## Lab 2-01: Algorithms

### Introduction

XGBoost

The XGBoost (eXtreme Gradient Boosting) version of the gradient boosted trees method is a popular and efficient open-source implementation. Gradient boosting is a supervised learning approach for effectively predicting a target variable by combining an ensemble of estimates from a series of simpler and weaker models. Because of its strong handling of a range of data kinds, relationships, and distributions, as well as the diversity of hyperparameters that can be fine-tuned, the XGBoost method does well in machine learning contests. XGBoost may be used to solve issues involving regression, classification (binary and multi-class), and ranking.

Linear Learner Algorithm

Linear models have supervised learning algorithms for regression, binary classification, or multi-class classification problems. You give the model labels (x,y) with x being a high dimensional vector and y numeric. The algorithm learns a linear function, or, for classification problems, a linear threshold function and maps a vector x to an approximation of label y. Both classification and regression problems may be solved with Amazon SageMaker's Linear Learner algorithm.

### Case Study Enterprise Financial Service– Kasasa

Background

Community banks and credit unions use Kasasa's decades-old, cutting-edge financial technology and marketing services to help people feel proud of their money. The business's main office is in Austin, Texas.

Challenge

You work as an AWS architect in Kasasa. Kasasa wants to build a model that can predict the legitimacy of a UFO sighting based on information supplied by the submitter.

Proposed Solution

We can use the ground truth dataset that has been collected by the organization previously. The organization has previously determined whether a sighting could be explained (as a hoax or other natural explanation), unexplained or probable.

We will build a model to determine whether newly reported UFO sightings are legitimate or not (explained, unexplained or probable). The organization requires the model to be at least 90% accurate. Hence, we will choose an algorithm, prepare and transform our data, and determine how accurate our model is.

Our result is that we need to have a model artifact stored in S3. We also want to present different model validation metrics, like accuracy, recall, precision, and the F1 score. We would also present the model error rate to show that our model is getting smarter and smarter, and the error rate decreases as we train our model.

We will build a model using the XGBoost algorithm as a multi-classification problem with researchOutcome, which is an attribute in our dataset as the target attribute. Our goal when using this algorithm is to minimize the training and the validation error.

The reason to choose XGBoost is that it is simple to implement and only needs two hyperparameters with 35 optional hyperparameters. We can use XGBoost for different problems, like ranking problems or regression problems, but here, we will use it as a classification problem, specifically, a multi-classification problem.

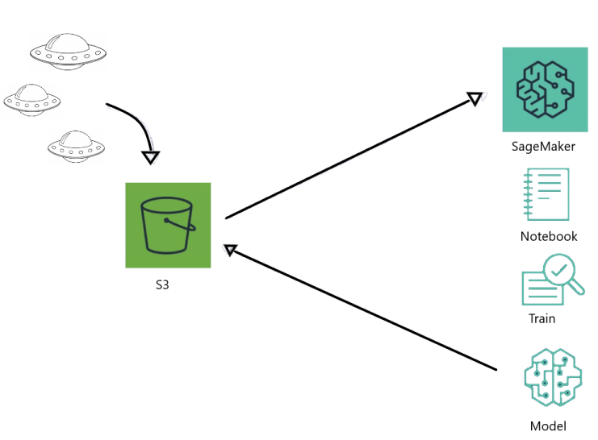
Since we are not exactly sure which attributes are most important, we need to use a classification algorithm over a clustering algorithm because if we choose a clustering algorithm, we need to know which attributes within our dataset will be the most determining factor.

We will employ a different algorithm to categorize the veracity of a UFO sighting. We will build a model using the Linear Learner algorithm as another multi-classification problem with the same attribute as our target attribute, researchOutcome. The goal is to maximize the training accuracy along with other metrics, like precision, F1 score, and recall. Linear Learner is a very flexible algorithm, and we can train different models and determine which attributes from our data are the most determining factors. Another feature of Linear Learner is it also has built-in hyperparameter tuning.

The same dataset that the organization previously gathered will be used. We are going to use this dataset and create a notebook instance. Within this notebook, we will analyze, visualize, prepare, transform, and get our data ready for our machine learning algorithm. In our case, XGBoost and Linear Learner both expect all of our values to be in numeric format, so we will need to encode those values or transform those values into numeric attributes.

Once we do the data transformation, we will divide our dataset into a training set, a validation set, and a testing set. Once we have our training and validation dataset ready, we will then run it through our algorithm, which will create a model and automatically upload that model onto S3. The logs from each of these algorithms from both the XGBoost and the Linear Learner will show us our model's error rate and accuracy.

Lab Diagram



*Figure 7-44: Lab Diagram*

Implementation Steps

1. Upload Data into S3.
2. Import all the necessary libraries.
3. Load the data from Amazon S3.
4. Clean, transform, analyze, and prepare the dataset.
5. Create and Train our model (XGBoost).
6. Create and train our model (Linear Learner).

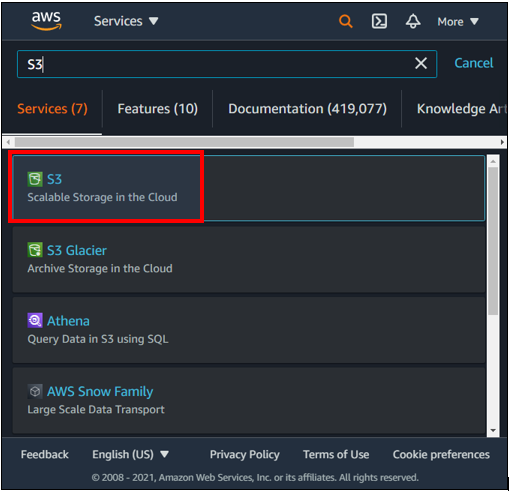
### Solution

**Note:** A lot of trial and error happens during the machine learning process. This is just part of the machine learning process. You will continuously create jobs that fail, and you will figure out the reasons why. You will realize that you do not have enough data, so you will have to get more. You will realize your data may not be in the right format. You might have accidentally left out a hyperparameter or used the wrong value for a hyperparameter. You will realize that your model is extremely inaccurate, so you must change something. You will realize that one algorithm works better than the other. But you need to remember that the machine learning process is a trial-and-error.

**Step 1: Upload Data into S3**

1. Log in to the **AWS Console**.

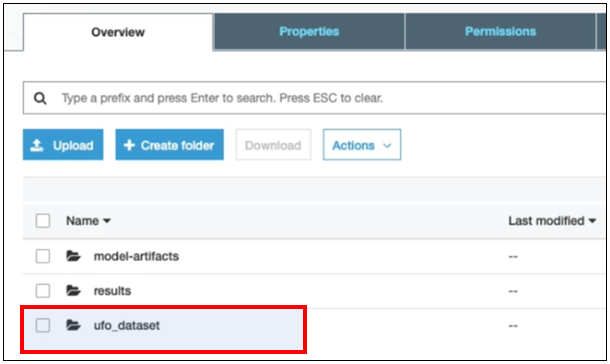
2. Go to **Services** and click on **S3**.



