Using custom containers with AI Platform Training

Learning Objectives:

1. Learn how to create a train and a validation split with Big Query
2. Learn how to wrap a machine learning model into a Docker container and train in on CAIP
3. Learn how to use the hyperparameter tunning engine on GCP to find the best hyperparameters
4. Learn how to deploy a trained machine learning model GCP as a rest API and query it

In this lab, you develop, package as a docker image, and run on **AI Platform Training** a training application that trains a multi-class classification model that predicts the type of forest cover from cartographic data. The dataset used in the lab is based on **Covertype Data Set** from UCI Machine Learning Repository.

The training code uses `scikit-learn` for data pre-processing and modeling. The code has been instrumented using the `hypertune` package so it can be used with **AI Platform** hyperparameter tuning.

In [ ]:

```
import json
import os
import numpy as np
import pandas as pd
import pickle
import uuid
import time
import tempfile

from googleapiclient import discovery
from googleapiclient import errors

from google.cloud import bigquery
from jinja2 import Template
from kfp.components import func_to_container_op
from typing import NamedTuple

from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
```

Configure environment settings

Set location paths, connections strings, and other environment settings. Make sure to update `REGION`, and `ARTIFACT_STORE` with the settings reflecting your lab environment.

- `REGION` - the compute region for AI Platform Training and Prediction
- `ARTIFACT_STORE` - the GCS bucket created during installation of AI Platform Pipelines. The bucket name starts with the `qwiklabs-gcp-xx-xxxxxxx-`

`kubeflowpipelines-default` prefix.

In [ ]:
```
!gsutil ls
```

In [ ]:
```
REGION = 'us-central1'
ARTIFACT_STORE = 'gs://qwiklabs-gcp-xx-xxxxxxx-kubeflowpipelines-default'   #C

PROJECT_ID = !(gcloud config get-value core/project)
PROJECT_ID = PROJECT_ID[0]
DATA_ROOT='{}/data'.format(ARTIFACT_STORE)
JOB_DIR_ROOT='{}/jobs'.format(ARTIFACT_STORE)
TRAINING_FILE_PATH='{}/{}/{}'.format(DATA_ROOT, 'training', 'dataset.csv')
VALIDATION_FILE_PATH='{}/{}/{}'.format(DATA_ROOT, 'validation', 'dataset.csv'
```

Explore the Covertype dataset

In [ ]:
```
%%bigquery
SELECT *
FROM `covertype_dataset.covertype`
```

Create training and validation splits

Use BigQuery to sample training and validation splits and save them to GCS storage

Create a training split

In [ ]:
```
!bq query \
-n 0 \
--destination_table covertype_dataset.training \
--replace \
--use_legacy_sql=false \
'SELECT * \
FROM `covertype_dataset.covertype` AS cover \
WHERE \
MOD(ABS(FARM_FINGERPRINT(TO_JSON_STRING(cover))), 10) IN (1, 2, 3, 4)'
```

In [ ]:
```
!bq extract \
--destination_format CSV \
covertype_dataset.training \
$TRAINING_FILE_PATH
```

Create a validation split

In [ ]:
```
!bq query \
-n 0 \
--destination_table covertype_dataset.validation \
--replace \
--use_legacy_sql=false \
'SELECT * \
FROM `covertype_dataset.covertype` AS cover \
WHERE \
MOD(ABS(FARM_FINGERPRINT(TO_JSON_STRING(cover))), 10) IN (8)'
```

In [ ]:
```
!bq extract \
--destination_format CSV \
covertype_dataset.validation \
$VALIDATION_FILE_PATH
```

In [ ]:
```
df_train = pd.read_csv(TRAINING_FILE_PATH)
df_validation = pd.read_csv(VALIDATION_FILE_PATH)
print(df_train.shape)
print(df_validation.shape)
```

Develop a training application

Configure the `sklearn` training pipeline.

The training pipeline preprocesses data by standardizing all numeric features using `sklearn.preprocessing.StandardScaler` and encoding all categorical features using `sklearn.preprocessing.OneHotEncoder` . It uses stochastic gradient descent linear classifier (`SGDClassifier`) for modeling.

In [ ]:
```
numeric_feature_indexes = slice(0, 10)
categorical_feature_indexes = slice(10, 12)

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_feature_indexes),
        ('cat', OneHotEncoder(), categorical_feature_indexes)
    ])

pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', SGDClassifier(loss='log', tol=1e-3))
])
```

Convert all numeric features to `float64`

To avoid warning messages from `StandardScaler` all numeric features are converted to `float64` .

In [ ]:
```
num_features_type_map = {feature: 'float64' for feature in df_train.columns[nu

df_train = df_train.astype(num_features_type_map)
df_validation = df_validation.astype(num_features_type_map)
```

Run the pipeline locally.

