In [ ]:
```
# Copyright 2021 Google LLC
#
# 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.
```

Vertex SDK: AutoML training text classification model for batch prediction

 Run in Colab  View on GitHub Open in Vertex AI Workbench

Overview

This tutorial demonstrates how to use the Vertex SDK to create text classification models and do batch prediction using a Google Cloud AutoML model.

Dataset

The dataset used for this tutorial is the Happy Moments dataset from Kaggle Datasets. The version of the dataset you will use in this tutorial is stored in a public Cloud Storage bucket.

Objective

In this tutorial, you create an AutoML text classification model from a Python script, and then do a batch prediction using the Vertex SDK. You can alternatively create and deploy models using the `gcloud` command-line tool or online using the Cloud Console.

The steps performed include:

- Create a Vertex `Dataset` resource.
- Train the model.
- View the model evaluation.
- Make a batch prediction.

There is one key difference between using batch prediction and using online prediction:

- Prediction Service: Does an on-demand prediction for the entire set of instances (i.e., one or more data items) and returns the results in real-time.
- Batch Prediction Service: Does a queued (batch) prediction for the entire set of instances in the background and stores the results in a Cloud Storage bucket when ready.

Costs

This tutorial uses billable components of Google Cloud:

- Vertex AI
- Cloud Storage

Learn about Vertex AI pricing and Cloud Storage pricing, and use the Pricing Calculator to generate a cost estimate based on your projected usage.

Set up your local development environment

If you are using Colab or Google Cloud Notebooks, your environment already meets all the requirements to run this notebook. You can skip this step.

Otherwise, make sure your environment meets this notebook's requirements. You need the following:

- The Cloud Storage SDK
- Git
- Python 3
- virtualenv
- Jupyter notebook running in a virtual environment with Python 3

The Cloud Storage guide to Setting up a Python development environment and the Jupyter installation guide provide detailed instructions for meeting these requirements. The following steps provide a condensed set of instructions:

1. Install and initialize the SDK.
2. Install Python 3.
3. Install virtualenv and create a virtual environment that uses Python 3. Activate the virtual environment.
4. To install Jupyter, run `pip3 install jupyter` on the command-line in a terminal shell.
5. To launch Jupyter, run `jupyter notebook` on the command-line in a terminal shell.

6. Open this notebook in the Jupyter Notebook Dashboard.

Installation

Install the latest version of Vertex SDK for Python.

In [ ]:
```
import os

# Google Cloud Notebook
if os.path.exists("/opt/deeplearning/metadata/env_version"):
    USER_FLAG = "--user"
else:
    USER_FLAG = ""

! pip3 install --upgrade google-cloud-aiplatform $USER_FLAG
```

Install the latest GA version of *google-cloud-storage* library as well.

In [ ]:
```
! pip3 install -U google-cloud-storage $USER_FLAG
```

In [ ]:
```
if os.environ["IS_TESTING"]:
    ! pip3 install --upgrade tensorflow $USER_FLAG
```

Restart the kernel

Once you've installed the additional packages, you need to restart the notebook kernel so it can find the packages.

In [ ]:
```
import os

if not os.getenv("IS_TESTING"):
    # Automatically restart kernel after installs
    import IPython

    app = IPython.Application.instance()
    app.kernel.do_shutdown(True)
```

Before you begin

GPU runtime

This tutorial does not require a GPU runtime.

Set up your Google Cloud project

The following steps are required, regardless of your notebook environment.

1. Select or create a Google Cloud project. When you first create an account, you get a $300 free credit towards your compute/storage costs.

2. Make sure that billing is enabled for your project.

3. Enable the following APIs: Vertex AI APIs, Compute Engine APIs, and Cloud Storage.

4. If you are running this notebook locally, you will need to install the Cloud SDK).

5. Enter your project ID in the cell below. Then run the cell to make sure the Cloud SDK uses the right project for all the commands in this notebook.

Note: Jupyter runs lines prefixed with `!` as shell commands, and it interpolates Python variables prefixed with `$`.

In [ ]:
```
PROJECT_ID = "[your-project-id]"  # @param {type:"string"}
```

In [ ]:
```
if PROJECT_ID == "" or PROJECT_ID is None or PROJECT_ID == "[your-project-id]":
    # Get your GCP project id from gcloud
    shell_output = ! gcloud config list --format 'value(core.project)' 2>/dev/null
    PROJECT_ID = shell_output[0]
    print("Project ID:", PROJECT_ID)
```

In [ ]:
```
! gcloud config set project $PROJECT_ID
```

Region

You can also change the `REGION` variable, which is used for operations throughout the rest of this notebook. Below are regions supported for Vertex AI. We recommend that you choose the region closest to you.

