```
{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "098da034",
   "metadata": {},
   "source": [
    "# Assignment4 : Setting Up a Machine Learning Pipeline in The Cloud"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "12f56985",
   "metadata": {},
   "source": [
    "## Description\n",
    "\n",
    "In this assignment, you will learn how to train, save, load, deploy, and run a machine learning model in a cloud environment using Azure Machine Learning (Azure ML) services. Azure ML offers tools similar to the Amazon SageMaker SDK, enabling script-based training and model deployment as web services. You will begin with a straightforward ML code to understand how it can be executed and deployed in Azure. Following this, you'll apply what you've learned to deploy and run a more sophisticated ML code in the Azure cloud."
   ]
  },
  {
   "cell_type": "markdown",
   "id": "8724b87b",
   "metadata": {},
   "source": [
    "## Step 1. Training a Regression model on a local host"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "c48b4019",
   "metadata": {},
   "source": [
    "### 1- Create a dataset and save"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
```

```
 "id": "4e1b2241",
 "metadata": {},
 "outputs": [],
 "source": [
  "from sklearn import datasets\n",
  "import pickle"
 ]
},
{
 "cell_type": "code",
 "execution_count": 2,
 "id": "d83b2d29",
 "metadata": {},
 "outputs": [],
 "source": [
  "X, y = datasets.make_regression(100, 1, noise=5, bias=0)"
 ]
},
{
 "cell_type": "code",
 "execution_count": 3,
 "id": "c25527fc",
 "metadata": {},
 "outputs": [],
 "source": [
  "pickle.dump([X,y], open('./train.pickle', 'wb'))"
 ]
},
{
 "cell_type": "markdown",
 "id": "a601e752",
 "metadata": {},
 "source": [
  "### 2- Create a model from the dataset"
 ]
},
{
 "cell_type": "code",
 "execution_count": 4,
 "id": "8aa04224",
 "metadata": {},
 "outputs": [],
 "source": [
  "from sklearn.linear_model import LinearRegression\n",
  "import pickle"
 ]
```

```
  },
  {
   "cell_type": "code",
   "execution_count": 5,
   "id": "00caf709",
   "metadata": {},
   "outputs": [],
   "source": [
    "[XX, yy] = pickle.load(open('./train.pickle', 'rb'))"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 6,
   "id": "fc9f9158",
   "metadata": {},
   "outputs": [],
   "source": [
    "model = LinearRegression()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 7,
   "id": "47ba5aca",
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/html": [
       "<style>#sk-container-id-1 {color: black;background-color:
white;}#sk-container-id-1 pre{padding: 0;}#sk-container-id-1
div.sk-toggleable {background-color: white;}#sk-container-id-1
label.sk-toggleable__label {cursor: pointer;display: block;width:
100%;margin-bottom: 0;padding: 0.3em;box-sizing: border-box;text-align:
center;}#sk-container-id-1 label.sk-toggleable__label-arrow:before
{content: \"▸\";float: left;margin-right: 0.25em;color:
#696969;}#sk-container-id-1 label.sk-toggleable__label-arrow:hover:before
{color: black;}#sk-container-id-1 div.sk-estimator:hover
label.sk-toggleable__label-arrow:before {color: black;}#sk-container-id-1
div.sk-toggleable__content {max-height: 0;max-width: 0;overflow:
hidden;text-align: left;background-color: #f0f8ff;}#sk-container-id-1
div.sk-toggleable__content pre {margin: 0.2em;color: black;border-radius:
0.25em;background-color: #f0f8ff;}#sk-container-id-1
input.sk-toggleable__control:checked~div.sk-toggleable__content
{max-height: 200px;max-width: 100%;overflow: auto;}#sk-container-id-1
```

input.sk-toggleable__control:checked~label.sk-toggleable__label-arrow:before {content: \"▾\";}#sk-container-id-1 div.sk-estimator input.sk-toggleable__control:checked~label.sk-toggleable__label {background-color: #d4ebff;}#sk-container-id-1 div.sk-label input.sk-toggleable__control:checked~label.sk-toggleable__label {background-color: #d4ebff;}#sk-container-id-1 input.sk-hidden--visually {border: 0;clip: rect(1px 1px 1px 1px);clip: rect(1px, 1px, 1px, 1px);height: 1px;margin: -1px;overflow: hidden;padding: 0;position: absolute;width: 1px;}#sk-container-id-1 div.sk-estimator {font-family: monospace;background-color: #f0f8ff;border: 1px dotted black;border-radius: 0.25em;box-sizing: border-box;margin-bottom: 0.5em;}#sk-container-id-1 div.sk-estimator:hover {background-color: #d4ebff;}#sk-container-id-1 div.sk-parallel-item::after {content: \"\";width: 100%;border-bottom: 1px solid gray;flex-grow: 1;}#sk-container-id-1 div.sk-label:hover label.sk-toggleable__label {background-color: #d4ebff;}#sk-container-id-1 div.sk-serial::before {content: \"\";position: absolute;border-left: 1px solid gray;box-sizing: border-box;top: 0;bottom: 0;left: 50%;z-index: 0;}#sk-container-id-1 div.sk-serial {display: flex;flex-direction: column;align-items: center;background-color: white;padding-right: 0.2em;padding-left: 0.2em;position: relative;}#sk-container-id-1 div.sk-item {position: relative;z-index: 1;}#sk-container-id-1 div.sk-parallel {display: flex;align-items: stretch;justify-content: center;background-color: white;position: relative;}#sk-container-id-1 div.sk-item::before, #sk-container-id-1 div.sk-parallel-item::before {content: \"\";position: absolute;border-left: 1px solid gray;box-sizing: border-box;top: 0;bottom: 0;left: 50%;z-index: -1;}#sk-container-id-1 div.sk-parallel-item {display: flex;flex-direction: column;z-index: 1;position: relative;background-color: