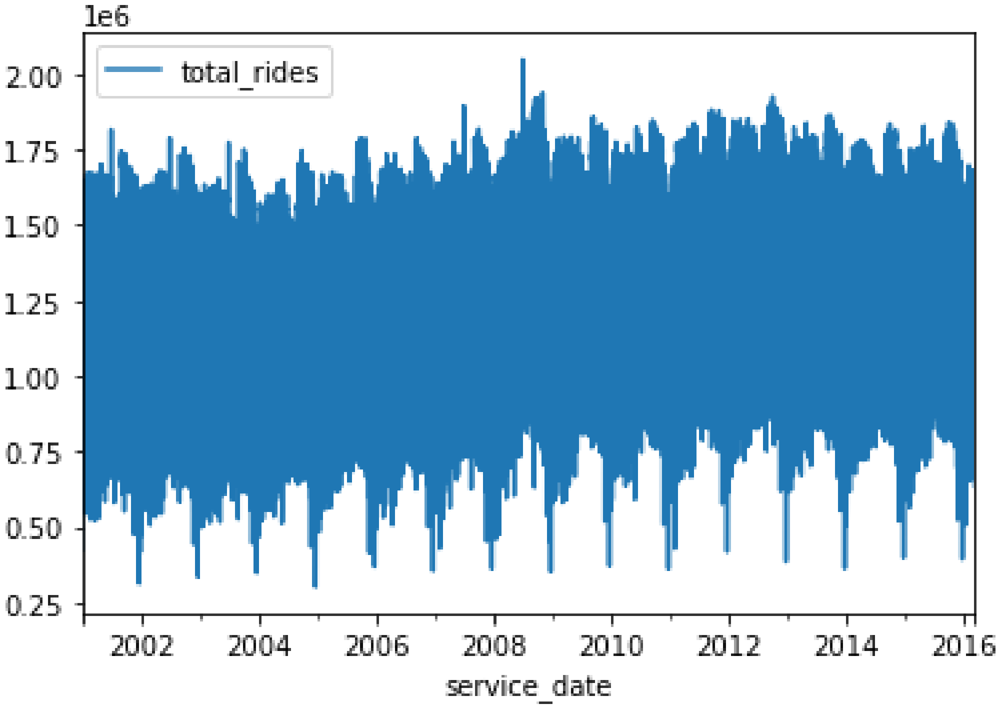
```
[10]: # no. of features in the dataset
      n_features = len(df.columns)

      # Index of target column
      target_col_num = df.columns.get_loc(target_col)
```

```
[11]: # Splitting the data into test/train
      size = int(len(df) * train_split)

      # Creating test/train splits
      df_train, df_test = df[0:size].copy(deep=True), df[size:len(df)].copy(deep=True)

      df_train.head()
```

```
[11]:               total_rides
      service_date
      2001-01-01         423647
      2001-01-02        1282779
      2001-01-03        1361355
      2001-01-04        1420032
      2001-01-05        1448343
```

[12]:

```
_ = df_train.plot()
```



Preprocessing the dataset for better Results

[13]:

```
# Scaling and Transforming the dataset functions
feature_scaler = StandardScaler()
target_scaler = StandardScaler()

# scaling the input features and target column
# Using seaprate scaler for target column
def scale(df,
          fit=True,
          target_col=target_col,
          feature_scaler=feature_scaler,
          target_scaler=target_scaler):
    # reshaping the target column values
    target = df[target_col].values.reshape(-1, 1)
    if fit:
        target_scaler.fit(target)
    target_scaled = target_scaler.transform(target)

    features = df.loc[:, df.columns != target_col].values

    if features.shape[1]:
        if fit:
            feature_scaler.fit(features)
        features_scaled = feature_scaler.transform(features)

        # aggregating the feature and target column after scaling and␣
 ↪transforming
        df_scaled = pd.DataFrame(features_scaled)
        target_col_num = df.columns.get_loc(target_col)
        df_scaled.insert(target_col_num, target_col, target_scaled)
        df_scaled.columns = df.columns

    else:
        df_scaled = pd.DataFrame(target_scaled, columns=df.columns)

    return df_scaled

# transofrming the scaled values back to their inital values
def inverse_scale(data, target_scaler=target_scaler):

    df = pd.DataFrame()
    data_scaled = np.empty([data.shape[1], data.shape[0]])
    for i in range(data.shape[1]):
        # using inverse_transofrm function
        data_scaled[i] = target_scaler.inverse_transform([data[:,i]])
    return data_scaled.transpose()

df_train_scaled=scale(df_train)
df_test_scaled=scale(df_test, False)
```

