

University of Salford, MSc Data Science

Module: Big Data Tools and Techniques

Date: Trimester 2, 2024-2025

Session: Workshop Week 7

Topic: Machine Learning in Databricks using MLlib and MLflow

Tools: Databricks Community Edition

Instructors: Nathan Topping, Dr Taha Mansouri, and Dr Kaveh kiani.

Objectives:

After completing this workshop, you will be able to:

- Use Spark SQL to carry out exploratory data analysis
- Use the MLlib library to pre-process your dataset ready to use for machine learning
- Use the MLlib library to train a classification model
- Evaluate the accuracy of a classification model
- Track runs and experiments using MLflow and the Databricks Experiment UI

Table of Contents

A Brief Introduction to Supervised Machine Learning	3
MLlib	4
MLflow.....	4
Part 1: Fire up the Databricks workspace	5
Part 2: Creating a new notebook.....	6
Part 3: Exploratory Data Analysis.....	7
Part 4: Data Pre-processing	10
Part 5: Training the Model.....	12
Part 6: Evaluating the Model	14
Part 7: Using MLflow & Viewing Run Details.....	15
Part 8: Using ParamGridBulder & TrainValidationSplit for Grid Search	18

A Brief Introduction to Supervised Machine Learning

When we carry out supervised machine learning, we want to build a model which we can use to **predict** something which is of interest to us – this could be anything from the price of a house, whether a patient has a disease or whether a potential customer is creditworthy. It is called **supervised learning** because we start with a **labelled dataset** which includes the attribute we are interested in as well as other features which we think will be useful for making predictions. For example, if we want a model which can predict whether or not a patient has diabetes, these features may include things such as their blood pressure, blood sugar levels etc. We use this dataset to train the model. Once we have trained the model, if we are satisfied with its performance, we can then use it to make predictions on unlabelled data in the future.

Sometimes when we carry out supervised learning, we want to predict a continuous variable – this is something like the price of a house, which can take any value within a range. This is known as **regression**. Other times, we want to predict a categorical variable, a variable which can only take specific values. For example, this might be whether or not a patient has a disease which is either ‘Yes’ or ‘No’. This is known as **classification**.

In this workshop, we will be training a classification model to detect whether or not a room is occupied from temperature, humidity, CO2, and humidity ratio measurements. These readings could be taken from sensors within a room, and a model which predicts whether or not a room is occupied from these readings would be very useful in a number of use cases – for example, for a smart energy system to automatically control the lights, or in a security system to detect intrusion.

Machine learning is a key part of a Data Scientist’s role, and we cover machine learning in a lot more depth in the Machine Learning & Data Mining (MLDM) module of this course (if you haven’t already taken this module!) In MLDM, we will also discuss the different machine learning algorithms and how they work – however, for now we will simply focus on how we can use one in practice. Typically, when we are carrying out supervised machine learning, we are likely to follow a process like the below:

1. Exploratory Data Analysis
2. Data Pre-processing
3. Model training
4. Model evaluation
5. Model deployment

This is a very simplified overview, and in practise, we may need to carry out more than one iteration of the model training to get a model which is performing well enough for our purpose. To evaluate a model, we want to know how it will perform on data it’s never seen before. So, we can see how it performs on previously unseen data, we split our initial dataset into two parts; a **training dataset** which we use for training the model, and a **test dataset** which we use to evaluate how well it works when making predictions on new data.

There are two new libraries we're going to introduce in today's workshop

MLlib

MLlib is Spark's machine learning library. It provides implementations of many common machine learning algorithms, including classification, regression, clustering and collaborative filtering algorithms. It also provides tools for data pre-processing, such as feature extraction, transformation and selection.

There are two packages within MLlib. The first is `pyspark.mllib` – this leverages the RDD APIs provided by Spark. However, this is now in 'maintenance mode' and the primary API is `pyspark.ml` which is a DataFrame-based API. We will be using this one in today's workshop.

In MLDM, we introduce Scikit Learn and Tensorflow, which are libraries which are more widely used for machine learning. These libraries are generally the most appropriate choices unless you are working with big data, and it becomes difficult or inconvenient to use these libraries. When working with big data, MLlib makes ***distributed*** machine learning very simple. The dataset we use today, for example, is very small (around one thousand rows) so in practice we would be likely to use Scikit Learn for this. However, the aim of today is to introduce you to MLlib, because if you are working with big data, MLlib becomes a useful tool to complement the other libraries we cover on this course.

You can view the documentation for MLlib here:

<https://spark.apache.org/docs/latest/ml-guide.html>

MLflow

MLflow is an open-source platform to manage the ML lifecycle, including experimentation, reproducibility, deployment, and a central model registry. MLflow currently offers four components:

- MLflow Tracking: Record and query experiments: code, data, config, and results
- MLflow Projects: Package data science code in a format to reproduce runs on any platform
- MLflow Models: Deploy machine learning models in diverse serving environments
- Model Registry: Store, annotate, discover, and manage models in a central repository

MLflow is included in Databricks Community Edition, meaning that you can utilize its Tracking and Model APIs easily within a notebook. It can be used with many machine learning libraries, including Scikit Learn and Tensorflow, as well as MLlib.