white;}#sk-container-id-1 div.sk-parallel-item:first-child::after {align-self: flex-end;width: 50%;}#sk-container-id-1 div.sk-parallel-item:last-child::after {align-self: flex-start;width: 50%;}#sk-container-id-1 div.sk-parallel-item:only-child::after {width: 0;}#sk-container-id-1 div.sk-dashed-wrapped {border: 1px dashed gray;margin: 0 0.4em 0.5em 0.4em;box-sizing: border-box;padding-bottom: 0.4em;background-color: white;}#sk-container-id-1 div.sk-label label {font-family: monospace;font-weight: bold;display: inline-block;line-height: 1.2em;}#sk-container-id-1 div.sk-label-container {text-align: center;}#sk-container-id-1 div.sk-container {/* jupyter's `normalize.less` sets `[hidden] { display: none; }` but bootstrap.min.css set `[hidden] { display: none !important; }` so we also need the `!important` here to be able to override the default hidden behavior on the sphinx rendered scikit-learn.org. See: https://github.com/scikit-learn/scikit-learn/issues/21755 */display: inline-block !important;position: relative;}#sk-container-id-1 div.sk-text-repr-fallback {display: none;}</style><div id=\"sk-container-id-1\" class=\"sk-top-container\"><div

class=\"sk-text-repr-fallback\"><pre>LinearRegression()</pre>In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org. <div class=\"sk-container\" hidden><div class=\"sk-item\"><div class=\"sk-estimator sk-toggleable\"><input class=\"sk-toggleable__control sk-hidden--visually\" id=\"sk-estimator-id-1\" type=\"checkbox\" checked><label for=\"sk-estimator-id-1\" class=\"sk-toggleable__label sk-toggleable__label-arrow\">LinearRegression</label><div class=\"sk-toggleable__content\"><pre>LinearRegression()</pre> "
      ],
      "text/plain": [
       "LinearRegression()"
      ]
     },
     "execution_count": 7,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "model.fit(XX,yy)"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "0f0ba432",
   "metadata": {},
   "source": [
    "### 3- Make a test prediction"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 8,
   "id": "ee21fc81",
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "array([-0.51996188, 86.31367353, 173.14730893, 259.98094434])"
      ]
     },
     "execution_count": 8,

```
   "metadata": {},
   "output_type": "execute_result"
  }
 ],
 "source": [
  "model.predict([[0],[1],[2],[3]])"
 ]
},
{
 "cell_type": "markdown",
 "id": "e8d7df6e",
 "metadata": {},
 "source": [
  "### 4- Save the model to a file"
 ]
},
{
 "cell_type": "code",
 "execution_count": 9,
 "id": "fb971804",
 "metadata": {},
 "outputs": [],
 "source": [
  "p = pickle.dumps(model)"
 ]
},
{
 "cell_type": "code",
 "execution_count": 10,
 "id": "fe2d9f92",
 "metadata": {},
 "outputs": [],
 "source": [
  "pickle.dump(model, open('./model.pickle', 'wb'))"
 ]
},
{
 "cell_type": "markdown",
 "id": "cbaa7fed",
 "metadata": {},
 "source": [
  "### 5- Later load the model from a file"
 ]
},
{
 "cell_type": "code",
```

```
 "execution_count": 11,
 "id": "be9d2abf",
 "metadata": {},
 "outputs": [],
 "source": [
  "from sklearn.linear_model import LinearRegression\n",
  "import pickle"
 ]
},
{
 "cell_type": "code",
 "execution_count": 12,
 "id": "94e63048",
 "metadata": {},
 "outputs": [],
 "source": [
  "loaded_model = pickle.load(open('./model.pickle', 'rb'))"
 ]
},
{
 "cell_type": "markdown",
 "id": "8cbb920e",
 "metadata": {},
 "source": [
  "### 6- Make a test prediction"
 ]
},
{
 "cell_type": "code",
 "execution_count": 14,
 "id": "65309fa2",
 "metadata": {},
 "outputs": [
  {
   "data": {
    "text/plain": [
     "array([ -0.51996188,  86.31367353, 173.14730893, 259.98094434])"
    ]
   },
   "execution_count": 14,
   "metadata": {},
   "output_type": "execute_result"
  }
 ],
 "source": [
  "loaded_model.predict([[0],[1],[2],[3]])"
```

```
   ]
  },
  {
   "cell_type": "markdown",
   "id": "fa84a8bd",
   "metadata": {},
   "source": [
    "## Step 2.0 Training a Regression Model in the Cloud\n",
    "\n",
    "One of the simplest ways to run a training job in the cloud on Azure is by using Azure Machine Learning Studio and its integrated Jupyter Notebooks. Azure Machine Learning allows you to run and train custom models in Python by executing a Python script. In this step, you will learn how to manage the entire lifecycle of an ML application (load data, train the model, save the model, load the model, and make predictions) using Azure Machine Learning Studio. Azure Machine Learning Studio automatically uses an Azure Machine Learning Compute instance to handle the training job. We will use Azure Blob Storage to save the data and model and use it for predictions in the future.\n",
    "\n",
    "1) Setting up Azure Notebooks\n",
    "2) Creating Azure Machine Learning Script\n",
    "3) Creating Deployment Script\n"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "336cf56f",
   "metadata": {},
   "source": [
    "## Step 2.1  Create an Azure Machine Learning Compute Instance for Jupyter Notebooks\n",
    "\n",
    "### 1) Create a new workspace\n",
    "Access your Azure portal. Before creating a compute instance for Jupyter Notebooks in Azure, you need to have an Azure Machine Learning Workspace set up. \n",
    "\n",
    "Create a New Workspace:\n",
    "- In the Azure portal, click on \"Create a resource\".\n",
    "- Search for “Machine Learning” in the Marketplace.\n",
    "- Select “Machine Learning” from the search results and click \"Create\".\n",