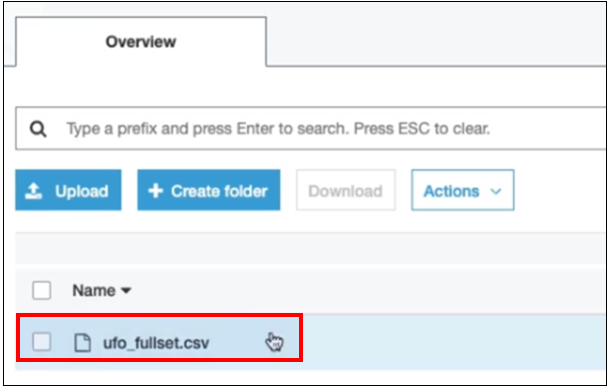
3. Click on the bucket **ml-labs-ips**.



4. Click on the subfolder **ufo\_dataset.**



5. Here, you can see the UFO dataset named **uf0\_fullset.csv.**

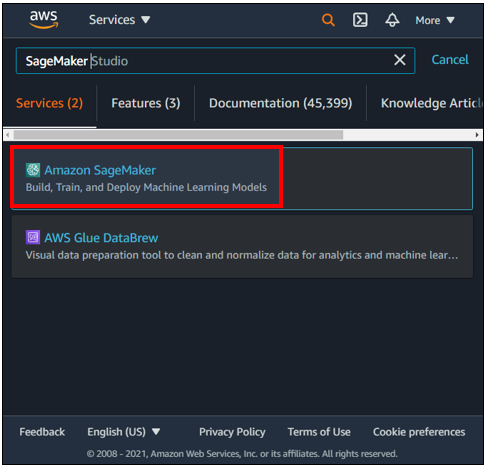


If you do not have this dataset, you can get it from the GitHub link below:

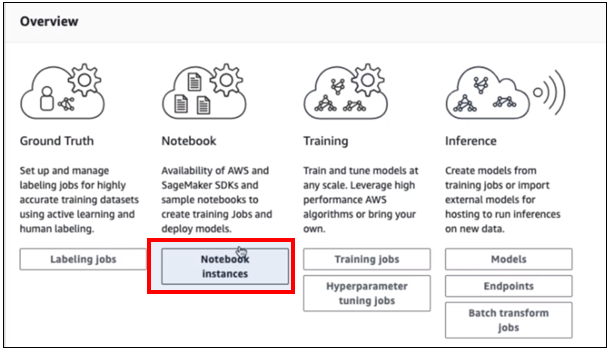
<https://github.com/12920/IPSpecialist01/blob/main/Course_AWS_Certified_Machine_Learning-master%20(1).zip>

Use Chapter7/ufo\_fullset.csv

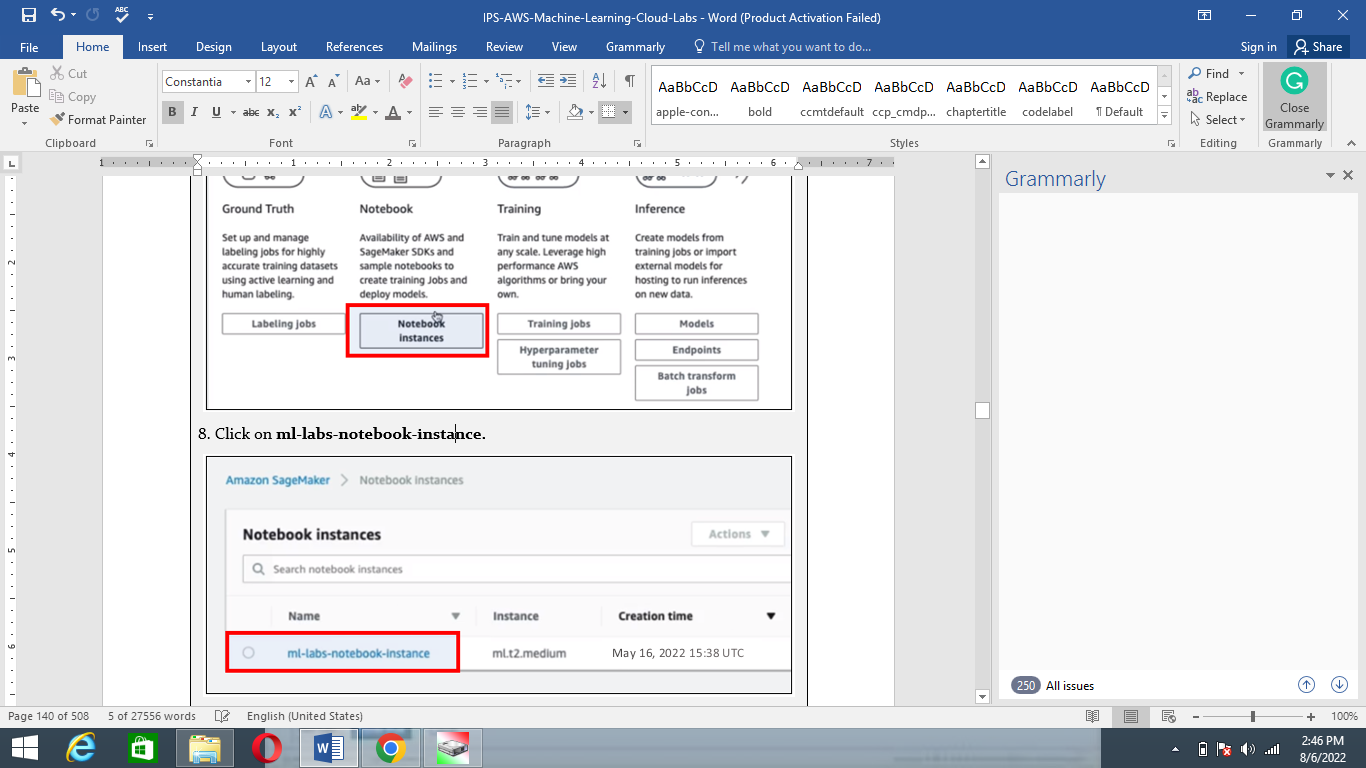
6. Navigate to **SageMaker.**



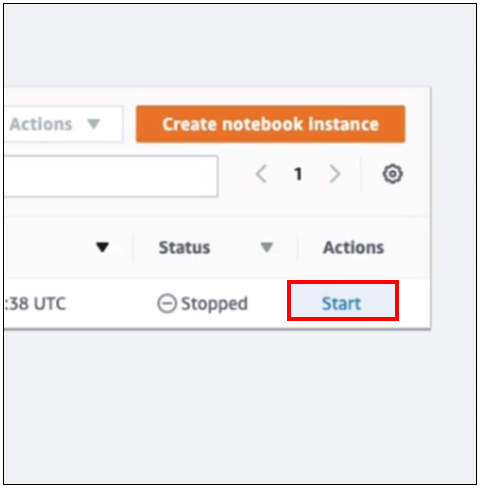
7. Click on **Notebook Instances**.