You can read more online here:

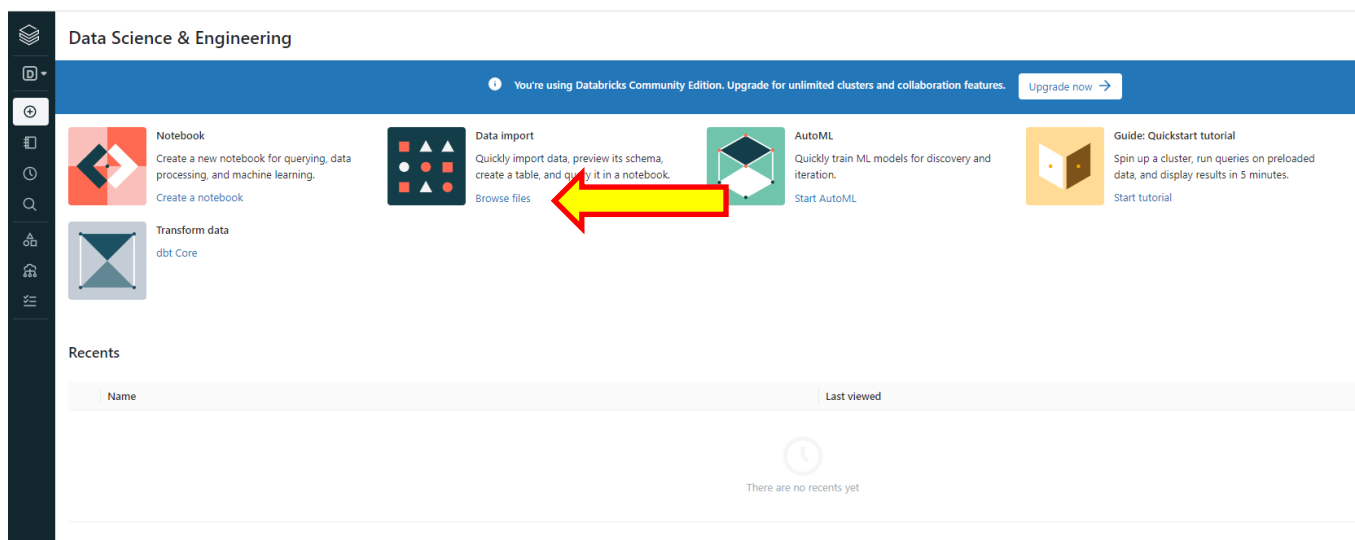
<https://mlflow.org/>

Part 1: Fire up the Databricks workspace

1. Log in to your Databricks Community Edition account. Here is the link:

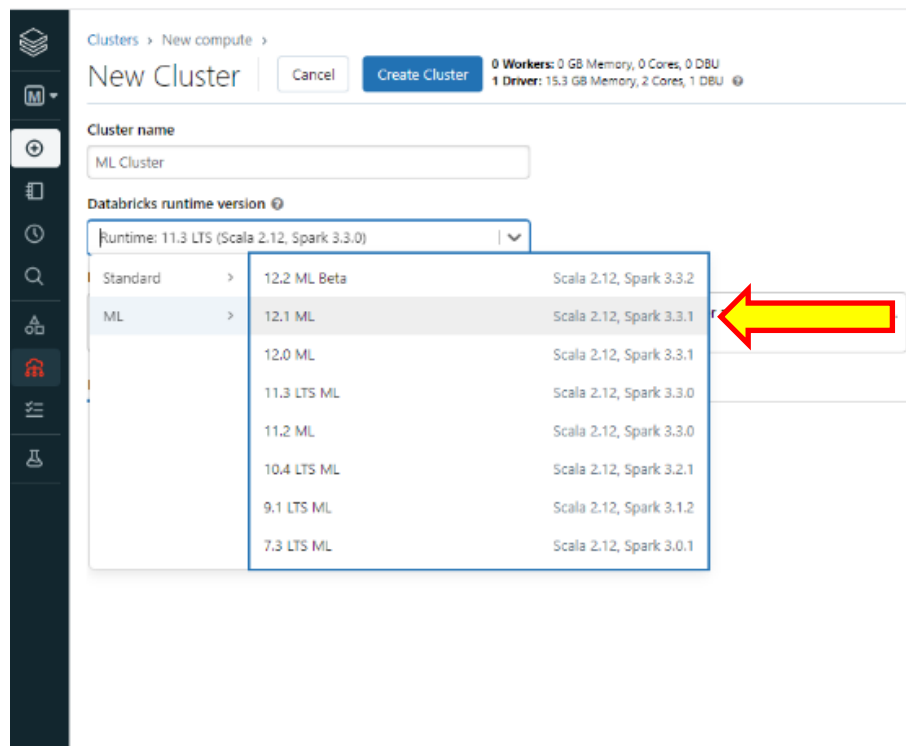
<https://community.cloud.databricks.com/login.html?nextUrl=%2F>

2. You will need to download the csv file **Occupancy_Detection_Data.csv** from Blackboard for use in this workshop. Then on your Databricks home screen select 'Browse files' under the Data Import option on your screen, and then navigate to the directory location where you have saved the csv file. Upload this to Databricks.



3. Click on "Create Compute" and type in a new name for the cluster, any name that you like.

4. From the dropdown, you **must** select an ML runtime and not a standard runtime. This will ensure that it has all the necessary libraries installed for this session. Select a **12.1 ML** runtime.



Note that it will take a few minutes to create the cluster. After some time, the green circle next to the cluster name will gain a green tick meaning the cluster has successfully started up.