    "\n",
    "Configure the Workspace:\n",
    "- Fill in the required fields such as the subscription, resource group
```

```
(you can create a new one if needed), and region.\n",
    "- Provide a unique name for your workspace.\n",
    "- Choose the default options or configure additional settings as per
your requirement.\n",
    "- Click \"Review + Create\" and then \"Create\" to initiate the
deployment of the workspace.\n",
    "\n",
    "Create a Compute Instance:\n",
    "- In the Machine Learning Studio, navigate to the “Compute” section in
the left-hand menu.\n",
    "- Under the “Compute Instances” tab, create a new compute
instance.\n",
    "- Provide a name for your compute instance.\n",
    "- Choose the default options for the instance (Standard_E4ds_v4) and
create it (it may take a few minutes for the status to change to
\"Running\").\n",
    "- Once the status changes from \"Creating\" to \"Running\", open
Jupyter Notebooks or JupyterLab from the compute instance's options."
   ]
  },
  {
   "cell_type": "markdown",
   "id": "c4dc4c0e",
   "metadata": {},
   "source": [
    "### 2) Create Folder Structure in Azure Notebooks\n",
    "Once inside Azure Notebooks, you will need to create some new
files/folders and upload some files created from the previous section.
Navigate to the folder where you want to create the folder and click on
AzureML notebook\n",
    "\n",
    "Upload these files into Azure Notebooks:\n",
    "- train.pickle\n",
    "\n",
    "<br/>\n",
    "\n",
    "Folder Structure:\n",
    "\n",
    "``` bash\n",
    "train_dir (folder)\n",
    "    train.pickle (file - Step 1.0)\n",
    "conda_dependency_file.yml (file - Step 2.1)\n",
    "regressionDeployment.ipynb (file - Step 2.2)\n",
    "score.py (file - Step 2.3)\n",
    "\n",
    "script (folder)\n",
```

```
    "    script.py (file - Step 2.3)\n",
    "```\n",
    "\n",
    "<br/>"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "0c753b59",
   "metadata": {},
   "source": [
    "## Step 2.2 Creating Azure Machine Learning Script\n",
    "\n",
    "### <span style=\"color:red\">You copy the code below combine all of these scripts together to create one training script called: <br/> <b>script.py </b></script>\n",
    "\n",
    "Azure Machine Learning Scripts can be broken down into 5 distinct parts. \n",
    "1) Creating Arguments\n",
    "2) Load Dataset\n",
    "3) Load Model\n",
    "4) Predict with Model\n",
    "5) Main Function\n"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "2b9a1a31",
   "metadata": {},
   "source": [
    "### 1. Creating Arguments"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "f96cf94c",
   "metadata": {},
   "source": [
    "### Training script"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "440bafa4",
   "metadata": {},
```

```
   "source": [
    "- Your Azure Machine Learning training script will contain a function
called \"parse_args\" that will save environment variables to your training
file.\n",
    "- These arguments will act like environment variables and will be
utilized throughout the rest of your script.\n",
    "- Many of these arguments will be utilized as file paths to upload
your artifacts (model, training data, etc) into Azure Blob Storage. \n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "b5d64ecb",
   "metadata": {},
   "outputs": [],
   "source": [
    "import argparse\n",
    "import os\n",
    "from sklearn.linear_model import LinearRegression\n",
    "import pickle\n",
    "\n",
    "def parse_args():\n",
    "    \"\"\"\n",
    "    Parse arguments.\n",
    "    \"\"\"\n",
    "    parser = argparse.ArgumentParser()\n",
    "\n",
    "    # hyperparameters sent by the client are passed as command-line
arguments to the script.\n",
    "    # We don't use these but I left them in as a useful template for
future development\n",
    "    parser.add_argument(\"--copy_X\",        type=bool,
default=True)\n",
    "    parser.add_argument(\"--fit_intercept\", type=bool,
default=True)\n",
    "    \n",
    "    # Data directories\n",
    "    # In Azure, use Azure ML Datasets or paths in Azure Blob
Storage\n",
    "    # Here, we assume the paths are passed directly as arguments\n",
    "    parser.add_argument(\"--train\", type=str,
default=os.environ.get(\"AZUREML_DATAREFERENCE_train\"))\n",
    "    parser.add_argument(\"--test\", type=str,
default=os.environ.get(\"AZUREML_DATAREFERENCE_test\"))\n",
    "\n",
```

```
    "    # Model directory: in Azure, typically set to './outputs'\n",
    "    # Model artifacts saved here can be registered to Azure ML
workspace after training\n",
    "    parser.add_argument(\"--model_dir\", type=str,
default='./outputs')\n",
    "\n",
    "    return parser.parse_known_args()"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "e5453765",
   "metadata": {},
   "source": [
    "### Important Arguments Explained (Don't copy code below)\n"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "50596316",
   "metadata": {},
   "source": [
    "In Azure Machine Learning, we will first review the training script
file and then describe how you can execute the training job through Azure
Machine Learning Studio’s Jupyter Notebooks. Begin your training script by
importing necessary Python packages, followed by defining a function that
parses arguments passed to the script. A crucial argument here is
