Lab 3.5 - Student Notebook

Overview

This lab is a continuation of the guided labs in Module 3.

In this lab, you will deploy a trained model and perform a prediction against the model. You will then delete the endpoint and perform a batch transform on the test dataset.

Introduction to the business scenario

You work for a healthcare provider, and want to improve the detection of abnormalities in orthopedic patients.

You are tasked with solving this problem by using machine learning (ML). You have access to a dataset that contains six biomechanical features and a target of *normal* or *abnormal*. You can use this dataset to train an ML model to predict if a patient will have an abnormality.

About this dataset

This biomedical dataset was built by Dr. Henrique da Mota during a medical residence period in the Group of Applied Research in Orthopaedics (GARO) of the Centre Médico-Chirurgical de Réadaptation des Massues, Lyon, France. The data has been organized in two different, but related, classification tasks.

The first task consists in classifying patients as belonging to one of three categories:

- *Normal* (100 patients)
- *Disk Hernia* (60 patients)
- *Spondylolisthesis* (150 patients)

For the second task, the categories *Disk Hernia* and *Spondylolisthesis* were merged into a single category that is labeled as *abnormal*. Thus, the second task consists in classifying patients as belonging to one of two categories: *Normal* (100 patients) or *Abnormal* (210 patients).

Attribute information

Each patient is represented in the dataset by six biomechanical attributes that are derived from the shape and orientation of the pelvis and lumbar spine (in this order):

- Pelvic incidence
- Pelvic tilt
- Lumbar lordosis angle
- Sacral slope
- Pelvic radius
- Grade of spondylolisthesis

The following convention is used for the class labels:

- DH (Disk Hernia)
- Spondylolisthesis (SL)
- Normal (NO)
- Abnormal (AB)

For more information about this dataset, see the Vertebral Column dataset webpage.

Dataset attributions

This dataset was obtained from: Dua, D. and Graff, C. (2019). UCI Machine Learning Repository (http://archive.ics.uci.edu/ml). Irvine, CA: University of California, School of Information and Computer Science.

Lab setup

Because this solution is split across several labs in the module, you run the following cells so that you can load the data and train the model to be deployed.

Note: The setup can take up to 5 minutes to complete.

Importing the data

By running the following cells, the data will be imported and ready for use.

Note: The following cells represent the key steps in the previous labs.

In [1]:
```
bucket='c169682a4380821l11159312t1w168507467552-labbucket-xhj1ja67irhd'
```

In [2]:
```
import warnings, requests, zipfile, io
warnings.simplefilter('ignore')
import pandas as pd
from scipy.io import arff

import os
import boto3
import sagemaker
from sagemaker.image_uris import retrieve
from sklearn.model_selection import train_test_split
```

```
sagemaker.config INFO - Not applying SDK defaults from location: /etc/xdg/sagem
aker/config.yaml
sagemaker.config INFO - Not applying SDK defaults from location: /home/ec2-use
r/.config/sagemaker/config.yaml
```