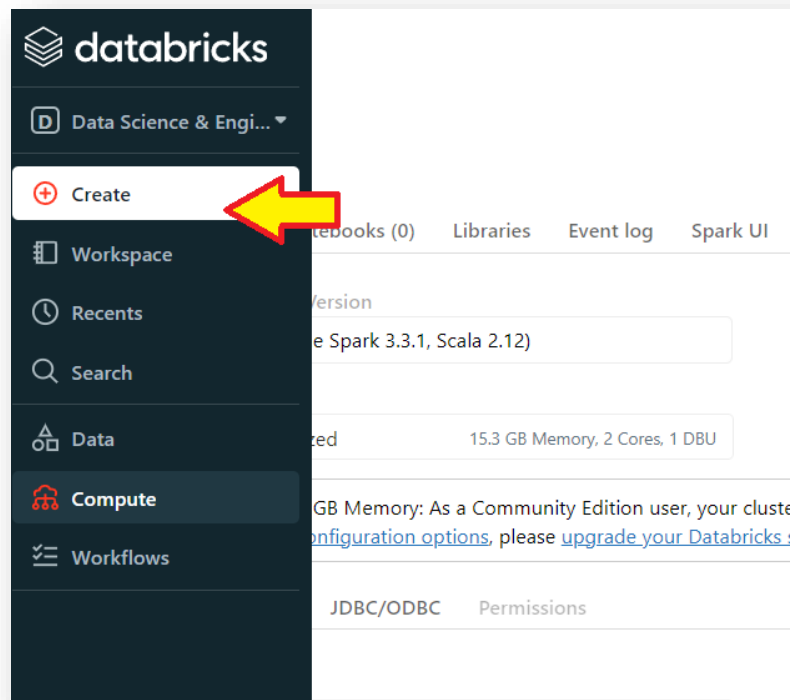
Part 2: Creating a new notebook

Databricks programs are written in Notebooks. A notebook is a collection of runnable cells which contain your commands. Look at:

<https://docs.databricks.com/notebooks/notebooks-use.html>

for an overview of how to use a Databricks notebook.

1. Click on the button “Create” and select “Notebook”.



2. Name your notebook, e.g. “Week 7 Machine Learning”, leave the language as the default Python and the cluster should be the cluster you have just created.

Part 3: Exploratory Data Analysis

Before we carry out any machine learning on a dataset, it is good practice to explore the dataset and get a better understanding of the data we are working with. This typically includes plotting graphs to help us visualise the data and/or computing summary statistics, such as the mean. In supervised machine learning, we are particularly interested in whether there are any differences in the features for examples belonging to the different labels (or **classes**) we are trying to predict.

Let's start by reading the data into a Spark DataFrame and then exploring it.

1. Before we start, we need to import the MLflow platform and enable autologging. We will also adjust the logging level to suppress non-error messages from MLflow for cleaner output.

```
11:07 AM (1s)
# import mlflow and autolog machine learning runs

import mlflow
import logging


logging.getLogger("mlflow").setLevel(logging.ERROR)
mlflow.pyspark.ml.autolog()
```

- Now we can read the dataset which we uploaded into a Spark DataFrame using `spark.read.csv()`. Because the first row of the file contains the header, we set `header` to `true`. We also want to infer the schema from the file so have set the option `inferSchema` to `true`.

Cmd 2

```
1 # read data into spark DataFrame
2
3 occupancyDF = spark.read.csv("/FileStore/tables/Occupancy_Detection_Data.csv",
4                               header = "true",
5                               inferSchema="true")
```

We can look at the schema by expanding the output below the cell. We can see that the Timestamp column has not been recognised as a timestamp data type. If we were using this column, we would need to convert this to the right data type; however, we will be dropping this column before training our model, so we don't need to worry about dealing with this.

▼  occupancyDF: pyspark.sql.dataframe.DataFrame

- Timestamp: string
- Temperature: double
- Humidity: double
- CO2: double
- HumidityRatio: double
- Occupancy: integer

- We can use the `display()` method to view the data in the output. The Occupancy column is either 1 if the room is occupied or 0 if the room is not occupied.

Cmd 3

```
1 # Use display to view occupancy DataFrame and create Databricks data visualisations
2
3 occupancyDF.display()
```