model_dir, which is a string representing the local path in the Azure
environment where the training job writes the model artifacts. Typically,
this path is set to ./outputs in Azure. After training, artifacts in this
directory can be registered to Azure ML workspace for model hosting or
directly uploaded to Azure Blob Storage. Another important argument to
consider is train_data_dir, representing the location of your training
data. In Azure, this would typically be a path in Azure Blob Storage or an
Azure ML Dataset, where your data is stored and accessed during the .fit
method in your training script."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "fce29900",
   "metadata": {},
   "outputs": [],
   "source": [
    "# model_dir \n",
```

```
    "# - The most important argument. \n",
    "# - File Path where training job writes the model artifacts to.\n",
    "\n",
    "# - AZURE_MODEL_DIR - ALWAYS SET TO \"./outputs\"\n",
    "#                 - Azure looks for saves the artifacts in this
directory after training. Artifacts in this directory can be registered to
Azure ML workspace for model hosting or directly uploaded to Azure Blob
Storage.\n",
    "parser.add_argument(\"--model_dir\", type=str,
default='./outputs')\n",
    "    \n",
    "# train\n",
    "# - Stores the location of the training data.\n",
    "# - Stores the Azure Blob Storage string to your training data that
you passed when calling the .fit function.\n",
    "parser.add_argument(\"--train\", type=str,
default=os.environ.get(\"AZUREML_DATAREFERENCE_train\"))"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "3a366d77",
   "metadata": {},
   "source": [
    "### 2. Load Dataset"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "09423794",
   "metadata": {},
   "source": [
    "After creating arguments we want to load our training data onto the
instance using the train argument that we have.\n",
    "\n",
    "So, we will create functions to load training (and optionally testing)
data. The way you define these functions\n",
    "depend on how your data is stored but it should be fairly easy to
create your own version. Here the argument\n",
    "path is the location of the Azure Blob Storage or Azure ML Dataset
containing your training (and test) data. If you do not have offline
training\n",
    "data and want to import data from a repository you can do so by simply
ignoring the argument and directly\n",
    "importing your data."
   ]
```

```
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "28f51e63",
   "metadata": {},
   "outputs": [],
   "source": [
    "def load_dataset(path):\n",
    "    \"\"\"\n",
    "    Load entire dataset.\n",
    "    \"\"\"\n",
    "    # Find all files with a pickle ext but we only load the first one
in this sample:\n",
    "    files = [os.path.join(path, file) for file in os.listdir(path) if
file.endswith(\"pickle\")]\n",
    "\n",
    "    if len(files) == 0:\n",
    "        raise ValueError(\"Invalid # of files in dir:
{}\".format(path))\n",
    "    \n",
    "    [X, y] = pickle.load(open(files[0], 'rb'))\n",
    "    \n",
    "    return X, y"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "7f84b01f",
   "metadata": {},
   "source": [
    "### 3) Load Model"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "5ce7b179",
   "metadata": {},
   "source": [
    "We want to have a function that stores and returns a regression model
"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 6,
```

```
   "id": "7a12675d",
   "metadata": {},
   "outputs": [],
   "source": [
    "def model_fn(model_dir):\n",
    "    \"\"\"\n",
    "    Load the model for inference\n",
    "    \"\"\"\n",
    "    loaded_model = pickle.load(open(model_dir + \"/model.pickle\",
'rb'))\n",
    "    return loaded_model"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "afaf0146",
   "metadata": {},
   "source": [
    "### 4) Predict With Model"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "5a6259ee",
   "metadata": {},
   "source": [
    "We will use a prediction function for future inferences. "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "d94ccc6a",
   "metadata": {},
   "outputs": [],
   "source": [
    "def predict_fn(input_data, model):\n",
    "    \"\"\"\n",
    "    Apply model to the incoming request\n",
    "    \"\"\"\n",
    "    return model.predict(input_data)"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "8650976b",
```

```
   "metadata": {},
   "source": [
    "The training script will have a main function that will combine all of these functions and train the model. <br/><br/>\n",
    "When you launch the training job through this script in Azure, whatever is executed in the script constitutes the training job. This allows for extensive customization according to your requirements. <br/><br/>However, it's important to note that to save and access outputs, like your trained model, outside of this training job in Azure, you must utilize the model_dir argument. In Azure, this is typically set to ./outputs, and this is where the script should save the trained model. When running the training script through Azure Machine Learning Studio’s Jupyter Notebooks, you define this directory. Azure Machine Learning will then automatically handle the artifacts in this directory for later use, such as for model registration or deployment."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "2b24aab2-294d-4562-a079-f7c74874d0b3",