In [3]:
```
f_zip = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00212/verteb
r = requests.get(f_zip, stream=True)
Vertebral_zip = zipfile.ZipFile(io.BytesIO(r.content))
Vertebral_zip.extractall()

data = arff.loadarff('column_2C_weka.arff')
df = pd.DataFrame(data[0])

class_mapper = {b'Abnormal':1,b'Normal':0}
df['class']=df['class'].replace(class_mapper)

cols = df.columns.tolist()
cols = cols[-1:] + cols[:-1]
df = df[cols]

train, test_and_validate = train_test_split(df, test_size=0.2, random_state=42
test, validate = train_test_split(test_and_validate, test_size=0.5, random_sta

prefix='lab3'

train_file='vertebral_train.csv'
test_file='vertebral_test.csv'
validate_file='vertebral_validate.csv'

s3_resource = boto3.Session().resource('s3')
def upload_s3_csv(filename, folder, dataframe):
    csv_buffer = io.StringIO()
    dataframe.to_csv(csv_buffer, header=False, index=False )
    s3_resource.Bucket(bucket).Object(os.path.join(prefix, folder, filename)).

upload_s3_csv(train_file, 'train', train)
upload_s3_csv(test_file, 'test', test)
upload_s3_csv(validate_file, 'validate', validate)
```

```
container = retrieve('xgboost',boto3.Session().region_name,'1.0-1')

hyperparams={"num_round":"42",
             "eval_metric": "auc",
             "objective": "binary:logistic"}

s3_output_location="s3://{}/{}/output/".format(bucket,prefix)
xgb_model=sagemaker.estimator.Estimator(container,
                                       sagemaker.get_execution_role(),
                                       instance_count=1,
                                       instance_type='ml.m4.xlarge',
                                       output_path=s3_output_location,
                                        hyperparameters=hyperparams,
                                        sagemaker_session=sagemaker.Session())

train_channel = sagemaker.inputs.TrainingInput(
    "s3://{}/{}/train/".format(bucket,prefix,train_file),
    content_type='text/csv')

validate_channel = sagemaker.inputs.TrainingInput(
    "s3://{}/{}/validate/".format(bucket,prefix,validate_file),
    content_type='text/csv')

data_channels = {'train': train_channel, 'validation': validate_channel}

xgb_model.fit(inputs=data_channels, logs=False)

print('ready for hosting!')
```

```
INFO:sagemaker:Creating training-job with name: sagemaker-xgboost-2025-08-09-0
8-28-09-658
2025-08-09 08:28:10 Starting - Starting the training job..
2025-08-09 08:28:25 Starting - Preparing the instances for training....
2025-08-09 08:28:51 Downloading - Downloading input data....
2025-08-09 08:29:16 Downloading - Downloading the training image..........
2025-08-09 08:30:12 Training - Training image download completed. Training in p
rogress....
2025-08-09 08:30:32 Uploading - Uploading generated training model.
2025-08-09 08:30:45 Completed - Training job completed
ready for hosting!
```

Step 1: Hosting the model

Now that you have a trained model, you can host it by using Amazon SageMaker hosting services.

The first step is to deploy the model. Because you have a model object, *xgb_model*, you can use the **deploy** method. For this lab, you will use a single ml.m4.xlarge instance.

In [21]:
```
xgb_predictor = xgb_model.deploy(initial_instance_count=1,
                 serializer = sagemaker.serializers.CSVSerializer(),
                 instance_type='ml.m4.xlarge')
```

```
INFO:sagemaker:Creating model with name: sagemaker-xgboost-2025-08-09-08-51-2
8-752
INFO:sagemaker:Creating endpoint-config with name sagemaker-xgboost-2025-08-0
9-08-51-28-752
INFO:sagemaker:Creating endpoint with name sagemaker-xgboost-2025-08-09-08-51-2
8-752
------!
```

Step 2: Performing predictions

Now that you have a deployed model, you will run some predictions.

First, review the test data and re-familiarize yourself with it.

In [5]:
```
test.shape
```

Out[5]: (31, 7)

You have 31 instances, with seven attributes. The first five instances are:

In [6]:
```
test.head(5)
```

Out[6]:

	class	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pel
136	1	88.024499	39.844669	81.774473	48.179830	1
230	0	65.611802	23.137919	62.582179	42.473883	1
134	1	52.204693	17.212673	78.094969	34.992020	1
130	1	50.066786	9.120340	32.168463	40.946446	
47	1	41.352504	16.577364	30.706191	24.775141	1

You don't need to include the target value (class). This predictor can take data in the comma-separated values (CSV) format. You can thus get the first row *without the class column* by using the following code:

`test.iloc[:1,1:]`

The **iloc** function takes parameters of [*rows*,*cols*]

To only get the first row, use `0:1`. If you want to get row 2, you could use `1:2`.