► (1) Spark Jobs

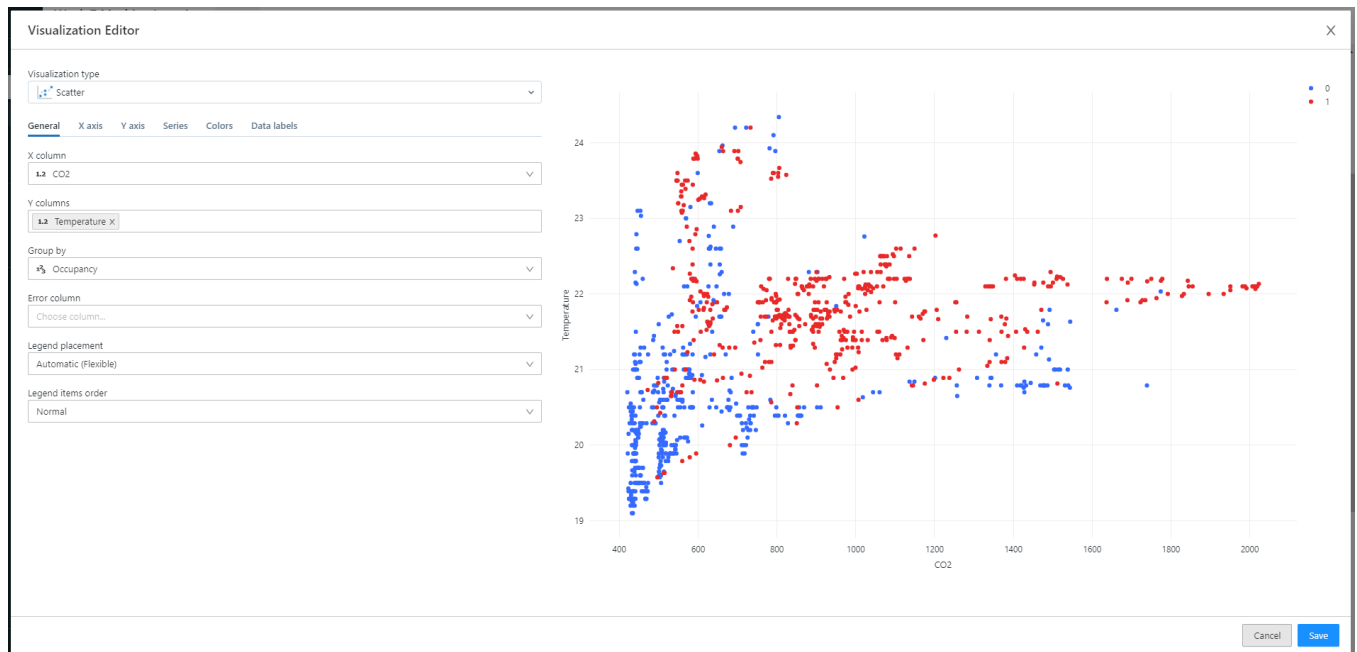
Table ▼ +

	Timestamp ▲	Temperature ▲	Humidity ▲	CO2 ▲	HumidityRatio ▲	Occupancy ▲
1	11/02/2015 14:51	21.7675	31.1225	1009.5	0.005021569	1
2	11/02/2015 14:57	21.79	31.46333333	1027.333333	0.005084053	1
3	11/02/2015 15:04	21.89	31.6	1060.5	0.005137857	1
4	11/02/2015 15:19	21.89	31.73	1100	0.005159169	1
5	11/02/2015 15:36	21.89	30.65	896.5	0.004982159	1
6	11/02/2015 15:59	21.89	30.42666667	792.6666667	0.004945567	1
7	11/02/2015 16:01	21.89	30.39	797.75	0.00493956	1

⬇ ▼ 1,000 rows | Truncated data ▼ | 0.92 seconds runtime

Command took 0.92 seconds -- by n.j.topping@salford.ac.uk at 2/26/2023, 10:08:11 PM on ML Cluster V2

We can also make use of the Databricks in-built visualisation capabilities to get a better understanding of the variables. Click the plus sign next to 'Table' in the output, and then 'Visualization' to open the visualization editor. If you select CO2 for the X column, Temperature for the Y column and Occupancy under Group By you can see the below scatterplot. We can see from the below that occupied rooms tend to have higher temperatures and CO2 levels than the unoccupied rooms.



Challenge 1

Use the in-built Databricks visualisations to explore each variable's distribution using a histogram.

4. If we create a temporary view, we can also use SQL to query the data. Run the below two code cells to create a temporary view and then return the mean value for each variable, grouped by the occupancy status. As we thought from the scatterplot, the mean value for temperature and CO2 levels are both higher for occupied rooms than unoccupied rooms.

Cmd 4

```
1 occupancyDF.createOrReplaceTempView("occupancyView")
```

Command took 0.27 seconds -- by n.j.topping@salford.ac.uk at 2/26/2023, 10:17:01 PM on ML Cluster V2

Cmd 5

```
1 %sql
2
3 SELECT Occupancy, AVG(Temperature) AS Average_Temperature,
4        AVG(Humidity) AS Average_Humidity,
5        AVG(CO2) AS Average_CO2,
6        AVG(HumidityRatio) AS Average_Humidity_Ratio
7 FROM occupancyView
8 GROUP BY Occupancy
```

You should see this output:

Table ▾ +

	Occupancy ▲	Average_Temperature ▲	Average_Humidity ▲	Average_CO2 ▲	Average_Humidity_Ratio ▲
1	1	21.871518931764747	27.96006930359028	958.1930020277889	0.004539106375253549
2	0	20.58318598559928	28.040931237681857	622.8635232447936	0.0041981035245579525

Challenge 2

Use Spark SQL queries to determine the minimum and maximum values for each variable for when the room is occupied and for when the room is unoccupied. Also confirm the number of rows in the dataset which are labelled as occupied and the number labelled as unoccupied.

Part 4: Data Pre-processing

Once we have explored the data, we need to process it so that we can use it to train a classification model. Firstly, we are going to drop the timestamp column as we don't want to use this during training.


5. Run the below code to drop the Timestamp column and then print the schema so we can verify that it has been dropped.

Cmd 6

```

1  # drop timestamp column
2
3  occupancyDF = occupancyDF.drop(occupancyDF.Timestamp)
4
5  occupancyDF.printSchema()

```

▶  occupancyDF: pyspark.sql.dataframe.DataFrame = [Temperature: double, Humidity: double ... 3 more fields]

```

root
|-- Temperature: double (nullable = true)
|-- Humidity: double (nullable = true)
|-- CO2: double (nullable = true)
|-- HumidityRatio: double (nullable = true)
|-- Occupancy: integer (nullable = true)

```

Command took 0.11 seconds -- by n.j.topping@salford.ac.uk at 2/26/2023, 10:27:22 PM on ML Cluster V2

6. Our next step is to pre-process the data so that it is in the correct format to train an MLlib model. When we are using MLlib, it expects the features to be provided in a vector. To provide it in this format we can use the RFormula transformer. It is named this because it uses a syntax which is common in the R programming language (which we cover in Applied Statistics and Data Visualisation). To show that we want to predict Occupancy from Temperature, Humidity, CO2 and HumidityRatio we use the below:

$$\text{Occupancy} \sim \text{Temperature} + \text{Humidity} + \text{CO2} + \text{HumidityRatio}$$

However, because we are using all the columns in the DataFrame, apart from Occupancy, for the prediction we can simplify this by using a full stop instead of writing this out in full, like this:

$$\text{Occupancy} \sim .$$

Run the below code:

Cmd 7

```

>
1  # preprocess data into correct format
2
3  from pyspark.ml.feature import RFormula
4
5  preprocess = RFormula(formula="Occupancy ~ .")
6
7  occupancyDF = preprocess.fit(occupancyDF).transform(occupancyDF)
8
9  occupancyDF.show(5)

```

You should get an output similar to the below. We can see that this transformer has added two new columns to the Data Frame. The first is 'features' and contains a vector which includes the values from the first four columns. The second is 'label' and contains the values from the Occupancy column.