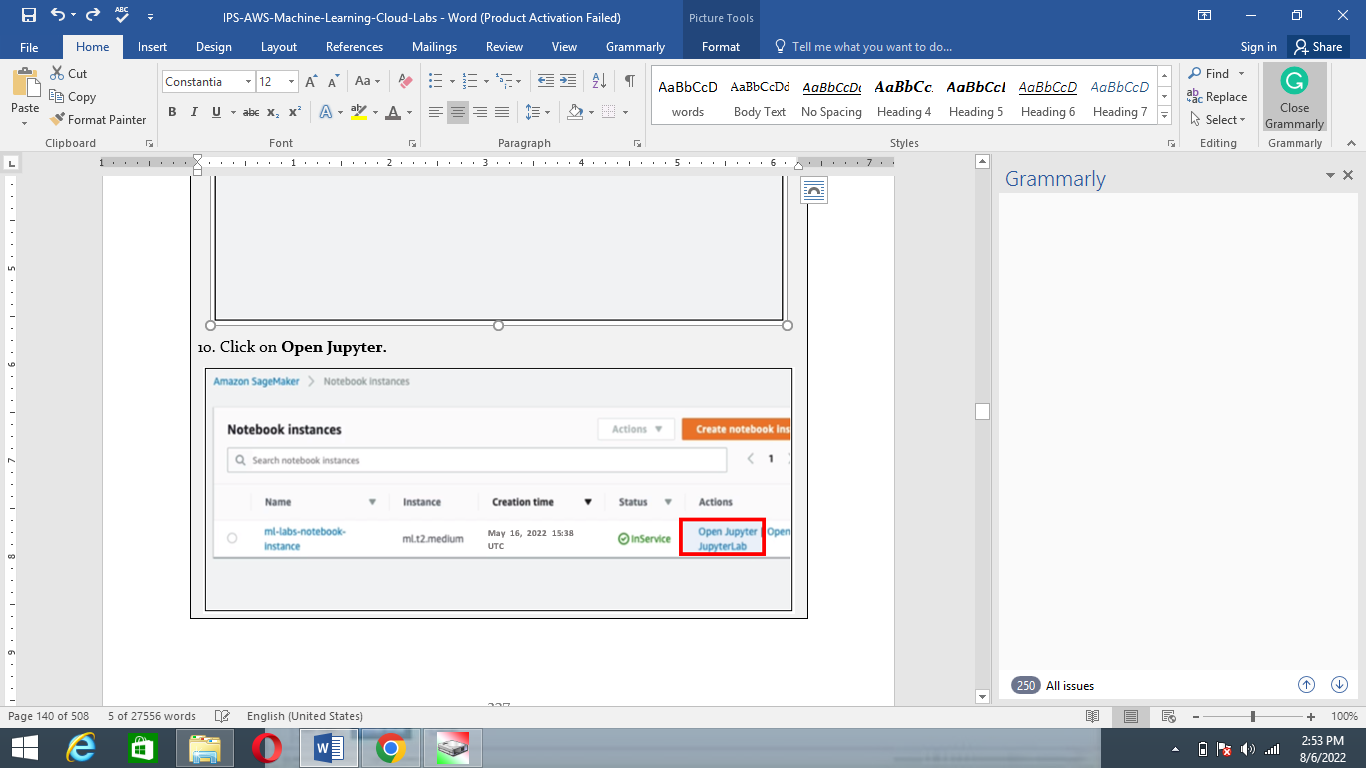
8. Click on **ml-labs-notebook-instance.**

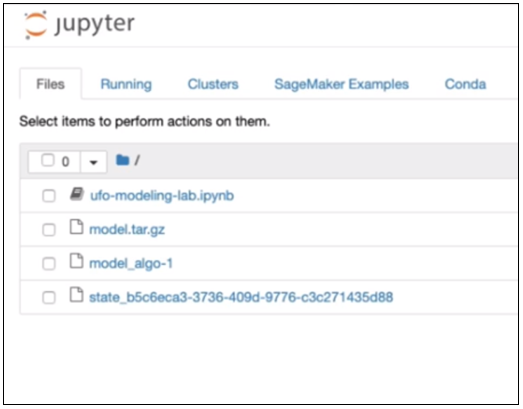


9. Click on **Start**.

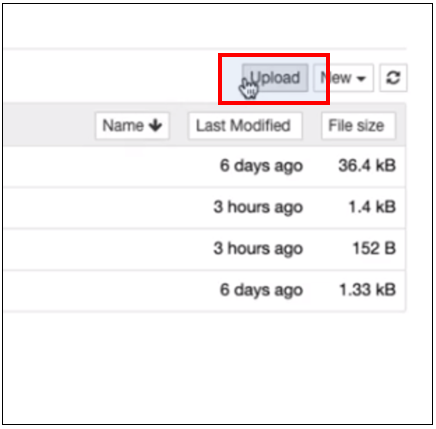


10. Click on **Open Jupyter.**

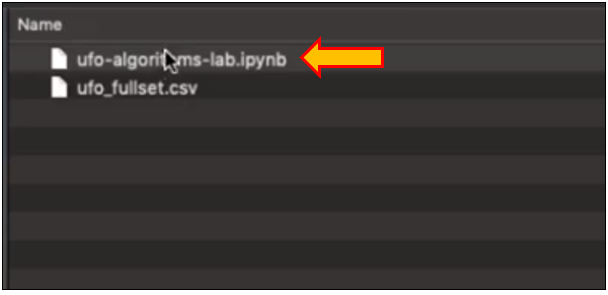




11. Click on **Upload.**



12. Select **ufo-algorithms-lab.ipynb.**

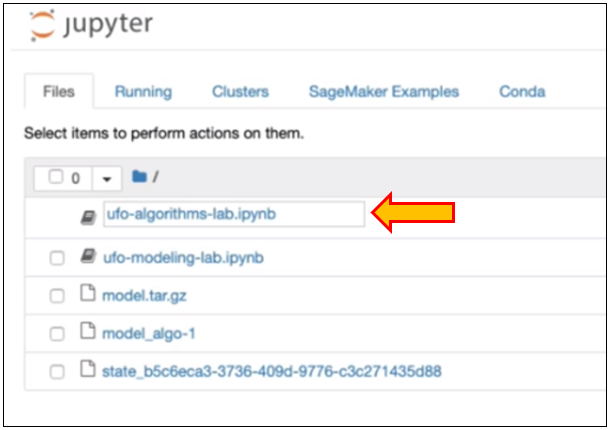


You can also download it from Github from the link given below:

<https://github.com/12920/IPSpecialist01/blob/main/Course_AWS_Certified_Machine_Learning-master%20(1).zip>

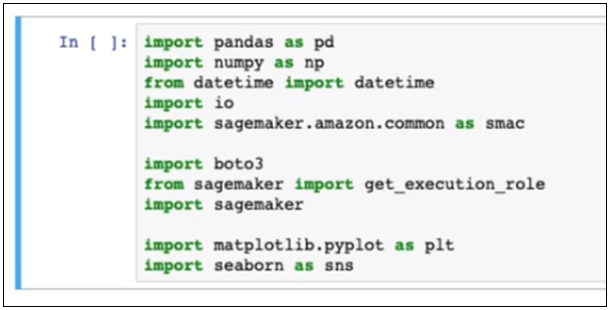
Use Chapter7/ufo-algorithm-lab.ipynb

13. Navigate into the **Jupyter notebook**.



**Step 2: Import all the necessary libraries**

1. Use the code given below to import all the necessary libraries.

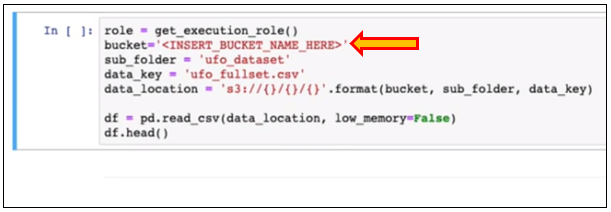


We are going to use NumPy and Pandas, along with some other libraries. We are also going to use the Python SageMaker libraries that AWS offers. We are also going to use Matplotlib and Seaborn. These Python libraries allow us to build visualizations and graphs to help us analyze our data.

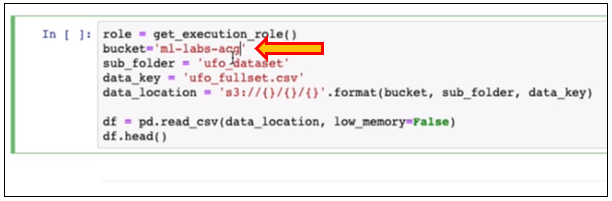
**Step 3: Load the data from Amazon S3**

We will load our data from S3 into the memory of our notebook instance.

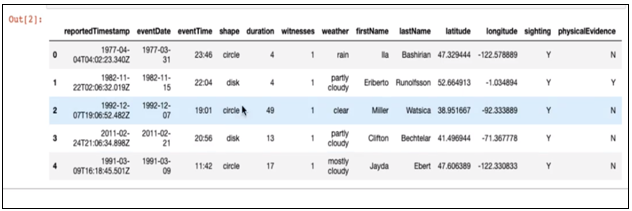
1. Insert your bucket name in the code.



2. Name your bucket, such as **ml-labs-ips**.

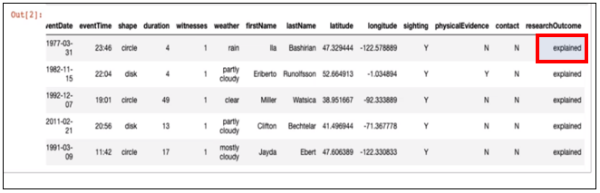


3. Once we run this, it will download the dataset from S3 and load it into the memory of our notebook instance. We will store that into a data frame and look at what that data frame or our dataset looks like.



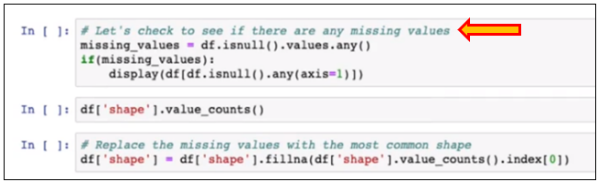
4. We are trying to solve the problem of determining what the researchOutcome is.

**Note:** All of this is explained, but there are explained, unexplained, and probable options available.

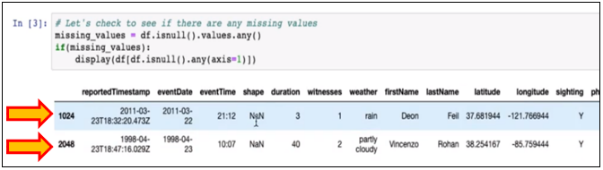


**Step 4: Clean, transform, analyze, and prepare the dataset**

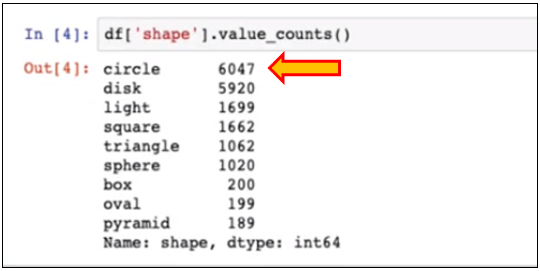
1. We need to clean, transform, analyze, and start preparing our dataset. We need to check to see any missing values within our dataset. If there are missing values, then it will display those, and we can look at what we might need to do to fix our missing values.



2. When we run the code, we get some missing values. If we look, the shape for a couple of these observations is null or not a number.



3. We will replace those missing values with the most common shape or the most common value. We could do this by getting the count for each shape; hence, we can see that a circle is the most common shape.



4. We will replace the missing values with the most common shape; in this case, it will replace those two missing values with the circle shape.



5. Now, we will start preparing our dataset by transforming some of the values into the correct data types.

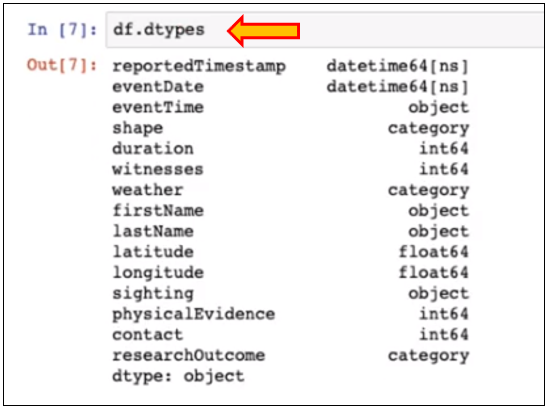
We are going to:

* Convert the reportedTimestamp and eventDate to datetime data types
* Convert the shape and weather to a category data type
* Map the physicalEvidence and contact from ‘Y,’ ‘N’ to 0, 1
* Convert the researchOutcome to category data.type (target attribute)

Once we successfully run these, we should be able to see the data types in the new format.