To get all columns *except* the first column (*col 0*), use `1:`

In [7]:
```
row = test.iloc[0:1,1:]
row.head()
```

Out[7]:

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_rad
136	88.024499	39.844669	81.774473	48.17983	116.601

You can convert this to a comma-separated values (CSV) file, and store it in a string buffer.

In [8]:
```
batch_X_csv_buffer = io.StringIO()
row.to_csv(batch_X_csv_buffer, header=False, index=False)
test_row = batch_X_csv_buffer.getvalue()
print(test_row)
```

```
88.0244989,39.84466878,81.77447308,48.17983012,116.6015376,56.76608323
```

Now, you can use the data to perform a prediction.

In [9]:
```
xgb_predictor.predict(test_row)
```

Out[9]:
```
b'0.9966071844100952'
```

The result you get isn't a *0* or a *1*. Instead, you get a *probability score*. You can apply some conditional logic to the probability score to determine if the answer should be presented as a 0 or a 1. You will work with this process when you do batch predictions.

For now, compare the result with the test data.

In [10]:
```
test.head(5)
```

Out[10]:

	class	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pel
136	1	88.024499	39.844669	81.774473	48.179830	1
230	0	65.611802	23.137919	62.582179	42.473883	1
134	1	52.204693	17.212673	78.094969	34.992020	1
130	1	50.066786	9.120340	32.168463	40.946446	
47	1	41.352504	16.577364	30.706191	24.775141	1

Question: Is the prediction accurate?

Challenge task: Update the previous code to send the second row of the dataset. Are those predictions correct? Try this task with a few other rows.

It can be tedious to send these rows one at a time. You could write a function to submit these values in a batch, but SageMaker already has a batch capability. You will examine that feature next. However, before you do, you will terminate the model.

In [25]:
```
test_row_2 = test.iloc[1].drop('class')
payload_2 = ','.join(map(str, test_row_2.values))
response_2 = xgb_predictor.predict(payload_2)
print("Prediction for 2nd row:", response_2)
print("Actual class for 2nd row:", test.iloc[1]['class'])
```

```
Prediction for 2nd row: b'0.777283251285553'
Actual class for 2nd row: 0.0
```

Step 3: Terminating the deployed model

To delete the endpoint, use the **delete_endpoint** function on the predictor.

In [11]:
```
xgb_predictor.delete_endpoint(delete_endpoint_config=True)
```

```
INFO:sagemaker:Deleting endpoint configuration with name: sagemaker-xgboost-202
5-08-09-08-30-46-613
INFO:sagemaker:Deleting endpoint with name: sagemaker-xgboost-2025-08-09-08-3
0-46-613
```

Step 4: Performing a batch transform

When you are in the training-testing-feature engineering cycle, you want to test your holdout or test sets against the model. You can then use those results to calculate metrics. You could deploy an endpoint as you did earlier, but then you must remember to delete the endpoint. However, there is a more efficient way.

You can use the transformer method of the model to get a transformer object. You can then use the transform method of this object to perform a prediction on the entire test dataset. SageMaker will:

- Spin up an instance with the model
- Perform a prediction on all the input values
- Write those values to Amazon Simple Storage Service (Amazon S3)
- Finally, terminate the instance

You will start by turning your data into a CSV file that the transformer object can take as input. This time, you will use **iloc** to get all the rows, and all columns *except* the first column.

In [12]:
```
batch_X = test.iloc[:,1:];
```

```
batch_X.head()
```

Out[12]:

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_rad
136	88.024499	39.844669	81.774473	48.179830	116.601
230	65.611802	23.137919	62.582179	42.473883	124.128
134	52.204693	17.212673	78.094969	34.992020	136.972
130	50.066786	9.120340	32.168463	40.946446	99.712
47	41.352504	16.577364	30.706191	24.775141	113.266

Next, write your data to a CSV file.

In [13]:
```
batch_X_file='batch-in.csv'
upload_s3_csv(batch_X_file, 'batch-in', batch_X)
```

Last, before you perform a transform, configure your transformer with the input file, output location, and instance type.