   "metadata": {},
   "outputs": [],
   "source": [
    "if __name__ == \"__main__\":\n",
    "    \n",
    "    args, _ = parse_args()\n",
    "    \n",
    "    \"\"\"\n",
    "    Train a Linear Regression\n",
    "    \"\"\"\n",
    "    print(\"Training mode\")\n",
    "\n",
    "    try:\n",
    "        X_train, y_train = load_dataset(args.train)\n",
    "        # X_test, y_test = load_dataset(args.test)  # Uncomment if test data is available\n",
    "\n",
    "        hyperparameters = {\n",
    "            \"copy_X\": args.copy_X,\n",
    "            \"fit_intercept\": args.fit_intercept\n",
    "        }\n",
    "        \n",
    "        print(\"Training...\")\n",
    "        model = LinearRegression()\n",
    "        model.set_params(**hyperparameters)\n",
```

```
    "                \n",
    "        model.fit(X_train, y_train)\n",
    "\n",
    "        # Save the model to the outputs directory (Azure ML
specific)\n",
    "        pickle.dump(model, open(os.path.join(args.model_dir,
\"model.pickle\"), 'wb'))\n",
    "        print(\"Model saved in: \" + os.path.join(args.model_dir,
\"model.pickle\"))\n",
    "\n",
    "       \n",
    "\n",
    "    except Exception as e:\n",
    "        # Write out an error file. This can be useful for debugging in
Azure\n",
    "        trc = traceback.format_exc()\n",
    "        with open(os.path.join(args.model_dir, \"failure\"), \"w\") as
s:\n",
    "            s.write(\"Exception during training: \" + str(e) +
\"\\\\n\" + trc)\n",
    "\n",
    "        # Printing this causes the exception to be in the training job
logs\n",
    "        print(\"Exception during training: \" + str(e) + \"\\\\n\" +
trc, file=sys.stderr)\n",
    "\n",
    "        # A non-zero exit code causes the training job to be marked as
Failed\n",
    "        sys.exit(255)"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "329ee251",
   "metadata": {},
   "source": [
    "## Step 2.3 Creating Deployment Script"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "549ca5a2",
   "metadata": {},
   "source": [
    "### 1) Prepare the notebook to run the training job\n",
    "- In your Azure Machine Learning Jupyter Notebook, first specify the
```

```
location of the training script. If your script is named script.py and is located in a folder named scripts on your notebook instance, the path would be /scripts/script.py.\n",
    "- You have the option to pass additional arguments, such as hyperparameters, to your training script. Define these in your notebook as needed.\n",
    "- Create a training environment in Azure, similar to an EC2 instance in SageMaker. In Azure, this involves setting up a ScriptRunConfig or an Estimator, specifying the Azure compute resource (VM size) for training.\n",
    "- After configuring your ScriptRunConfig or Estimator, initiate the training job. This is done by calling the .submit() method on your Experiment object, analogous to the .fit command in SageMaker. This method should pass the training data location, usually a path in Azure Blob Storage or a reference to an Azure ML Dataset. This data location is fed into the training script, akin to the args.train parameter in SageMaker.\n",
    "- If your training script is set to import data directly from an online source or a repository, you can omit the data path argument and handle data loading within the script."
   ]
  },
  {
   "cell_type": "markdown",
   "id": "2cab67c3",
   "metadata": {},
   "source": [
    "### <span style=\"color:red\">You copy the code below combine all of these scripts together to create one deployment notebook: <br/> <b>regressionDeployment.ipynb </b></span>\n",
    "\n",
    "Deployment Scripts can be broken down into 7 distinct parts. \n",
    "1) Creating Initial Variables\n",
    "2) Upload Training Data to Azure Blob Storage\n",
    "3) Create Hyperparameters\n",
    "4) Estimator Parameters\n",
    "5) Running the Training Job\n",
    "6) Deploying the Model to an Endpoint\n",
    "7) Deleting The Endpoint"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "78659b46",
   "metadata": {},
   "source": [
```

```
    "### 1) Creating Initial Variables"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "da0339e1",
   "metadata": {},
   "source": [
    "#### 1.1) Create some variables that will be used through this
process:"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "4b65e5de",
   "metadata": {},
   "outputs": [],
   "source": [
    "from azureml.core import Workspace, Experiment, ScriptRunConfig,
Environment\n",
    "from azureml.core.dataset import Dataset\n",
    "from azureml.data.dataset_factory import FileDatasetFactory\n",
    "\n",
    "# Connect to your Azure ML Workspace\n",
    "workspace = Workspace.from_config()\n",
    "\n",
    "datastore = workspace.get_default_datastore()\n",
    "\n",
    "# Define paths for data in Azure Blob Storage\n",
    "datastore_path = \"script-mode-workflow/pickle\"\n",
    "train_data_path = os.path.join(datastore_path, \"train\")\n",
    "# Assuming your file is in a local directory named \"train_dir\" that
contains \"train.pickle\"\n",
    "local_train_dir = os.path.join(os.getcwd(), \"train_dir\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "7e2eade6",
   "metadata": {},
   "source": [
    "#### 1.2) Create the conda dependency file for our script to access
the necessary libraries."