[14]:

```
df_train_scaled.head()
```

[14]:

```
   total_rides
0    -2.442494
1    -0.262138
2    -0.062724
3     0.086190
4     0.158040
```

0.1.1 Create sequences of data

[15]:

```
def reframe(data, n_input_steps = n_input_steps, n_output_steps =␣
 ↪n_output_steps, target_col = target_col):

    # getting the index of target column
    target_col_num = data.columns.get_loc(target_col)

    df = pd.DataFrame(data)
    cols=list()
    for i in range(n_input_steps, 0, -1):
        cols.append(df.shift(i))
    for i in range(0, n_output_steps):
        cols.append(df.shift(-i))

    # Concatenating values
    df = pd.concat(cols, axis=1)

    # dropping all the NaN values
    df.dropna(inplace=True)

    # Splitting the data into feature and label columns
    n_feature_cols = n_input_steps * n_features
    features = df.iloc[:,0:n_feature_cols]
    target_cols = [i for i in range(n_feature_cols + target_col_num,␣
 ↪n_feature_cols + n_output_steps * n_features, n_features)]
    targets = df.iloc[:,target_cols]

    return (features, targets)

# finally storing the reframed dataset
X_train_reframed, y_train_reframed = reframe(df_train_scaled)
X_test_reframed, y_test_reframed = reframe(df_test_scaled)
```

Reshaping the dataset so that input values to match output

0.1.2 Prepare test data

[16]:

```
X_train = X_train_reframed.values.reshape(-1, n_input_steps, n_features)
X_test = X_test_reframed.values.reshape(-1, n_input_steps, n_features)
y_train = y_train_reframed.values.reshape(-1, n_output_steps, 1)
y_test = y_test_reframed.values.reshape(-1, n_output_steps, 1)
```

[17]:

```
# directories in bucket

TRAINER_DIR = 'trainer'
EXPORT_DIR = 'tf_export'
```

[18]:

```
# creating the trainer directory

!mkdir $TRAINER_DIR
```

[19]:

```
# Copying the arrays to npy files

np.save(TRAINER_DIR + '/x_train.npy', X_train)
np.save(TRAINER_DIR + '/x_test.npy', X_test)
np.save(TRAINER_DIR + '/y_train.npy', y_train)
np.save(TRAINER_DIR + '/y_test.npy', y_test)
```

0.1.3 model code

[20]:

```
model_template = f"""import argparse
import numpy as np
import os
import tempfile

from google.cloud import storage
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, LSTM
from tensorflow.keras.callbacks import EarlyStopping

n_features = {n_features}

# lookback window
n_input_steps = {n_input_steps}

# How many steps to predict forward
n_output_steps = {n_output_steps}

epochs = {epochs}
patience = {patience}

def download_blob(bucket_name, source_blob_name, destination_file_name):
    storage_client = storage.Client()

    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(source_blob_name)
    blob.download_to_filename(destination_file_name)

    print("Blob " + source_blob_name + " downloaded to " +␣
 ↪destination_file_name + ".")

def extract_bucket_and_prefix_from_gcs_path(gcs_path: str):
    if gcs_path.startswith("gs://"):
        gcs_path = gcs_path[5:]
    if gcs_path.endswith("/"):
        gcs_path = gcs_path[:-1]

    gcs_parts = gcs_path.split("/", 1)
    gcs_bucket = gcs_parts[0]
    gcs_blob_prefix = None if len(gcs_parts) == 1 else gcs_parts[1]
    return (gcs_bucket, gcs_blob_prefix)

def get_args():
    parser = argparse.ArgumentParser()
    parser.add_argument(
        '--data-uri',
        default=None,
        help='URL where the training files are located')
    args = parser.parse_args()
    print(args)
    return args

def main():
    args = get_args()
    bucket_name, blob_prefix = extract_bucket_and_prefix_from_gcs_path(args.
 ↪data_uri)

    local_data_dir = os.path.join(os.getcwd(), tempfile.gettempdir())
    files = ['x_train.npy', 'y_train.npy', 'x_test.npy', 'y_test.npy']

    for file in files:
        download_blob(bucket_name, os.path.join(blob_prefix,file), os.path.
 ↪join(local_data_dir,file))

    X_train = np.load(local_data_dir + '/x_train.npy')
    y_train = np.load(local_data_dir + '/y_train.npy')
    X_test = np.load(local_data_dir + '/x_test.npy')
    y_test = np.load(local_data_dir + '/y_test.npy')

    # Build and train the model
    model = Sequential([
        LSTM({lstm_units}, input_shape=[n_input_steps, n_features],␣
 ↪recurrent_activation=None),
        Dense(n_output_steps)])

    model.compile(optimizer='adam', loss='mae')

    early_stopping = EarlyStopping(monitor='val_loss', patience=patience)
    _ = model.fit(x=X_train, y=y_train, validation_data=(X_test, y_test),␣
 ↪epochs=epochs, callbacks=[early_stopping])

    # Export the model
    model.save(os.environ["AIP_MODEL_DIR"])

if __name__ == '__main__':
    main()
"""
```

```

with open(os.path.join(TRAINER_DIR, 'task.py'), 'w') as f:
    f.write(model_template.format(**globals()))
```