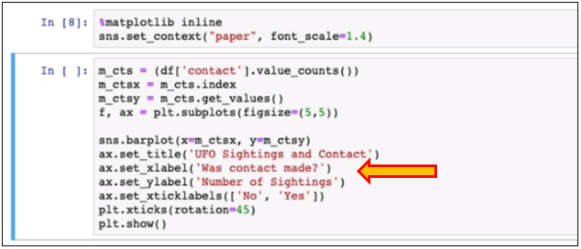
6. If we type **df.dtypes**, we should be able to see the different data types that our attributes are in now.



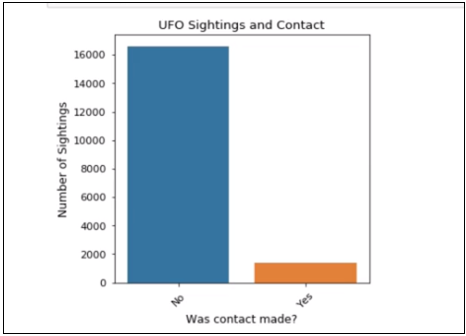
7. We can visualize some of our data to find any important information about that data. We might be able to find things like outliers, or we might be able to find the distribution of our data. We may be able to find any imbalances in our data as well.

8. We will configure Matplotlib and the Seaborn library, so we can start drawing out some visualizations.

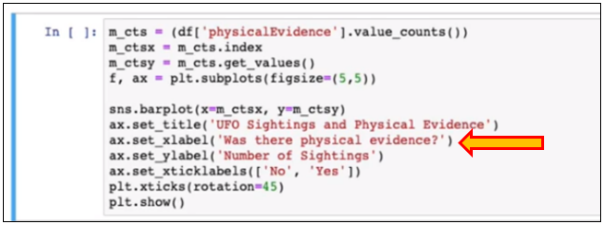
The first visualization is the UFO Sightings and Contact. Was contact made?



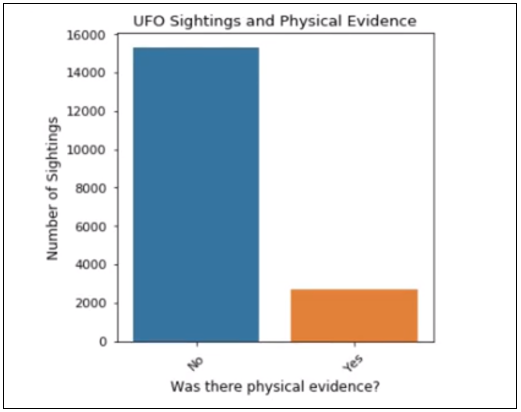
9. We can see that with most of our sightings, no contact was made.



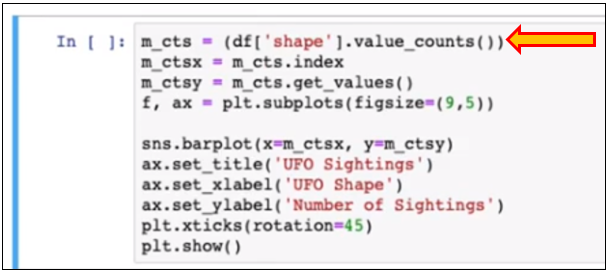
10. Now, to see how many sightings had some physical evidence associated with them.



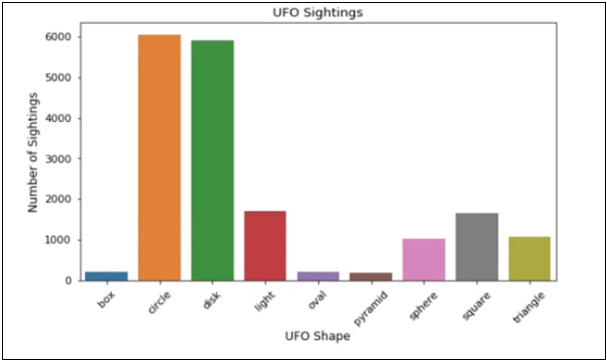
11. Once we create this visual, you can see that the majority, again, there was not much physical evidence left behind.



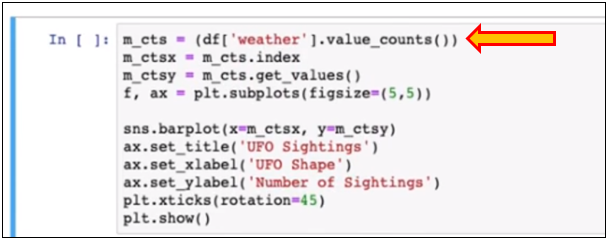
12. Next, look at the distribution of the different shapes for the UFO sightings.



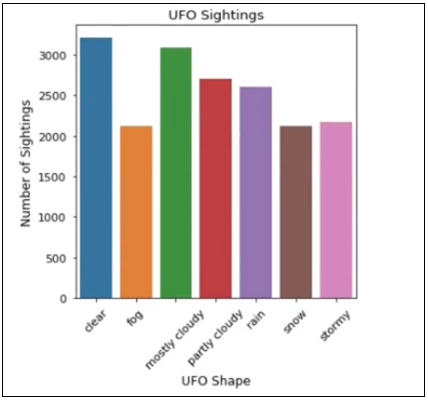
13. We can see that most of these sightings were either circles or disks, with a few more that were light and square. Then, they slowly decrease in value with the other shapes.



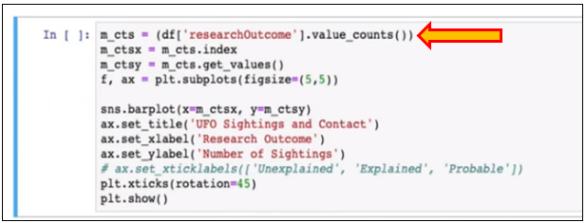
14. Now, to look at the distribution of weather, we can run the code below.