In [14]:
```
batch_output = "s3://{}/{}/batch-out/".format(bucket,prefix)
batch_input = "s3://{}/{}/batch-in/{}".format(bucket,prefix,batch_X_file)

xgb_transformer = xgb_model.transformer(instance_count=1,
                                       instance_type='ml.m4.xlarge',
                                       strategy='MultiRecord',
                                       assemble_with='Line',
                                       output_path=batch_output)

xgb_transformer.transform(data=batch_input,
                         data_type='S3Prefix',
                         content_type='text/csv',
                         split_type='Line')
xgb_transformer.wait()
```

```
INFO:sagemaker:Creating model with name: sagemaker-xgboost-2025-08-09-08-34-4
9-176
INFO:sagemaker:Creating transform job with name: sagemaker-xgboost-2025-08-09-0
8-34-49-689
.......................................
...
```

After the transform completes, you can download the results from Amazon S3 and compare them with the input.

First, download the output from Amazon S3 and load it into a pandas DataFrame.

In [15]:
```
s3 = boto3.client('s3')
obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix,'batch-
target_predicted = pd.read_csv(io.BytesIO(obj['Body'].read()),sep=',',names=['
```

```
target_predicted.head(5)
```

Out[15]:

	class
0	0.996607
1	0.777283
2	0.994641
3	0.993690
4	0.939139

You can use a function to convert the probabilty into either a *0* or a *1*.

The first table output will be the *predicted values*, and the second table output is the *original test data*.

In [16]:

```
def binary_convert(x):
    threshold = 0.65
    if x > threshold:
        return 1
    else:
        return 0

target_predicted['binary'] = target_predicted['class'].apply(binary_convert)

print(target_predicted.head(10))
test.head(10)
```

```
      class  binary
0  0.996607       1
1  0.777283       1
2  0.994641       1
3  0.993690       1
4  0.939139       1
5  0.997396       1
6  0.991977       1
7  0.987518       1
8  0.993334       1
9  0.682776       1
```

Out[16]:

	class	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pel
136	1	88.024499	39.844669	81.774473	48.179830	1
230	0	65.611802	23.137919	62.582179	42.473883	1
134	1	52.204693	17.212673	78.094969	34.992020	1
130	1	50.066786	9.120340	32.168463	40.946446	
47	1	41.352504	16.577364	30.706191	24.775141	1
135	1	77.121344	30.349874	77.481083	46.771470	1
100	1	84.585607	30.361685	65.479486	54.223922	1
89	1	71.186811	23.896201	43.696665	47.290610	1
297	0	45.575482	18.759135	33.774143	26.816347	1
4	1	49.712859	9.652075	28.317406	40.060784	1

Note: The *threshold* in the **binary_convert** function is set to *.65*.

Challenge task: Experiment with changing the value of the threshold. Does it impact the results?

Note: The initial model might not be good. You will generate some metrics in the next lab, before you tune the model in the final lab.

In [24]:

```
import numpy as np
from sklearn.metrics import classification_report
X_test = test.drop('class', axis=1)
y_test = test['class']
predictions = xgb_predictor.predict(X_test.to_numpy().tolist())
if isinstance(predictions, (bytes, bytearray)):
    predictions = predictions.decode("utf-8")
pred_list = [float(x) for x in predictions.strip().split(",")]
y_pred_proba = np.array(pred_list)
def evaluate_thresholds(probabilities, y_true, thresholds=[0.5, 0.65, 0.8]):
    for thresh in thresholds:
        preds = (probabilities >= thresh).astype(int)
        acc = (preds == y_true).mean()
        print(f"Threshold: {thresh}, Accuracy: {acc:.4f
evaluate_thresholds(y_pred_proba, y_test)
```

```
Threshold: 0.5, Accuracy: 0.8387
Threshold: 0.65, Accuracy: 0.8065
Threshold: 0.8, Accuracy: 0.8065
```

Congratulations!

You have completed this lab, and you can now end the lab by following the lab guide instructions.