[21]:

```
# Copy the data files to  GCS bucket

!gsutil -m cp -r trainer/*.npy $BUCKET_URI/$TRAINER_DIR
```

```
Copying file://trainer/x_test.npy [Content-Type=application/octet-stream]…
Copying file://trainer/x_train.npy [Content-Type=application/octet-stream]…
Copying file://trainer/y_test.npy [Content-Type=application/octet-stream]…
Copying file://trainer/y_train.npy [Content-Type=application/octet-stream]…
/ [4/4 files][  1.9 MiB/  1.9 MiB] 100% Done
Operation completed over 4 objects/1.9 MiB.
```

[22]:

```
# List the contents of the bucket

!gsutil ls $BUCKET_URI/$TRAINER_DIR
```

```
gs://vertex-ai-projects/trainer/x_test.npy
gs://vertex-ai-projects/trainer/x_train.npy
gs://vertex-ai-projects/trainer/y_test.npy
gs://vertex-ai-projects/trainer/y_train.npy
```

0.1.4 Starting the training job

[23]:

```
CMDARGS = [
    f"--data-uri={BUCKET_URI}/{TRAINER_DIR}"
]
TRAIN_VERSION = "tf-cpu.2-6"
DEPLOY_VERSION = "tf2-cpu.2-6"

TRAIN_IMAGE = "us-docker.pkg.dev/vertex-ai/training/{}:latest".
 ↪format(TRAIN_VERSION)
DEPLOY_IMAGE = "us-docker.pkg.dev/vertex-ai/prediction/{}:latest".
 ↪format(DEPLOY_VERSION)
```

[24]:

```
# Re-run these additional parameters if you need to create a new training job

TIMESTAMP = str(datetime.datetime.now().time())
JOB_NAME = 'vertex_ai_training_' + TIMESTAMP
MODEL_DISPLAY_NAME = MODEL_NAME + TIMESTAMP
```