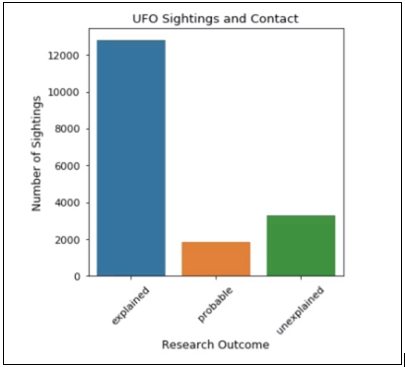
15. We can see here that most of the sightings were either clear or cloudy, and the other weather attributes follow. This distribution looks fairly even, so this may not correlate much with the outcome.



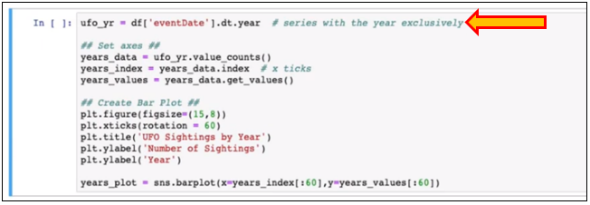
16. Now, we can run the code below to see what the researchOutcome distribution looks like.



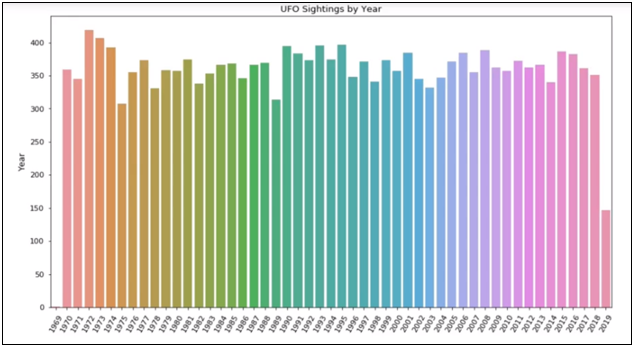
17. We can see the distribution in the researchOutcomes looks like the majority of our sightings were explained.



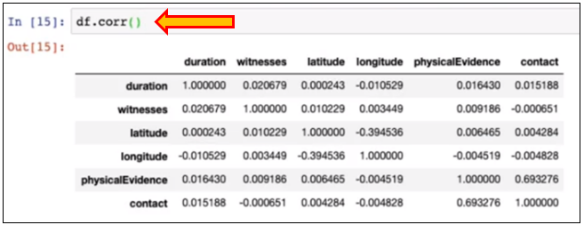
18. Looking at the distribution of sightings per year, we can run the code below.



19. You can see that the distribution has the sightings per year is fairly even. This may not be much of a deciding factor in what the researchOutcome is.



20. We are going to look at the correlation of these values.



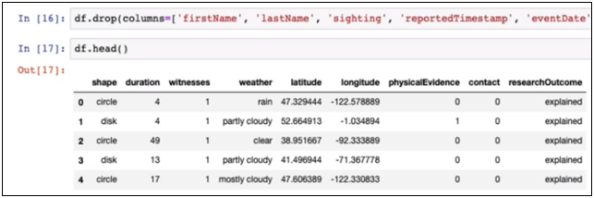
21. You can see some strong correlations between physical evidence and contact. So usually, there is physical evidence at least 69% of the time whenever there is contact. We need to include both of these within our training data. We do not see shape and weather because these are categorical variables, so we cannot find any correlation between them. We will include those within our training data as well.

22. We are going to drop the columns that are not important. If you look at the dataset, there are several columns that are not important to include within our training dataset. For example, we can drop the sighting because it is always Yes. We can also drop the first and last names because they are unimportant when determining the researchOutcome. We can also drop the reportedTimeStamp. And, we could create some buckets for the eventDate and the eventTime, for example, the season or what quarter it was in. But since the distribution of the dates is even, we can drop those.



To drop these values, we can simply call df.drop and pass in the name of the columns we want to get rid of.

23. If we take a look at our dataset by calling df.head, we can see that these are the only attributes left within our dataset.



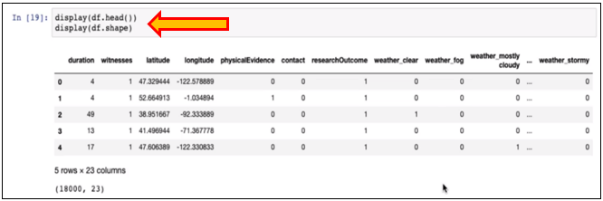
24. The XGBoost and the Linear Learner are both expecting numeric inputs. Hence, shape, weather, and researchOutcome need to be converted into numeric values.

Therefore, we will hot encode the weather and the shape attribute and map the researchOutcome, or the target attribute, into numeric values.

25. We will assign unexplained to zero, explained to one, and probable to two. Our machine learning algorithm is going to take all of the numeric input attributes from our dataset and try to find some correlation or some mapping that can be done to determine the researchOutcome as zero, one, or two.

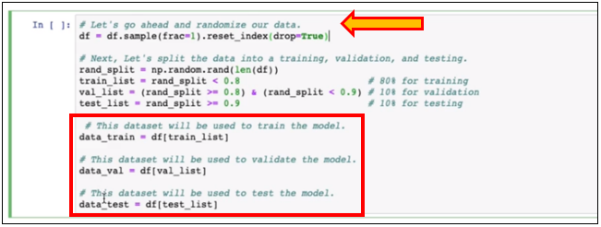


26. We can call df.head and df.shape to see what our dataset looks like now and how many attributes and columns we have.



We can see here that we have one hot encoded the weather and the shape attribute, and every other attribute is in numeric format.

27. Before we load our data into our machine learning algorithm, we will randomize and split the data into a couple of different datasets. We will randomize the data and split it into training, validation, and a testing dataset. We will split the data into 80% training, 10% validation, and 10% testing.

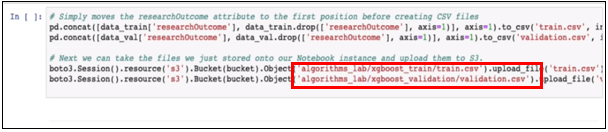


You can see here that the first thing we do is randomize the data and then split it into three different datasets; data\_train, data\_val, and data\_test.

28. Now, we need to rearrange our attributes, so the very first attribute is our target attribute or researchOutcome. The reason for doing this is that this is what XGBoost requires for the input data. It requires the first attribute in the dataset by the target attribute.

29. After rearranging those columns, we then want to store those as CSV files in our notebook instance. And then, we can load those files to S3 into their specific folders.

We will create a new folder called algorithms\_lab and a sub\_folder called xgboost\_train and xgboost\_validation. We will create the appropriate files for the training.csv and the validation.csv.



We are using CSV and not LibSVM because a data frame can quite easily be transformed into a CSV file by calling CSV.