   ]
  },
```

```
  {
   "cell_type": "markdown",
   "id": "e3c0b115",
   "metadata": {},
   "source": [
    "```\n",
    "name: sklearn-env\n",
    "channels:\n",
    "  - defaults\n",
    "dependencies:\n",
    "  - python=3.8\n",
    "  - scikit-learn\n",
    "  - pandas\n",
    "  - numpy\n",
    "  - pip\n",
    "  - pip:\n",
    "      - azureml-defaults\n",
    "```"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "28b1dbd2",
   "metadata": {},
   "source": [
    "### 2) Upload Training Data to Azure Blob Storage"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "e40a1500",
   "metadata": {},
   "source": [
    "In Azure, we want to upload the training data to Azure Blob Storage,
so that it's available for Azure Machine Learning training. "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "2dc9abf2",
   "metadata": {},
   "outputs": [],
   "source": [
    "# Upload the entire directory to the datastore\n",
    "FileDatasetFactory.upload_directory(src_dir=local_train_dir,\n",
```

```
    "                                       target=(datastore,
train_data_path),\n",
    "                                       overwrite=True)\n",
    "\n",
    "# Create a FileDataset pointing to the uploaded directory/files\n",
    "train_dataset = Dataset.File.from_files(path=(datastore,
train_data_path))\n",
    "\n",
    "# This path can be used to access the dataset during training\n",
    "datastore_train_path = train_dataset.as_mount()"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "140a2c07",
   "metadata": {},
   "source": [
    "### 3) Create Hyperparameters"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "2d736a78",
   "metadata": {},
   "outputs": [],
   "source": [
    "# This is not required as these values are the defaults:\n",
    "hyperparameters = {\n",
    "    \"--copy_X\": True,\n",
    "    \"--fit_intercept\": True,\n",
    "}"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "0f5b49e0",
   "metadata": {},
   "source": [
    "### 4) Estimator Parameters"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "6d1cc368",
   "metadata": {},
```

```
   "source": [
    "The Azure Machine Learning Estimator object is a high-level interface for Azure Machine Learning training. This object represents the algorithm, the data, and other configurations. \n",
    "\n",
    "If you are interested, you can read more about Azure Machine Learning estimators in the official Azure Machine Learning documentation.\n",
    "\n",
    "Define the environment and create a script run configuration"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "32dff6e1",
   "metadata": {},
   "outputs": [],
   "source": [
    "env = Environment.from_conda_specification(name=\"sklearn-env\", file_path=\"./conda_dependency_file.yml\")\n",
    "arguments = [f'{k}={v}' for k, v in hyperparameters.items()]\n",
    "arguments.extend(['--train', datastore_train_path])\n",
    "\n",
    "# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Modify this based on your script name !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n",
    "src = ScriptRunConfig(source_directory='./script',\n",
    "                      script='script.py',\n",
    "                      arguments=arguments,\n",
    "                      compute_target='local',  # Using the local compute (the Compute Instance)\n",
    "                      environment=env)\n",
    "\n",
    "\n",
    "experiment = Experiment(workspace, 'linearregression-experiment')"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "c0de4233",
   "metadata": {},
   "source": [
    "### 5) Running the Training Job\n",
    "When we submit the experiment with experiment.submit(src) in Azure Machine Learning, it initiates the training process by provisioning the necessary compute resources, which could be either a local compute instance or a remote compute cluster, depending on the configuration. The training
```

```
script and any specified data are then automatically transferred to the selected compute target. Once everything is in place, the training job begins execution, separate from the Jupyter Notebook server. You can monitor the progress of the training job directly from the Azure ML Studio interface, where detailed logs and metrics are available. Additionally, the output of the run.wait_for_completion(show_output=True) command in the Jupyter Notebook provides real-time logs streamed from the training job, giving you insights into the training process as it happens."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "aaa8d8f2",
   "metadata": {},
   "outputs": [],
   "source": [
    "# starting the training job\n",
    "run = experiment.submit(src)\n",
    "run.wait_for_completion(show_output=True)"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "f411616f",
   "metadata": {},
   "source": [
    "### 6) Deploying the Model to an Endpoint\n",
    "Deploying a machine learning model to an endpoint is a crucial step in making it available for real-world usage. In Azure Machine Learning, an endpoint acts as a scalable platform for hosting your trained model, allowing you to make predictions from new data.\n",