In [ ]:
```
X_train = df_train.drop('Cover_Type', axis=1)
y_train = df_train['Cover_Type']
X_validation = df_validation.drop('Cover_Type', axis=1)
y_validation = df_validation['Cover_Type']

pipeline.set_params(classifier__alpha=0.001, classifier__max_iter=200)
pipeline.fit(X_train, y_train)
```

Calculate the trained model's accuracy.

In [ ]:
```
accuracy = pipeline.score(X_validation, y_validation)
print(accuracy)
```

Prepare the hyperparameter tuning application.

Since the training run on this dataset is computationally expensive you can benefit from running a distributed hyperparameter tuning job on AI Platform Training.

In [ ]:
```
TRAINING_APP_FOLDER = 'training_app'
os.makedirs(TRAINING_APP_FOLDER, exist_ok=True)
```

Write the tuning script.

Notice the use of the `hypertune` package to report the `accuracy` optimization metric to AI Platform hyperparameter tuning service.

In [ ]:
```
%%writefile {TRAINING_APP_FOLDER}/train.py

# Copyright 2019 Google Inc. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#             http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

import os
import subprocess
import sys

import fire
import pickle
import numpy as np
import pandas as pd

import hypertune

from sklearn.compose import ColumnTransformer
from sklearn.linear_model import SGDClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder


def train_evaluate(job_dir, training_dataset_path, validation_dataset_path, a

    df_train = pd.read_csv(training_dataset_path)
    df_validation = pd.read_csv(validation_dataset_path)

    if not hptune:
        df_train = pd.concat([df_train, df_validation])
```

```
    numeric_feature_indexes = slice(0, 10)
    categorical_feature_indexes = slice(10, 12)

    preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_feature_indexes),
        ('cat', OneHotEncoder(), categorical_feature_indexes)
    ])

    pipeline = Pipeline([
        ('preprocessor', preprocessor),
        ('classifier', SGDClassifier(loss='log',tol=1e-3))
    ])

    num_features_type_map = {feature: 'float64' for feature in df_train.colum
    df_train = df_train.astype(num_features_type_map)
    df_validation = df_validation.astype(num_features_type_map)

    print('Starting training: alpha={}, max_iter={}'.format(alpha, max_iter))
    X_train = df_train.drop('Cover_Type', axis=1)
    y_train = df_train['Cover_Type']

    pipeline.set_params(classifier__alpha=alpha, classifier__max_iter=max_ite
    pipeline.fit(X_train, y_train)

    if hptune:
        X_validation = df_validation.drop('Cover_Type', axis=1)
        y_validation = df_validation['Cover_Type']
        accuracy = pipeline.score(X_validation, y_validation)
        print('Model accuracy: {}'.format(accuracy))
        # Log it with hypertune
        hpt = hypertune.HyperTune()
        hpt.report_hyperparameter_tuning_metric(
          hyperparameter_metric_tag='accuracy',
          metric_value=accuracy
        )

    # Save the model
    if not hptune:
        model_filename = 'model.pkl'
        with open(model_filename, 'wb') as model_file:
            pickle.dump(pipeline, model_file)
        gcs_model_path = "{}/{}".format(job_dir, model_filename)
        subprocess.check_call(['gsutil', 'cp', model_filename, gcs_model_path
        print("Saved model in: {}".format(gcs_model_path))

if __name__ == "__main__":
    fire.Fire(train_evaluate)
```

Package the script into a docker image.

Notice that we are installing specific versions of `scikit-learn` and `pandas` in the training image. This is done to make sure that the training runtime is aligned with the serving runtime. Later in the notebook you will deploy the model to AI Platform Prediction, using the 1.15 version of AI Platform Prediction runtime.

Make sure to update the URI for the base image so that it points to your project's **Container Registry**.

In [ ]:

```
%%writefile {TRAINING_APP_FOLDER}/Dockerfile
```

```
FROM gcr.io/deeplearning-platform-release/base-cpu
RUN pip install -U fire cloudml-hypertune scikit-learn==0.20.4 pandas==0.24.2
WORKDIR /app
COPY train.py .

ENTRYPOINT ["python", "train.py"]
```

Build the docker image.

You use **Cloud Build** to build the image and push it your project's **Container Registry**. As you use the remote cloud service to build the image, you don't need a local installation of Docker.

In [ ]:
```
IMAGE_NAME='trainer_image'
IMAGE_TAG='latest'
IMAGE_URI='gcr.io/{}/{}:{}'.format(PROJECT_ID, IMAGE_NAME, IMAGE_TAG)
```

In [ ]:
```
!gcloud builds submit --tag $IMAGE_URI $TRAINING_APP_FOLDER
```

Submit an AI Platform hyperparameter tuning job

Create the hyperparameter configuration file.

Recall that the training code uses `SGDClassifier` . The training application has been designed to accept two hyperparameters that control `SGDClassifier` :

- Max iterations
- Alpha

The below file configures AI Platform hypertuning to run up to 6 trials on up to three nodes and to choose from two discrete values of `max_iter` and the linear range betwee 0.00001 and 0.001 for `alpha` .