- Americas: `us-central1`
- Europe: `europe-west4`
- Asia Pacific: `asia-east1`

You may not use a multi-regional bucket for training with Vertex AI. Not all regions provide support for all Vertex AI services.

Learn more about Vertex AI regions

In [ ]:
```
REGION = "us-central1"  # @param {type: "string"}
```

Timestamp

If you are in a live tutorial session, you might be using a shared test account or project. To avoid name collisions between users on resources created, you create a timestamp for each instance session, and append the timestamp onto the name of resources you create in this tutorial.

In [ ]:
```
from datetime import datetime

TIMESTAMP = datetime.now().strftime("%Y%m%d%H%M%S")
```

Authenticate your Google Cloud account

If you are using Google Cloud Notebooks, your environment is already authenticated. Skip this step.

If you are using Colab, run the cell below and follow the instructions when prompted to authenticate your account via oAuth.

Otherwise, follow these steps:

In the Cloud Console, go to the Create service account key page.

Click Create service account.

In the **Service account name** field, enter a name, and click **Create**.

In the **Grant this service account access to project** section, click the Role drop-down list. Type "Vertex" into the filter box, and select **Vertex Administrator**. Type "Storage Object Admin" into the filter box, and select **Storage Object Admin**.

Click Create. A JSON file that contains your key downloads to your local environment.

Enter the path to your service account key as the GOOGLE_APPLICATION_CREDENTIALS variable in the cell below and run the cell.

In [ ]:
```
# If you are running this notebook in Colab, run this cell and follow the
# instructions to authenticate your GCP account. This provides access to your
# Cloud Storage bucket and lets you submit training jobs and prediction
# requests.

import os
import sys

# If on Google Cloud Notebook, then don't execute this code
if not os.path.exists("/opt/deeplearning/metadata/env_version"):
    if "google.colab" in sys.modules:
        from google.colab import auth as google_auth

        google_auth.authenticate_user()

    # If you are running this notebook locally, replace the string below with the
    # path to your service account key and run this cell to authenticate your GCP
    # account.
    elif not os.getenv("IS_TESTING"):
        %env GOOGLE_APPLICATION_CREDENTIALS ''
```

Create a Cloud Storage bucket

The following steps are required, regardless of your notebook environment.

When you initialize the Vertex SDK for Python, you specify a Cloud Storage staging bucket. The staging bucket is where all the data associated with your dataset and model resources are retained across sessions.

Set the name of your Cloud Storage bucket below. Bucket names must be globally unique across all Google Cloud projects, including those outside of your organization.

In [ ]:
```
BUCKET_NAME = "gs://[your-bucket-name]"  # @param {type:"string"}
```

In [ ]:
```
if BUCKET_NAME == "" or BUCKET_NAME is None or BUCKET_NAME == "gs://[your-bucket-na
    BUCKET_NAME = "gs://" + PROJECT_ID + "aip-" + TIMESTAMP
```

Only if your bucket doesn't already exist: Run the following cell to create your Cloud Storage bucket.

In [ ]:
```
! gsutil mb -l $REGION $BUCKET_NAME
```

Finally, validate access to your Cloud Storage bucket by examining its contents:

In [ ]:
```
! gsutil ls -al $BUCKET_NAME
```

Set up variables

Next, set up some variables used throughout the tutorial.

Import libraries and define constants

In [ ]:
```
import google.cloud.aiplatform as aip
```

Initialize Vertex SDK for Python

Initialize the Vertex SDK for Python for your project and corresponding bucket.

In [ ]:
```
aip.init(project=PROJECT_ID, staging_bucket=BUCKET_NAME)
```

Tutorial

Now you are ready to start creating your own AutoML text classification model.

Location of Cloud Storage training data.

Now set the variable `IMPORT_FILE` to the location of the CSV index file in Cloud Storage.

In [ ]:
```
IMPORT_FILE = "gs://cloud-ml-data/NL-classification/happiness.csv"
```

Quick peek at your data

This tutorial uses a version of the Happy Moments dataset that is stored in a public Cloud Storage bucket, using a CSV index file.

Start by doing a quick peek at the data. You count the number of examples by counting the number of rows in the CSV index file (`wc -l`) and then peek at the first few rows.

