Lab 3.4 - Student Notebook

Overview

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

In this lab, you will split the data into three separate datasets:

- *Training Set* - This will be used to train the model.
- *Validation Set* - This will be used during training to validate the model.
- *Test Set* - This will be held back and used to produce metrics after the model is trained. You will use this dataset in an upcoming lab.

With the split data, you will train a XGBoost model by using Amazon SageMaker.

Introduction to the business scenario

You work for a healthcare provider, and want to improve 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 must run the following cells so that you can load the data.

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]:
```
import warnings, requests, zipfile, io
warnings.simplefilter('ignore')
import pandas as pd
from scipy.io import arff
import boto3
```

In [2]:
```
f_zip = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00212/vertebra
r = requests.get(f_zip, stream=True)
Vertebral_zip = zipfile.ZipFile(io.BytesIO(r.content))
Vertebral_zip.extractall()
```

In [3]:
```
data = arff.loadarff('column_2C_weka.arff')
df = pd.DataFrame(data[0])
```

In [4]:
```
class_mapper = {b'Abnormal':1,b'Normal':0}
df['class']=df['class'].replace(class_mapper)
```

Step 1: Exploring the data

You will start with a quick reminder of the data in the dataset.

To get the most out of this lab, carefully read the instructions and code before you run the cells. Take time to experiment!

First, use **shape** to examine the number of rows and columns.

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

Out[5]:
```
(310, 7)
```

Next, get a list of the columns.

In [6]:
```
df.columns
```

Out[6]:
```
Index(['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle',
       'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis', 'class'],
      dtype='object')
```

You can see the six biomechanical features, and that the target column is named *class*.

Step 2: Preparing the data

For this lab, you must split the data into three datasets.

An internet search will show many different ways to split datasets. Many code samples that you might find will split the dataset into the *target* and the *features*. Then, they will split each of those two datasets into three subsets, which results in a total of six datasets to track.

Moving the target column position

XGBoost requires the training data to be in a single file. The file must have the target value be the first column.

Get the target column and move it to the first position.

In [7]:
```
cols = df.columns.tolist()
cols = cols[-1:] + cols[:-1]
df = df[cols]
```

You should see that the **class** is now the first column.

In [8]:
```
df.columns
```

Out[8]:
```
Index(['class', 'pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle',
       'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis'],
      dtype='object')
```

Splitting the data

You will start by splitting the dataset into two datasets. You will use one dataset for training, and you will split the other dataset again for use with validation and testing.

You will use the *train_test_split function* from the *scikit-learn library*, which is a free machine learning library for Python. It has many algorithms and useful functions, such as the one you will use.

- For more information about the function, see the Train_test_split documentation.
- For more information about scikit-learn, see the scikit-learn guide

Because you don't have a lot of data, you want to make sure that the split datasets contain a representative amount of each class. Thus, you will use the *stratify* switch. Finally, you will use a random number so that you can repeat the splits.

In [9]:
```
from sklearn.model_selection import train_test_split
train, test_and_validate = train_test_split(df, test_size=0.2, random_state=42,
```

Next, split the *test_and_validate* dataset into two equal parts.

In [10]:
```
test, validate = train_test_split(test_and_validate, test_size=0.5, random_state
```

Examine the three datasets.

In [11]:
```
print(train.shape)
print(test.shape)
print(validate.shape)
```

```
(248, 7)
(31, 7)
(31, 7)
```

Now, check the distribution of the classes.

In [12]:
```
print(train['class'].value_counts())
print(test['class'].value_counts())
print(validate['class'].value_counts())
```

```
class
1    168
0     80
Name: count, dtype: int64
class
1    21
0    10
Name: count, dtype: int64
class
1    21
0    10
Name: count, dtype: int64
```

Uploading the data to Amazon S3

XGboost will load the data for training from Amazon Simple Storage Service (Amazon S3). Thus, you must write the data to a comma-separated values (CSV) file, and then upload the file to Amazon S3.

Start by setting up some variables to the S3 bucket, then create a function to upload the CSV file to Amazon S3. You can reuse this function.

First, explore the function.

Note the following line:

`dataframe.to_csv(csv_buffer, header=False, index=False)`

This line writes the pandas DataFrame (which was passed into the function) into the IO buffer that's named *csv_buffer*. You use a buffer because you don't need to write the file locally.

To stop the column headers from being written out, use `header=False`. To stop the pandas index from being output, use `index=False`.

To write the csv_buffer to Amazon S3 as an object, use the `put` operation on the `object`, which is a property of the `bucket`.

In [13]:
```
bucket='c188535a4866696l13672677t1w972348242892-labbucket-gqrnm4kwaweb'

prefix='lab3'

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

import os

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)).pu
```

Use the function that you created to upload the three datasets.

In [14]:
```
upload_s3_csv(train_file, 'train', train)
upload_s3_csv(test_file, 'test', test)
upload_s3_csv(validate_file, 'validate', validate)
```

Step 3: Training the model

Now that the data in Amazon S3, you can train a model.

The first step is to get the XGBoost container URI.

In [15]:
```
import boto3
from sagemaker.image_uris import retrieve
container = retrieve('xgboost',boto3.Session().region_name,'1.0-1')
```

```
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
```

Next, you must set some *hyperparameters* for the model. Because this is the first time you are training the model, you can use some values to get started.

In [16]:
```
hyperparams={"num_round":"42",
             "eval_metric": "auc",
             "objective": "binary:logistic"}
```

Use the **estimator** function to set up the model. Here are a few parameters of interest:

- **instance_count** - This defines how many instances will be used for training. You will use *one* instance.
- **instance_type** - This defines the instance type for training. In this case, it's *ml.m4.xlarge*.

In [17]:
```
import sagemaker
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())
```

```
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
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
```

The estimator needs *channels* to feed data into the model. For training, the *train_channel* and *validate_channel* will be used.

In [18]:
```
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}
```

Running **fit** will train the model.

Note: This process can take up to 5 minutes.

In [19]:
```
xgb_model.fit(inputs=data_channels, logs=False)
```

```
INFO:sagemaker:Creating training-job with name: sagemaker-xgboost-2026-02-03-13
-46-04-529
2026-02-03 13:46:06 Starting - Starting the training job..
2026-02-03 13:46:21 Starting - Preparing the instances for training...
2026-02-03 13:46:43 Downloading - Downloading input data......
2026-02-03 13:47:18 Downloading - Downloading the training image.........
2026-02-03 13:48:09 Training - Training image download completed. Training in p
rogress....
2026-02-03 13:48:29 Uploading - Uploading generated training model..
2026-02-03 13:48:43 Completed - Training job completed
```

After the training is complete, you are ready to test and evaluate the model. However, you will do testing and validation in later labs.

Congratulations!

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