You can also see from the output above the Data Frame that running this code has logged a **run** to an **experiment** in MLflow. We will be looking into what this means later.

```

▶ (1) Spark Jobs
▼ (1) MLflow run
  Logged 1 run to an experiment in MLflow. Learn more
▶ occupancyDF: pyspark.sql.dataframe.DataFrame = [Temperature: double, Humidity: double ... 5 more fields]
2023/02/26 22:37:27 INFO mlflow.utils.autologging_utils: Created MLflow autologging run with ID '...'
for the current pyspark.ml workflow
+-----+-----+-----+-----+-----+-----+-----+-----+
|Temperature|Humidity|CO2|HumidityRatio|Occupancy|features|label|
+-----+-----+-----+-----+-----+-----+-----+
|21.7675|31.1225|1009.5|0.005021569|1|[21.7675,31.1225,...]|1.0|
|21.79|31.46333333|1027.333333|0.005084053|1|[21.79,31.46333333...]|1.0|
|21.89|31.6|1060.5|0.005137857|1|[21.89,31.6,1060...]|1.0|
|21.89|31.73|1100.0|0.005159169|1|[21.89,31.73,1100...]|1.0|
|21.89|30.65|896.5|0.004982159|1|[21.89,30.65,896...]|1.0|
+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows

Command took 3.25 seconds -- by n.j.topping@salford.ac.uk at 2/26/2023, 10:37:27 PM on ML Cluster V2

```

- Our next step is to split the data into a training dataset and a test dataset. As we mentioned at the start of the workshop, when carrying out supervised learning we will hold back some of the dataset when we are training our model. This is so we have some labelled data that our model hasn't yet seen – we can then get it to make predictions for this dataset so we can see how well it performs on new data. We are randomly splitting the data with 70% allocated to the training Data Frame and 30% to be held back as test data.

```

Cmd 8
1 # split data into training and test datasets
2
3 (trainingDF, testDF) = occupancyDF.randomSplit([0.7, 0.3], seed=100)

▶ trainingDF: pyspark.sql.dataframe.DataFrame = [Temperature: double, Humidity: double ... 5 more fields]
▶ testDF: pyspark.sql.dataframe.DataFrame = [Temperature: double, Humidity: double ... 5 more fields]

Command took 0.13 seconds -- by n.j.topping@salford.ac.uk at 2/26/2023, 10:53:07 PM on ML Cluster V2

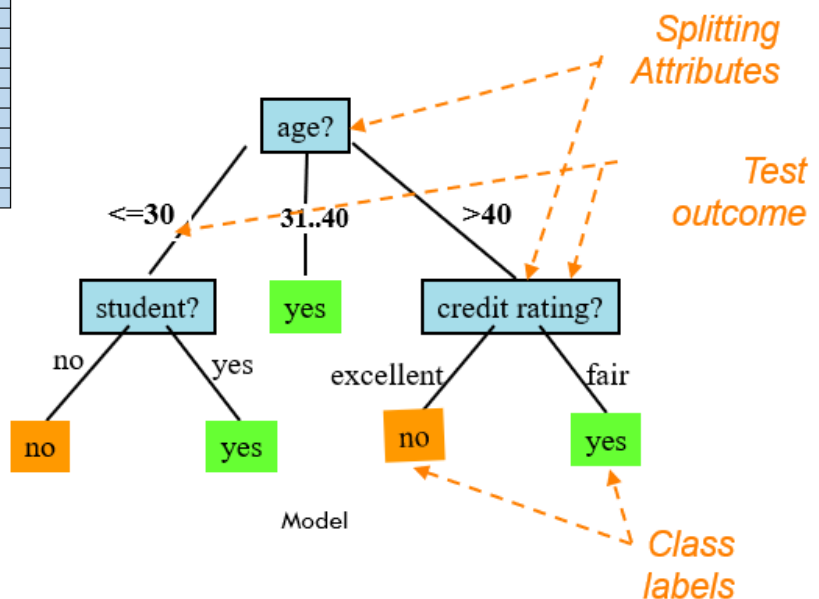
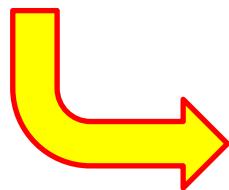
```

Part 5: Training the Model

- For this workshop, we are going to use a `DecisionTreeClassifier` estimator from within `MLlib` as our classification model. As we mentioned, in `MLDM` we will be discussing how this algorithm works in more detail.

However, the basic idea is provided below:

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no



The algorithm uses the training data to construct a *decision tree* which splits the data based on the features we are using to make the prediction. Each **branch node** splits the data based on one of the features and when you reach a **leaf node** the class label is the predicted class. So, for the decision tree above if we were given a new, unlabelled example:

age	Income	Student	<u>credit_rating</u>	<u>buys_computer</u>
<=30	No	Yes	fair	?

We can use the tree to make predictions as follows. At the first branch node, the data is split by age and in our example, the individual's age is less than or equal to 30. This means we take the corresponding branch in the tree, which is the one to the left. The next splitting attribute is whether they are a student. In this case they are, so we take the corresponding branch which is the one to the right. We are now at a leaf node which has the class label **Yes**. So, our prediction is that this individual will buy the computer. (In reality, we are not generally using the tree to make predictions manually!)

Once we have a decision tree model, it is very easy for us to generate predictions. However, first we have to train the model – by this we mean we have to construct an optimal decision tree. We will come back to the algorithm that does this in MLDM.

To implement this using MLlib, we first instantiate an instance of the estimator and specify the column in the Data Frame which is the label and the column which is the features. We can then use the `fit()` method to fit the model to our data – i.e., to train it.