    "\n",
    "(Make sure that the 'endpoint_name' used is not currently running.)"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "7c37209d",
   "metadata": {},
   "source": [
    "We need to create an inference before we deploy our model to an endpoint. For this we need to create a score.py file which contains the following code:\n",
    "\n",
    "```python   \n",
```

```
    "import json\n",
    "import joblib\n",
    "from azureml.core.model import Model\n",
    "\n",
    "def init():\n",
    "    global model\n",
    "    # Retrieve the path to the model file\n",
    "    model_path = Model.get_model_path('linearregression-model')  #
Replace with your model name\n",
    "    # Load the model\n",
    "    model = joblib.load(model_path)\n",
    "\n",
    "def run(raw_data):\n",
    "    data = json.loads(raw_data)\n",
    "    # Perform prediction using the loaded model\n",
    "    result = model.predict(data['data'])\n",
    "    return result.tolist()\n",
    "```\n",
    "Now in the main juptyer notebook we can deploy our model to an
endpoint."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "c2e7dfe5",
   "metadata": {},
   "outputs": [],
   "source": [
    "from azureml.core.model import InferenceConfig\n",
    "from azureml.core.webservice import AciWebservice\n",
    "from azureml.core.model import Model\n",
    "\n",
    "# Assuming you have a score.py file for inference\n",
    "inference_config = InferenceConfig(entry_script='score.py',
environment=env)\n",
    "# Set the ACI configuration\n",
    "aci_config = AciWebservice.deploy_configuration(cpu_cores=1,
memory_gb=1)\n",
    "\n",
    "# print(Model.get_model_path('linearregression-experiment'))\n",
    "model = Model(workspace, 'linearregression-model')\n",
    "# Deploy the model as a web service\n",
    "service = Model.deploy(workspace=workspace,\n",
    "                       name='linearregression-endpoint',\n",
    "                       models=[model],\n",
```

```
  "                       inference_config=inference_config, \n",
  "                       deployment_config=aci_config)\n",
  "service.wait_for_deployment(show_output=True)"
 ]
},
{
 "cell_type": "markdown",
 "id": "d10c84bb",
 "metadata": {},
 "source": [
  "Once completed, you can test the endpoint as follows:"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "36df7daa",
 "metadata": {},
 "outputs": [],
 "source": [
  "import json\n",
  "# Prepare a sample input for the model\n",
  "test_sample = json.dumps({'data': [[0],[1],[2],[3]]})\n",
  "test_sample = bytes(test_sample, encoding='utf8')\n",
  "\n",
  "# Use the web service to make predictions\n",
  "prediction = service.run(input_data=test_sample)\n",
  "print(prediction)"
 ]
},
{
 "cell_type": "markdown",
 "id": "ec0a73ee",
 "metadata": {},
 "source": [
  "### 7) Deleting The Endpoint"
 ]
},
{
 "cell_type": "markdown",
 "id": "a79fc5a4",
 "metadata": {},
 "source": [
  "Running this cell will remove the endpoint and configuration:"
 ]
},
```

```
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "1b1ed7cb",
   "metadata": {},
   "outputs": [],
   "source": [
    "service.delete()"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "2d71203a",
   "metadata": {},
   "source": [
    "## Step 3.0 Image recognition using Azure ML notebook\n",
    "\n",
    "Now you are able to use Azure Notebooks to run and deploy any machine learning application. For simplicity, you can consider the image recognition program that you used in Assignment 3, and apply steps 2 and 3 to develop a training script and a Jupyter notebook for running and deployment of this program in the cloud."
   ]
  },
  {
   "cell_type": "markdown",
   "id": "0aaf557b",
   "metadata": {},
   "source": [
    "## Step 3.0 Image Recognition using Azure Machine Learning Notebook\n",
    "\n",
    "Now you are able to simply use the Azure Machine Learning Notebook to run and deploy any machine learning application. For simplicity, you can consider the image recognition program that you used in Assignment 3, and apply steps 2 and 3 to develop a training script and a Jupyter notebook for running and deploying this program in the cloud.\n",
    "\n",
    "The general steps you should follow are:\n",
    "1) Download CIFAR 10 Data to Azure Notebooks. \n",
    "\n",
    "<br/>\n",
    "\n",
    "2) Suggested File Structure:\n",
    "\n",
    "    ``` bash\n",
```

```
    "    --potentialTrainingDataFile-- (file)\n",
    "    cnnDeployment.ipynb (file)\n",
    "    conda_dependency_file.yml (file)\n",
    "    score.py (file)\n",
    "    \n",
    "\n",
    "    scripts (folder)\n",
    "        script.py \n",
    "\n",
    "    \n",
    "\n",
    "    cifar-10-batches-py (folder)\n",
    "        data_batch_1 (file)\n",
    "        data_batch_2 (file)\n",
    "        ...\n",
    "        test_batch   (file)\n",
    "    ```\n",
    "\n",
    "\n",
    "<br/>\n",
    "\n",
    "3) Script Setup: Utilize the script you created in Step 2.2 and make