In [ ]:
```
if "IMPORT_FILES" in globals():
    FILE = IMPORT_FILES[0]
else:
    FILE = IMPORT_FILE

count = ! gsutil cat $FILE | wc -l
print("Number of Examples", int(count[0]))

print("First 10 rows")
! gsutil cat $FILE | head
```

Create the Dataset

Next, create the `Dataset` resource using the `create` method for the `TextDataset` class, which takes the following parameters:

- `display_name` : The human readable name for the `Dataset` resource.
- `gcs_source` : A list of one or more dataset index files to import the data items into the `Dataset` resource.
- `import_schema_uri` : The data labeling schema for the data items.

This operation may take several minutes.

In [ ]:
```
dataset = aip.TextDataset.create(
    display_name="Happy Moments" + "_" + TIMESTAMP,
    gcs_source=[IMPORT_FILE],
    import_schema_uri=aip.schema.dataset.ioformat.text.single_label_classification,
)

print(dataset.resource_name)
```

Create and run training pipeline

To train an AutoML model, you perform two steps: 1) create a training pipeline, and 2) run the pipeline.

Create training pipeline

An AutoML training pipeline is created with the `AutoMLTextTrainingJob` class, with the following parameters:

- `display_name` : The human readable name for the `TrainingJob` resource.
- `prediction_type` : The type task to train the model for.
  - `classification` : A text classification model.
  - `sentiment` : A text sentiment analysis model.
  - `extraction` : A text entity extraction model.
- `multi_label` : If a classification task, whether single (False) or multi-labeled (True).
- `sentiment_max` : If a sentiment analysis task, the maximum sentiment value.

The instantiated object is the DAG (directed acyclic graph) for the training pipeline.

In [ ]:
```
dag = aip.AutoMLTextTrainingJob(
    display_name="happydb_" + TIMESTAMP,
    prediction_type="classification",
    multi_label=False,
)

print(dag)
```

Run the training pipeline

Next, you run the DAG to start the training job by invoking the method `run` , with the following parameters:

- `dataset` : The `Dataset` resource to train the model.
- `model_display_name` : The human readable name for the trained model.
- `training_fraction_split` : The percentage of the dataset to use for training.
- `test_fraction_split` : The percentage of the dataset to use for test (holdout data).
- `validation_fraction_split` : The percentage of the dataset to use for validation.

The `run` method when completed returns the `Model` resource.

The execution of the training pipeline will take upto 20 minutes.

In [ ]:
```
model = dag.run(
    dataset=dataset,
    model_display_name="happydb_" + TIMESTAMP,
    training_fraction_split=0.8,
    validation_fraction_split=0.1,
    test_fraction_split=0.1,
)
```

Review model evaluation scores

After your model has finished training, you can review the evaluation scores for it.

First, you need to get a reference to the new model. As with datasets, you can either use the reference to the model variable you created when you deployed the model or you can list all of the models in your project.

In [ ]:
```
# Get model resource ID
models = aip.Model.list(filter="display_name=happydb_" + TIMESTAMP)

# Get a reference to the Model Service client
client_options = {"api_endpoint": f"{REGION}-aiplatform.googleapis.com"}
model_service_client = aip.gapic.ModelServiceClient(client_options=client_options)

model_evaluations = model_service_client.list_model_evaluations(
    parent=models[0].resource_name
)
model_evaluation = list(model_evaluations)[0]
print(model_evaluation)
```

Send a batch prediction request

Send a batch prediction to your deployed model.

Get test item(s)

Now do a batch prediction to your Vertex model. You will use arbitrary examples out of the dataset as a test items. Don't be concerned that the examples were likely used in training the model -- we just want to demonstrate how to make a prediction.

In [ ]:
```
test_items = ! gsutil cat $IMPORT_FILE | head -n2
if len(test_items[0]) == 3:
    _, test_item_1, test_label_1 = str(test_items[0]).split(",")
    _, test_item_2, test_label_2 = str(test_items[1]).split(",")
else:
    test_item_1, test_label_1 = str(test_items[0]).split(",")
    test_item_2, test_label_2 = str(test_items[1]).split(",")

print(test_item_1, test_label_1)
print(test_item_2, test_label_2)
```

Make the batch input file

Now make a batch input file, which you will store in your local Cloud Storage bucket. The batch input file can only be in JSONL format. For JSONL file, you make one dictionary entry per line for each data item (instance). The dictionary contains the key/value pairs:

- `content` : The Cloud Storage path to the file with the text item.
- `mime_type` : The content type. In our example, it is a `text` file.