[25]:

```
# Create and run the training job

job = aiplatform.CustomTrainingJob(
    display_name=JOB_NAME,
    script_path=f"{TRAINER_DIR}/task.py",
    container_uri=TRAIN_IMAGE,
    model_serving_container_image_uri=DEPLOY_IMAGE,
)

model = job.run(
        model_display_name=MODEL_DISPLAY_NAME,
        args=CMDARGS,
)
```

```
INFO:google.cloud.aiplatform.utils.source_utils:Training script copied to:
gs://vertex-ai-projects/aiplatform-2021-12-15-01:30:35.696-aiplatform_custom_tra
iner_script-0.1.tar.gz.
INFO:google.cloud.aiplatform.training_jobs:Training Output directory:
gs://vertex-ai-projects/aiplatform-custom-training-2021-12-15-01:30:35.913
INFO:google.cloud.aiplatform.training_jobs:View Training:
https://console.cloud.google.com/ai/platform/locations/us-
west1/training/1154663131025244160?project=358157140210
INFO:google.cloud.aiplatform.training_jobs:CustomTrainingJob
projects/358157140210/locations/us-west1/trainingPipelines/1154663131025244160
current state:
PipelineState.PIPELINE_STATE_PENDING
INFO:google.cloud.aiplatform.training_jobs:CustomTrainingJob
projects/358157140210/locations/us-west1/trainingPipelines/1154663131025244160
current state:
PipelineState.PIPELINE_STATE_PENDING
INFO:google.cloud.aiplatform.training_jobs:View backing custom job:
https://console.cloud.google.com/ai/platform/locations/us-
west1/training/6651869426184355840?project=358157140210
INFO:google.cloud.aiplatform.training_jobs:CustomTrainingJob
projects/358157140210/locations/us-west1/trainingPipelines/1154663131025244160
current state:
PipelineState.PIPELINE_STATE_RUNNING
INFO:google.cloud.aiplatform.training_jobs:CustomTrainingJob
projects/358157140210/locations/us-west1/trainingPipelines/1154663131025244160
current state:
PipelineState.PIPELINE_STATE_RUNNING
INFO:google.cloud.aiplatform.training_jobs:CustomTrainingJob
projects/358157140210/locations/us-west1/trainingPipelines/1154663131025244160
current state:
PipelineState.PIPELINE_STATE_RUNNING
INFO:google.cloud.aiplatform.training_jobs:CustomTrainingJob
projects/358157140210/locations/us-west1/trainingPipelines/1154663131025244160
current state:
PipelineState.PIPELINE_STATE_RUNNING
INFO:google.cloud.aiplatform.training_jobs:CustomTrainingJob run completed.
Resource name: projects/358157140210/locations/us-
```

```
west1/trainingPipelines/1154663131025244160
INFO:google.cloud.aiplatform.training_jobs:Model available at
projects/358157140210/locations/us-west1/models/8443862273226702848
```

0.2 Deploying the model endpoint

[26]:

```
DEPLOYED_NAME = f"{MODEL_NAME}_deployed-" + TIMESTAMP

endpoint = model.deploy(
    deployed_model_display_name=DEPLOYED_NAME,
    machine_type="n1-standard-4",
    min_replica_count=1,
    max_replica_count=1,
    traffic_split={"0": 100},
)
```

```
INFO:google.cloud.aiplatform.models:Creating Endpoint
INFO:google.cloud.aiplatform.models:Create Endpoint backing LRO:
projects/358157140210/locations/us-
west1/endpoints/8630515367157956608/operations/4909174282484973568
INFO:google.cloud.aiplatform.models:Endpoint created. Resource name:
projects/358157140210/locations/us-west1/endpoints/8630515367157956608
INFO:google.cloud.aiplatform.models:To use this Endpoint in another session:
INFO:google.cloud.aiplatform.models:endpoint =
aiplatform.Endpoint('projects/358157140210/locations/us-
west1/endpoints/8630515367157956608')
INFO:google.cloud.aiplatform.models:Deploying model to Endpoint :
projects/358157140210/locations/us-west1/endpoints/8630515367157956608
INFO:google.cloud.aiplatform.models:Deploy Endpoint model backing LRO:
projects/358157140210/locations/us-
west1/endpoints/8630515367157956608/operations/3763008177319182336
INFO:google.cloud.aiplatform.models:Endpoint model deployed. Resource name:
projects/358157140210/locations/us-west1/endpoints/8630515367157956608
```

0.3 Predictions on deployed model

[35]:

```
# Predictions for the first test instance

raw_predictions = endpoint.predict(instances=X_test.tolist()).predictions[0]
predicted_values = inverse_scale(np.array([raw_predictions])).round()

actual_values = inverse_scale(np.array(y_test[0]))
```

[36]:

```
# comparison to actual value

print('Predicted riders:', predicted_values)
print('Actual riders:    ', actual_values)
```

```
Predicted riders: [[1675134. 1687431. 1700339.  997220.  674979. 1602980.
1673882.]]
Actual riders:    [[1647321.]
 [1668584.]
 [1687618.]
 [1060043.]
 [ 786217.]
 [1517370.]
 [1506995.]]
```

0.4 Cleanup

[ ]:

```
# delete_training_job = True
# delete_model = True
# delete_endpoint = True


# delete_bucket = False

# job.delete()

# endpoint.delete(force=True)

# model.delete()

# # if delete_bucket and "BUCKET" in globals():
# #     ! gsutil -m rm -r $BUCKET
```