Cmd 9

```
1  from pyspark.ml.classification import DecisionTreeClassifier
2
3  dt = DecisionTreeClassifier(labelCol="label", featuresCol="features")
4
5  # train the model
6
7  model = dt.fit(trainingDF)
```

Part 6: Evaluating the Model

9. To evaluate the model, we need to use our trained model to make predictions on the test data, which we did not use when training the model. We generate predictions by using the `transform()` method on the test data.

Cmd 10

```
1  # make predictions on the test dataset
2
3  predictions = model.transform(testDF)
4
5  predictions.show()
```

We can see from the output that the predictions Data Frame includes the original columns from the data, but with three new columns appended – *rawPrediction*, *probability* and *prediction*. The probability is a vector containing the predicted probability that the example belongs to each class. The prediction column is the prediction derived from this – i.e., which class has the highest predicted probability.

► (1) Spark Jobs

► predictions: pyspark.sql.dataframe.DataFrame = [Temperature: double, Humidity: double ... 8 more fields]

Temperature	Humidity	CO2	HumidityRatio	Occupancy	features	label	rawPrediction	probability	prediction
19.2	30.7	429.0	0.004222155	0	[19.2,30.7,429.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.245	31.65	435.0	0.004366035	0	[19.245,31.65,435.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.26	31.46333333	431.6666667	0.004344191	0	[19.26,31.46333333,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.29	26.79	469.0	0.003702057	0	[19.29,26.79,469.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.29	27.5	432.0	0.00380077	0	[19.29,27.5,432.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.29	30.63333333	441.0	0.004236777	0	[19.29,30.63333333,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.29	30.745	425.0	0.004252327	0	[19.29,30.745,425.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.34	30.6	430.0	0.004245423	0	[19.34,30.6,430.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.39	27.1	468.5	0.003768685	0	[19.39,27.1,468.5,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.39	27.2	460.0	0.003782676	0	[19.39,27.2,460.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.39	27.5	440.6666667	0.003824654	0	[19.39,27.5,440.6,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.39	31.1	436.0	0.00432882	0	[19.39,31.1,436.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.39	31.2	434.0	0.004342836	0	[19.39,31.2,434.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.39	31.29	435.0	0.004355451	0	[19.39,31.29,435.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.39	31.29	436.3333333	0.004355451	0	[19.39,31.29,436.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.5	27.2	452.0	0.00380881	0	[19.5,27.2,452.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.5	27.6	444.0	0.00386517	0	[19.5,27.6,444.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.5	27.79	449.0	0.003891944	0	[19.5,27.79,449.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0

Command took 1.25 seconds -- by n.j.topping@salford.ac.uk at 2/26/2023, 11:36:09 PM on ML Cluster V2



10. We then need to instantiate an evaluator which will use an evaluation metric to provide a measure of how effective the model was. In this case we are using **accuracy** as our evaluation metric. Accuracy is defined as the proportion of predictions which were correct (in other words, which were the same as the correct label we have in the data.) We use the `evaluate()` method to generate the metric in question and then use `print()` to return this value in the output.

```

Cmd 11
1 # use evaluator to measure accuracy of predictions on test data
2
3 from pyspark.ml.evaluation import MulticlassClassificationEvaluator
4
5 evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
6
7 accuracy = evaluator.evaluate(predictions)
8
9 print("Accuracy = %g " % (accuracy))

```

► (1) Spark Jobs

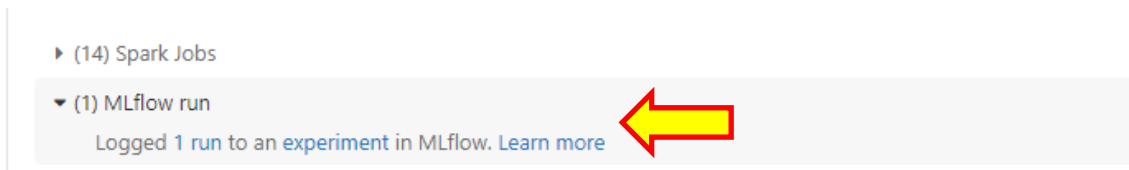
Accuracy = 0.892256

Command took 1.92 seconds -- by n.j.topping@salford.ac.uk at 2/26/2023, 11:44:37 PM on ML Cluster V2

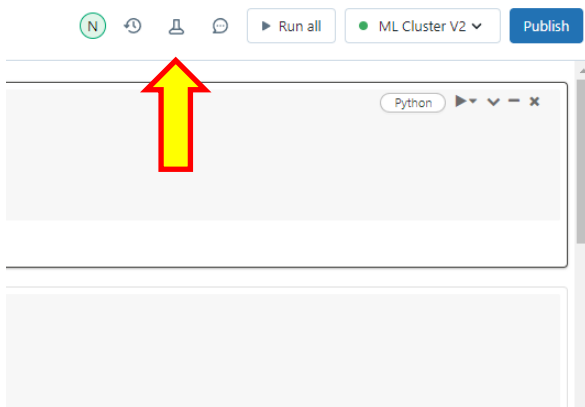
The accuracy is 0.892256, which means that 89% of the predictions made by our model on the test dataset are correct, which seems like a good result.

Part 7: Using MLflow & Viewing Run Details

If you scroll back up the notebook to where we fit the `DecisionTreeClassifier`, you should see from the output that it has logged a run to an experiment in MLflow.



We are now going to use the Databricks interface to view the details of the run which were logged using MLflow. To the top right of the page, click the test tube icon.



This will open a panel to the right which lists all the experiment runs which have been performed within this notebook. At the bottom of this, click on the link to the Experiment UI (you can also click the equivalent icon in the top right.)

Week 7 Machine Learning

Experiment Runs

Date ▾ ↻ ↗

legendary-wasp-269

2023-02-26 22:56:03 GMT

cacheNodeIds: False, ...

accuracy_testDF: 0.892

Models

spark

awesome-crab-982

2023-02-26 22:37:27 GMT

(n/a)

(n/a)

Showing 2 runs, for more information go to Experiment UI ↗

You can see the list of runs which have been logged to this experiment. You can rename them something more meaningful if you would like by selecting the tick box to the right of the Run Name and then selecting Rename.