For example:

```
                        {'content': '[your-bucket]/file1.txt',
    'mime_type': 'text'}
```

In [ ]:
```
import json

import tensorflow as tf

gcs_test_item_1 = BUCKET_NAME + "/test1.txt"
with tf.io.gfile.GFile(gcs_test_item_1, "w") as f:
    f.write(test_item_1 + "\n")
gcs_test_item_2 = BUCKET_NAME + "/test2.txt"
with tf.io.gfile.GFile(gcs_test_item_2, "w") as f:
    f.write(test_item_2 + "\n")

gcs_input_uri = BUCKET_NAME + "/test.jsonl"
with tf.io.gfile.GFile(gcs_input_uri, "w") as f:
    data = {"content": gcs_test_item_1, "mime_type": "text/plain"}
    f.write(json.dumps(data) + "\n")
    data = {"content": gcs_test_item_2, "mime_type": "text/plain"}
    f.write(json.dumps(data) + "\n")

print(gcs_input_uri)
! gsutil cat $gcs_input_uri
```

Make the batch prediction request

Now that your Model resource is trained, you can make a batch prediction by invoking the batch_predict() method, with the following parameters:

- `job_display_name` : The human readable name for the batch prediction job.
- `gcs_source` : A list of one or more batch request input files.
- `gcs_destination_prefix` : The Cloud Storage location for storing the batch prediction resuls.
- `sync` : If set to True, the call will block while waiting for the asynchronous batch job to complete.

In [ ]:
```
batch_predict_job = model.batch_predict(
    job_display_name="happydb_" + TIMESTAMP,
    gcs_source=gcs_input_uri,
    gcs_destination_prefix=BUCKET_NAME,
    sync=False,
)

print(batch_predict_job)
```

Wait for completion of batch prediction job

Next, wait for the batch job to complete. Alternatively, one can set the parameter `sync` to `True` in the `batch_predict()` method to block until the batch prediction job is completed.

In [ ]:
```
batch_predict_job.wait()
```

Get the predictions

Next, get the results from the completed batch prediction job.

The results are written to the Cloud Storage output bucket you specified in the batch prediction request. You call the method iter_outputs() to get a list of each Cloud Storage file generated with the results. Each file contains one or more prediction requests in a JSON format:

- `content` : The prediction request.
- `prediction` : The prediction response.
- `ids` : The internal assigned unique identifiers for each prediction request.
- `displayNames` : The class names for each class label.
- `confidences` : The predicted confidence, between 0 and 1, per class label.

In [ ]:
```
import json

import tensorflow as tf

bp_iter_outputs = batch_predict_job.iter_outputs()

prediction_results = list()
for blob in bp_iter_outputs:
    if blob.name.split("/")[-1].startswith("prediction"):
        prediction_results.append(blob.name)

tags = list()
for prediction_result in prediction_results:
    gfile_name = f"gs://{bp_iter_outputs.bucket.name}/{prediction_result}"
    with tf.io.gfile.GFile(name=gfile_name, mode="r") as gfile:
        for line in gfile.readlines():
            line = json.loads(line)
            print(line)
            break
```

Cleaning up

To clean up all Google Cloud resources used in this project, you can delete the Google Cloud project you used for the tutorial.

Otherwise, you can delete the individual resources you created in this tutorial:

- Dataset
- Pipeline
- Model
- Endpoint
- AutoML Training Job
- Batch Job
- Custom Job
- Hyperparameter Tuning Job
- Cloud Storage Bucket

In [ ]:
```
delete_all = True

if delete_all:
    # Delete the dataset using the Vertex dataset object
    try:
        if "dataset" in globals():
            dataset.delete()
    except Exception as e:
        print(e)

    # Delete the model using the Vertex model object
    try:
        if "model" in globals():
            model.delete()
    except Exception as e:
        print(e)

    # Delete the endpoint using the Vertex endpoint object
    try:
        if "endpoint" in globals():
            endpoint.delete()
    except Exception as e:
        print(e)

    # Delete the AutoML or Pipeline trainig job
    try:
        if "dag" in globals():
            dag.delete()
    except Exception as e:
        print(e)

    # Delete the custom trainig job
    try:
        if "job" in globals():
            job.delete()
    except Exception as e:
        print(e)

    # Delete the batch prediction job using the Vertex batch prediction object
    try:
```

```
        if "batch_predict_job" in globals():
            batch_predict_job.delete()
    except Exception as e:
        print(e)

    # Delete the hyperparameter tuning job using the Vertex hyperparameter tuning o
    try:
        if "hpt_job" in globals():
            hpt_job.delete()
    except Exception as e:
        print(e)

    if "BUCKET_NAME" in globals():
        ! gsutil rm -r $BUCKET_NAME
```