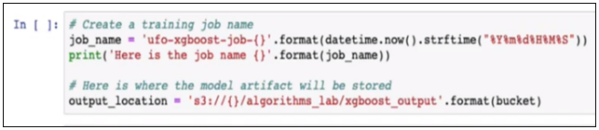
**Step 5: Creating and Training our model (XGBoost)**

1. The first thing is to get the container hosted in the Elastic Container Repository within AWS. We will specify the region name and the algorithm for which we would like to get the container.

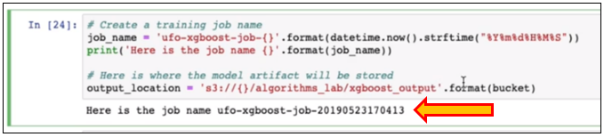


We will create a pointer that points to our S3 files for our training and our validation data.

2. First, we will create a specific job name with the job we are about to run.



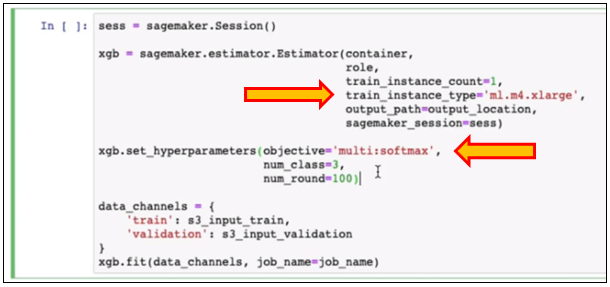
We can run the cell and see the job name.



3. Next, we can specify the training parameters we want to use with the XGBoost algorithm. We will pass in the algorithm container, IAM role, training instance type, and the count. In our case, we will use an ml.m4.xlarge as our training instance type and only use one of them. We will also pass in the S3 location for our output model artifact. We will also pass in the XGBoost hyperparameters. These are the only hyperparameters we will include within our XGBoost algorithm.

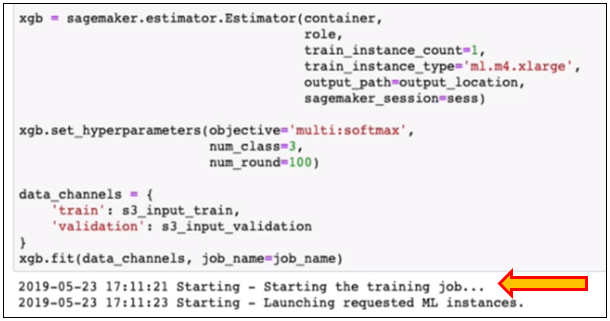
We can see that our objective is going to be multisoftmax. We could use softprob, but since we have a multi-classification problem, we need to set this to multisoftmax or multisoftprob. Both of these will give us our desired output.

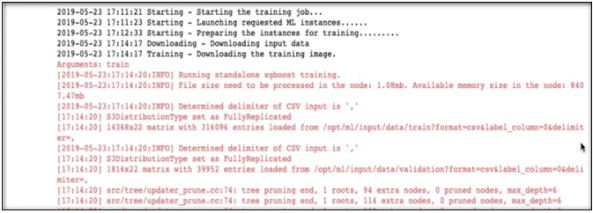
We will also include the number of classes, which is three, for explained, unexplained, and probable. We will also pass in the num\_rounds, the number of rounds we want our algorithm to pass over our data.



4. To start the job and our training process, we can use the .fit method and pass in the training and validation data.

5. You can see that the training job has started.

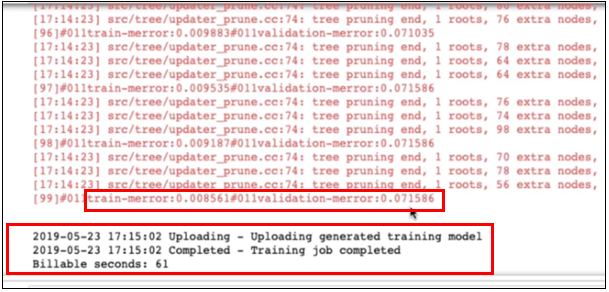




6. As we scroll down, we can see the model's output as it was training. Here, we have 0.03, which is about a 3% error; for validation, we can see that it has about a 6% error. That is around 94 to 97% accuracy.

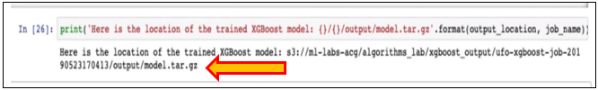


7. We will continue to scroll down and see what the final output was. You can see that the final output was about .8% training error, which is about 99% accuracy. And about a 7% error, which is around 93% accuracy.



This is exactly what we need. We have some data to show to our organization.

8. Now, we can print out the location of our trained model. If we look at this location S3, we can see our model artifact, and we can use this model artifact for inference to deploy it into a production environment later.

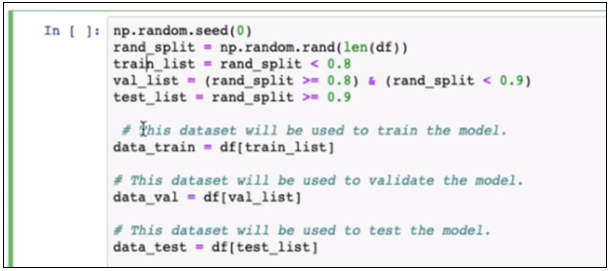


**Step 6: Create and train our model (Linear Learner)**

Now, we can start creating and training a Linear Learner model, where we can start to compare the accuracy between the two.

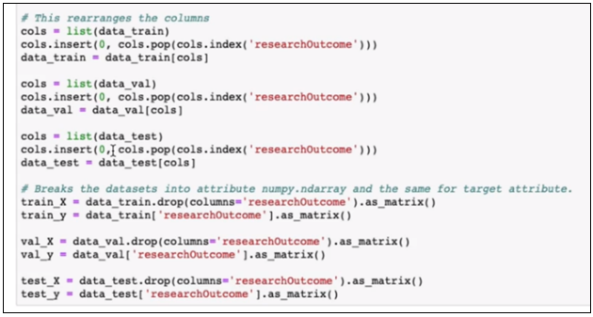
1. Since our data is already prepared and ready to go, we will randomize and split the data again for the Linear Learner algorithm.

2. We will split the data into three different datasets, a training, validation, and testing dataset. It will be 80% training, 10% validation, and 10% testing.

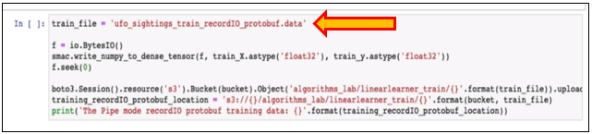


3. It will create a data frame for each dataset type, and it will rearrange the researchOutcome to where it is the very first column within the datasets for each of them.