In [ ]:
```
%%writefile {TRAINING_APP_FOLDER}/hptuning_config.yaml

# Copyright 2019 Google Inc. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#            http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

trainingInput:
  hyperparameters:
    goal: MAXIMIZE
    maxTrials: 4
    maxParallelTrials: 4
    hyperparameterMetricTag: accuracy
```

```
    enableTrialEarlyStopping: TRUE
    params:
    - parameterName: max_iter
      type: DISCRETE
      discreteValues: [
          200,
          500
          ]
    - parameterName: alpha
      type: DOUBLE
      minValue:  0.00001
      maxValue:  0.001
      scaleType: UNIT_LINEAR_SCALE
```

Start the hyperparameter tuning job.

Use the `gcloud` command to start the hyperparameter tuning job.

In [ ]:

```
JOB_NAME = "JOB_{}".format(time.strftime("%Y%m%d_%H%M%S"))
JOB_DIR = "{}/{}".format(JOB_DIR_ROOT, JOB_NAME)
SCALE_TIER = "BASIC"

!gcloud ai-platform jobs submit training $JOB_NAME \
--region=$REGION \
--job-dir=$JOB_DIR \
--master-image-uri=$IMAGE_URI \
--scale-tier=$SCALE_TIER \
--config $TRAINING_APP_FOLDER/hptuning_config.yaml \
-- \
--training_dataset_path=$TRAINING_FILE_PATH \
--validation_dataset_path=$VALIDATION_FILE_PATH \
--hptune
```

Monitor the job.

You can monitor the job using GCP console or from within the notebook using `gcloud` commands.

In [ ]:

```
!gcloud ai-platform jobs describe $JOB_NAME
```

In [ ]:

```
!gcloud ai-platform jobs stream-logs $JOB_NAME
```

Retrieve HP-tuning results.

After the job completes you can review the results using GCP Console or programatically by calling the AI Platform Training REST end-point.

In [ ]:

```
ml = discovery.build('ml', 'v1')

job_id = 'projects/{}/jobs/{}'.format(PROJECT_ID, JOB_NAME)
request = ml.projects().jobs().get(name=job_id)

try:
    response = request.execute()
except errors.HttpError as err:
    print(err)
except:
```

```
    print("Unexpected error")

response
```

The returned run results are sorted by a value of the optimization metric. The best run is the first item on the returned list.

In [ ]:
```
response['trainingOutput']['trials'][0]
```

Retrain the model with the best hyperparameters

You can now retrain the model using the best hyperparameters and using combined training and validation splits as a training dataset.

Configure and run the training job

In [ ]:
```
alpha = response['trainingOutput']['trials'][0]['hyperparameters']['alpha']
max_iter = response['trainingOutput']['trials'][0]['hyperparameters']['max_ite
```

In [ ]:
```
JOB_NAME = "JOB_{}".format(time.strftime("%Y%m%d_%H%M%S"))
JOB_DIR = "{}/{}".format(JOB_DIR_ROOT, JOB_NAME)
SCALE_TIER = "BASIC"

!gcloud ai-platform jobs submit training $JOB_NAME \
--region=$REGION \
--job-dir=$JOB_DIR \
--master-image-uri=$IMAGE_URI \
--scale-tier=$SCALE_TIER \
-- \
--training_dataset_path=$TRAINING_FILE_PATH \
--validation_dataset_path=$VALIDATION_FILE_PATH \
--alpha=$alpha \
--max_iter=$max_iter \
--nohptune
```

In [ ]:
```
!gcloud ai-platform jobs stream-logs $JOB_NAME
```

Examine the training output

The training script saved the trained model as the 'model.pkl' in the `JOB_DIR` folder on GCS.

In [ ]:
```
!gsutil ls $JOB_DIR
```

Deploy the model to AI Platform Prediction

Create a model resource

In [ ]:
```
model_name = 'forest_cover_classifier'
labels = "task=classifier,domain=forestry"

!gcloud ai-platform models create  $model_name \
```

```
 --regions=$REGION \
 --labels=$labels
```

Create a model version

In [ ]:
```
model_version = 'v01'

!gcloud ai-platform versions create {model_version} \
 --model={model_name} \
 --origin=$JOB_DIR \
 --runtime-version=1.15 \
 --framework=scikit-learn \
 --python-version=3.7\
 --region global
```

Serve predictions

Prepare the input file with JSON formated instances.

In [ ]:
```
input_file = 'serving_instances.json'

with open(input_file, 'w') as f:
    for index, row in X_validation.head().iterrows():
        f.write(json.dumps(list(row.values)))
        f.write('\n')
```

In [ ]:
```
!cat $input_file
```

Invoke the model

In [ ]:
```
!gcloud ai-platform predict \
--model $model_name \
--version $model_version \
--json-instances $input_file\
--region global
```

Licensed under the Apache License, Version 2.0 (the \"License\"); you may not use this file except in compliance with the License. You may obtain a copy of the License at https://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an \"AS IS\" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.