modification to it based on the new model. \n",
    "\n",
    "<br/>\n",
    "\n",
    "4) Notebook Setup: Utilize the notebook script from Step 2.3 and make
modification to it based on your new model.\n"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "0924ae88",
   "metadata": {},
   "source": [
    "Sure, here are the Azure equivalents for the helper functions you
provided:\n",
    "\n",
    "```python\n",
    "# CODE HELPER 1 - notice how data is saved -> training.cnn format\n",
    "def pickleTrainingData():\n",
    "    def unpickle(file):\n",
    "        with open(file, 'rb') as fo:\n",
    "            dict = pickle.load(fo, encoding='bytes')\n",
    "        return dict\n",
    "\n",
```

```
    "    train_data = np.empty((0, 32*32*3))\n",
    "    train_labels = []\n",
    "\n",
    "    for i in range(1, 2):\n",
    "        fileNameDataBatch = './cifar-10-batches-py/data_batch_' +
str(i)\n",
    "        batch = unpickle(fileNameDataBatch)\n",
    "        train_data = np.vstack((train_data, batch[b'data']))\n",
    "        train_labels += batch[b'labels']\n",
    "\n",
    "    train_labels = np.array(train_labels)\n",
    "    train_data = train_data.reshape(-1, 32, 32, 3) / 255.0\n",
    "    \n",
    "    # !!!!! NOTICE HOW THE DATA IS SAVED !!!!!!\n",
    "    # Will be returned in form of:\n",
    "    # train_label, train_data  = getDataBack()\n",
    "    pickle.dump([train_labels,train_data], open('./train.cnn',
'wb'))\n",
    "\n",
    "pickleTrainingData()\n",
    "\n",
    "# CODE HELPER 2\n",
    "def getTestData():\n",
    "    def unpickle(file):\n",
    "        with open(file, 'rb') as fo:\n",
    "            dict = pickle.load(fo, encoding='bytes')\n",
    "        return dict\n",
    "    fileNameTestBatch = './cifar-10-batches-py/test_batch'\n",
    "    test_data = unpickle(fileNameTestBatch)[b'data']\n",
    "    test_data = test_data.reshape(-1, 32, 32, 3) / 255.0\n",
    "    test_labels = np.array(unpickle(fileNameTestBatch)[b'labels'])\n",
    "    \n",
    "    num_samples_to_select = 600\n",
    "    random_indices = np.random.choice(test_data.shape[0],
num_samples_to_select, replace=False)\n",
    "    selected_test_data = test_data[random_indices]\n",
    "    selected_test_labels = test_labels[random_indices]\n",
    "    \n",
    "    return selected_test_data, selected_test_labels\n",
    "\n",
    "test_data, test_labels = getTestData()\n",
    "\n",
    "# CODE HELPER 3\n",
    "from sklearn.metrics import accuracy_score\n",
    "def getAccuracyOfPrediction(cnn_predictions, test_labels):\n",
    "    cnn_predicted_labels = np.argmax(cnn_predictions, axis=1)\n",
```

```
    "    accuracy = accuracy_score(test_labels, cnn_predicted_labels)\n",
    "    print(\"Accuracy:\", accuracy)\n",
    "\n",
    "\n",
    "# CODE HELPER 4\n",
    "import argparse\n",
    "import os\n",
    "import pickle\n",
    "import subprocess\n",
    "subprocess.run([\"pip\", \"install\", \"Werkzeug==2.0.3\"])\n",
    "subprocess.run([\"pip\", \"install\", \"tensorflow==2.4\"])\n",
    "import tensorflow as tf\n",
    "from tensorflow import keras\n",
    "\n",
    "# CODE HELPER 5 -> notice that the file name ends in .cnn\n",
    "files = [os.path.join(path, file) for file in os.listdir(path) if
file.endswith(\".cnn\")]\n",
    "\n",
    "# CODE HELPER 6\n",
    "loaded_model = tf.keras.models.load_model(os.path.join(model_dir,
\"modelCNN\"))\n",
    "\n",
    "# CODE HELPER 7\n",
    "model = keras.Sequential([\n",
    "    keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu',
input_shape=(32, 32, 3)),\n",
    "    keras.layers.MaxPooling2D(pool_size=2),\n",
    "    keras.layers.Conv2D(filters=64, kernel_size=3,
activation='relu'),\n",
    "    keras.layers.MaxPooling2D(pool_size=2),\n",
    "    keras.layers.Flatten(),\n",
    "    keras.layers.Dense(units=128, activation='relu'),\n",
    "    keras.layers.Dense(units=10, activation='softmax')\n",
    "])\n",
    "model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])\n",
    "\n",
    "model.fit(train_data, train_labels, epochs=3, batch_size=32,
validation_split=0.1)\n",
    "\n",
    "model.save(os.path.join(args.model_dir, \"modelCNN\"))\n",
    "```"
   ]
  },
  {
   "cell_type": "markdown",
```

```
   "id": "d9238132",
   "metadata": {},
   "source": [
    "## Deliverables:\n",
    "\n",
    "Upon completing this assignment, you need to submit the following:\n",
    "\n",
    "For the image recognition program:\n",
    "- Training scripts and notebooks\n",
    "- Accuracy result of testing your image recognition program.  \n",
    "- The total cost of running the program\n",
    "- The time taken to train, deploy and predict with Azure Notebook"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "07e57431",
   "metadata": {},
   "source": [
    "# Good Luck!"
   ]
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3 (ipykernel)",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.11.1"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```