We will also break the datasets into NumPy ndarrays. This is getting our data ready for the recordIO-protobuf. These are the necessary steps to get it ready for that format.

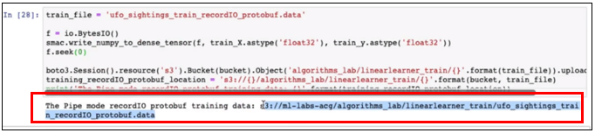


4. Now, we will create a recordIO file for training data and upload it to S3.



It is building a recordIO-protobuf for that dataset, which in our case is the training dataset, by taking the x-values, which are everything but the researchOutcome, and the y-values, which are only the researchOutcome.

5. When we run the code, we can see this is the location of where our recordIO-protobuf is stored.



6. We are going to perform the same steps for validation data.

When we run the code, we can see the S3 location of our validation data in recordIO-protobuf format.

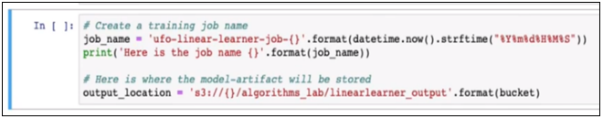


7. We will go through the same setup we did for the XGBoost algorithm, but we will use the Linear Learner algorithm.

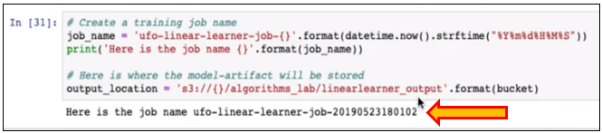
We are going to get everything we need from the ECR repository. This will create a container containing the Linear Learner docker image that we can use to train our model with.



8. We will create a custom job name and specify the output location for our model artifact.

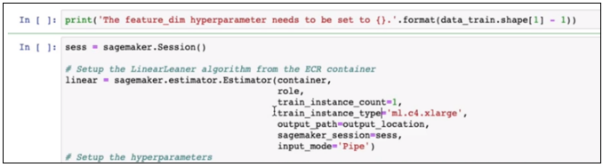


9. This will be the name of the training job we are about to create.



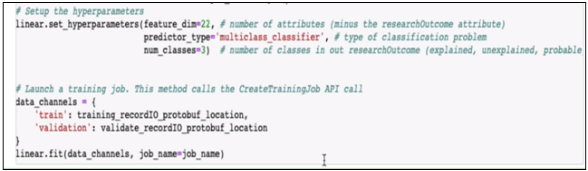
10. We will call the SageMaker library to create an estimator, which is a way to create a machine learning model by inputting the container and some other parameters about the algorithm we want to use. We can include the container, the IAM role, the training instance count, and the training instance type, and AWS recommends using a CPU instance. Hence, we will use an ml.c4.xlarge to train our model. We can specify the output location where the model artifact will live. We will pass in the SageMaker session and include the input\_mode type of Pipe.

We can take advantage of using recordIO-protobuf as the input type and stream the data directly from S3. This will optimize the streaming process of getting the data from S3 onto our notebook instance.

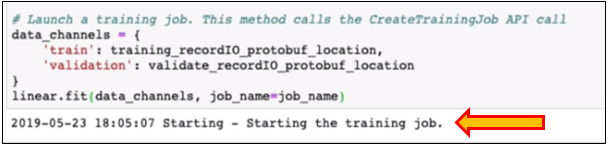


11. We can specify all the hyperparameters associated with the Linear Learner algorithm. We can scroll down and take a look at which ones are required. The ones that are required are the featured dimensions. In our case, it will be the number of attributes minus the researchOutcome, so we have 22 total features or 22 total attributes.

12. We will specify the predictor type, which is a multiclass\_classifier, because this is a multi-classification problem. We will also specify the number of classes, similar to the XGBoost, we have explained unexplained and probable. These are the only hyperparameters that are required for the Linear Learner algorithm.



13. We will leave the others as default and see what our results look like. We will call the .fit method, pass in our training data, which in this case is recordIO-protobuf, and pass in the job name that we created earlier.



14. The first thing you can do is the default configurations for the hyperparameters. So, if we look at this JSON file or this JSON data, we should be able to see each hyperparameter that the Linear Learner algorithm uses to create the model.



15. In our case, we can see the hyperparameters that we input. We can see num\_classes.



16. We can also see the featured dimensions.

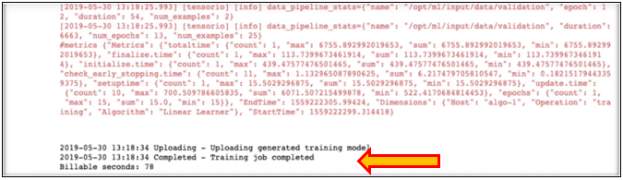


17. If we scroll down, we can see the output from our Linear Learner algorithm. The metrics that mean a lot to us are the validation\_score metric and the quality\_metric.

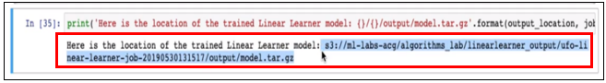
18. We can start to see the multiclass\_accuracy is around 0.94%, which is around 94% accuracy. And we can also see recall, precision, and the F1 score.



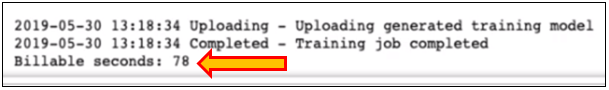
19. If we scroll down all the way to the bottom, we can see that the training job is completed.



20. If we run the cell, we can see where the Linear Learner model artifact is stored.

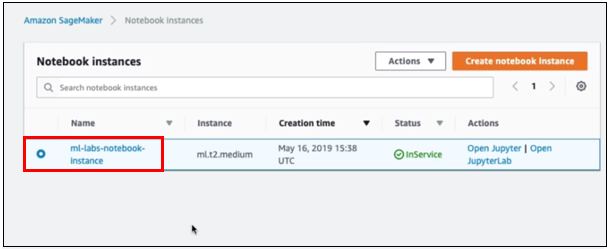


21. It is important to know how long the training job took, and you can see the billable seconds were 78 seconds. So, it charged us 78 seconds when it spun up the compute-optimized instance and trained our model.



Hence, we have created an XGBoost model and a Linear Learner model. Our XGBoost model has around 93% accuracy, and our Linear Learner has around 94% accuracy. Therefore, we have two models that are above 90% accurate. We can use these metrics to show the organization that we can use them in production or make inference calls.

22. We can now clean up our environment. Navigate back to SageMaker and select your notebook instance.



23. Use the Actions dropdown and select Stop.