Q metrics.rmse < 1 and params.model = "tree"

Sort: Created ▾

Columns ▾

Time created: All time ▾

State: Active ▾

<input type="checkbox"/>	Run Name	Created
<input type="checkbox"/>	legendary-wasp-269	1 hour ago
<input type="checkbox"/>	awesome-crab-982	1 hour ago

Search columns

☐ Version
☒ Models
☒ Metrics (1)
☒ accuracy_testDF
☒ Parameters (16)
☐ cacheNodeIds
☐ checkpointInterval
☐ featuresCol
☒ impurity
☐ labelCol
☐ leafCol
☒ maxBins
☒ maxDepth
☐ maxMemoryInMB
☐ minInfoGain

Metrics	Parameters		
accuracy_testDF	impurity	maxBins	maxDepth
0.892	gini	32	5
-	-	-	-

11. From the columns drop down, you can select additional columns to view in the table. For example, select accuracy_testDF from Metrics, and impurity, maxBins and maxDepth from the Parameters. You should see these appear in the table against one of the runs (the other run is simply the pre-processing step).

Time created: All time ▾		State: Active ▾					
<input type="checkbox"/>	Run Name	Created	Duration	Source	Models	Metrics	Parameters
						accuracy_testDF	impurity maxBins maxDepth
<input type="checkbox"/>	legendary-wasp-269	✓ 1 hour ago	1.1min	📁 Week 7 ...	🔗 spark	0.892	gini 32 5
<input type="checkbox"/>	awesome-crab-982	✓ 1 hour ago	0.6s	📁 Week 7 ...	-	-	- - -

These are examples of *hyperparameters*. When we are training a machine learning model, hyperparameters are settings which influence the training process. They can therefore be important in determining how good the trained model is at making predictions. Often, as a Data Scientist, you may want to experiment with different settings of these hyperparameters to see which ones produce the best results.

These particular hyperparameters are:

- maxDepth is a setting that determines the maximum number of times the model is allowed to split the data before it terminates in a leaf node
- impurity is a measure which is used by the algorithm to decide what attribute to split on at each branch node
- maxBins is a setting which influences how many different ways the algorithm can split the data on a specific attribute

We are now going to explore using grid search – which is where we define a ‘grid’ of values for these parameters and perform multiple machine learning runs to identify the optimal set of hyperparameters.

Part 8: Using ParamGridBulder & TrainValidationSplit for Grid Search

First, we need to import and instantiate the ParamGridBuilder, and then we add the different values we want to try for the parameters to the grid. In this case, we are going to try:

- Values of 3, 5 and 7 for the maxDepth (previously it was the default value of 5)
- Values of 16, 32 and 64 for the maxBins (previously it was the default value of 32)
- Gini and entropy as the options for impurity (previously it was the default, gini)

12. Run the code below to create the parameter grid which we will use for hyperparameter tuning.

Cmd 12

```
1  from pyspark.ml.tuning import ParamGridBuilder
2
3  # Create a parameter grid
4
5  parameters = ParamGridBuilder()\
6  .addGrid(dt.impurity,["gini", "entropy"])\
7  .addGrid(dt.maxDepth, [3, 5, 7])\
8  .addGrid(dt.maxBins, [16, 32, 64])\
9  .build()
```

So that we can train a model with different values of the hyperparameters, we have to split the dataset up again into training and validation datasets. This is so that we can train the model with each combination of parameters on the training dataset and then pick the best model using the results on the validation dataset. We want to keep the test dataset aside so that we can use this for evaluating the model once we have chosen the best performing option.

13. To run this, we instantiate a TrainValidationSplit object. We use the DecisionTreeClassifier we instantiated earlier, along with the parameters grid we just defined and the evaluator we used earlier.

Cmd 13

```
1  from pyspark.ml.tuning import TrainValidationSplit
2
3  # Define TrainValidationSplit
4
5  tvs = TrainValidationSplit()\
6  .setSeed(100)\
7  .setTrainRatio(0.75)\
8  .setEstimatorParamMaps(parameters)\
9  .setEstimator(dt)\
10 .setEvaluator(evaluator)
```

14. Now we're ready to perform hyperparameter tuning. We use the fit() method to do this – it may take a few moments to run. However, you should see from the output that it has logged 19 runs in MLflow. The first one is the splitting of the data, and the other 18 runs are the different hyperparameter combinations (2 values for impurity x 3 values for maxBins x 3 values for maxDepth = 18 combinations).

```
Cmd 14

1 # Train model using grid search
2
3 gridsearchModel = tvs.fit(trainingDF)

▶ (60) Spark Jobs
▼ (19) MLflow runs
  Logged 19 runs to an experiment in MLflow. Learn more

2023/02/27 01:04:11 INFO mlflow.utils.autologging_utils: Created MLflow autologging run with ID '630f0cc2'
for the current pyspark.ml workflow
2023/02/27 01:05:10 INFO mlflow.spark: Inferring pip requirements by reloading the logged model from the c
nts when calling log_model().
2023/02/27 01:06:11 INFO mlflow.spark: Inferring pip requirements by reloading the logged model from the c
nts when calling log_model().

Command took 2.85 minutes -- by n.j.topping@salford.ac.uk at 2/27/2023, 1:04:10 AM on ML Cluster V2
```

15. Once we have performed grid search, we can then find the best performing model out of the models which were built:

```
Cmd 15

1 # Select best model and identify the parameters
2
3 bestModel = gridsearchModel.bestModel
4
5 print("Parameters for the best model:")
6 print("MaxDepth Parameter: %g" %bestModel.getMaxDepth())
7 print("Impurity Parameter: %s" %bestModel.getImpurity())
8 print("MaxBins Parameter: %g" %bestModel.getMaxBins())

Parameters for the best model:
MaxDepth Parameter: 7
Impurity Parameter: entropy
MaxBins Parameter: 32

Command took 0.14 seconds -- by n.j.topping@salford.ac.uk at 2/27/2023, 1:09:31 AM on ML Cluster V2
```

16. We can also use the best model to make predictions on the test dataset which we still have held back. You should see from the output that there is a marginal improvement in the performance of the model now we have performed grid search to select the best hyperparameters.

```
Cmd 16

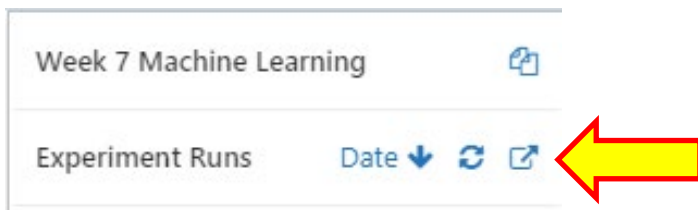
1 # Use the best model to make predictions on the hold out test set
2
3 evaluator.evaluate(bestModel.transform(testDF))

▶ (1) Spark Jobs

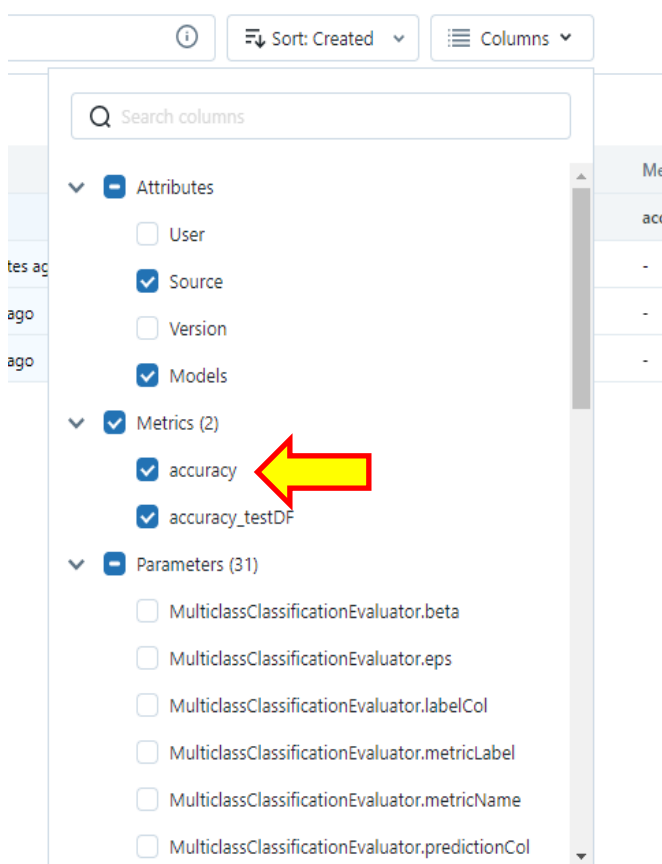
Out[19]: 0.9023569023569024

Command took 1.25 seconds -- by n.j.topping@salford.ac.uk at 2/27/2023, 1:11:43 AM on ML Cluster V2
```

Now we have performed more runs as part of our experiment, we can go back to the Experiment UI to view more information on all the runs which were completed as part of the grid search. Click the test tube icon to bring back the experiment panel, and then the icon in the top right of this panel to open the UI.



From the columns drop down in the Experiment UI, add in accuracy from the Metrics section.



Then select the plus sign next to the latest Run Name to expand the runs within this – you should now see a breakdown of all the runs completed as part of the grid search, along with the parameters for each one and the accuracy achieved. When we're completing multiple runs with different hyperparameters, MLflow tracking is very helpful for us in keeping track of the runs and the results achieved each time.

	Run Name	Created	Duration	Source	Models	Metrics		Parameters		
						accuracy	accuracy_testDF	impurity	maxBins	maxDepth
<input type="checkbox"/>	gregarious-hare-956	15 minutes ago	2.8min	Week 7 ...	spark, 1 more	-	-	-	-	-
<input type="checkbox"/>	capable-eel-669	15 minutes ago	42.5s	Week 7 ...	-	0.846	-	gini	64	7
<input type="checkbox"/>	skillful-kite-220	15 minutes ago	42.5s	Week 7 ...	-	0.84	-	gini	32	5
<input type="checkbox"/>	clean-ant-836	15 minutes ago	42.5s	Week 7 ...	-	0.817	-	gini	32	3
<input type="checkbox"/>	bald-fox-537	15 minutes ago	42.5s	Week 7 ...	-	0.834	-	entropy	16	5
<input type="checkbox"/>	able-smelt-969	15 minutes ago	42.5s	Week 7 ...	-	0.852	-	entropy	16	7
<input type="checkbox"/>	fearless-hare-928	15 minutes ago	42.5s	Week 7 ...	-	0.858	-	gini	16	7
<input type="checkbox"/>	nimble-wolf-580	15 minutes ago	42.5s	Week 7 ...	-	0.811	-	entropy	16	3
<input type="checkbox"/>	smiling-frog-447	15 minutes ago	42.5s	Week 7 ...	-	0.811	-	entropy	32	3
<input type="checkbox"/>	amazing-conch-660	15 minutes ago	42.5s	Week 7 ...	-	0.811	-	entropy	64	3
<input type="checkbox"/>	big-donkey-897	15 minutes ago	42.5s	Week 7 ...	-	0.852	-	gini	32	7
<input type="checkbox"/>	melodic-bee-313	15 minutes ago	42.5s	Week 7 ...	-	0.822	-	gini	16	3

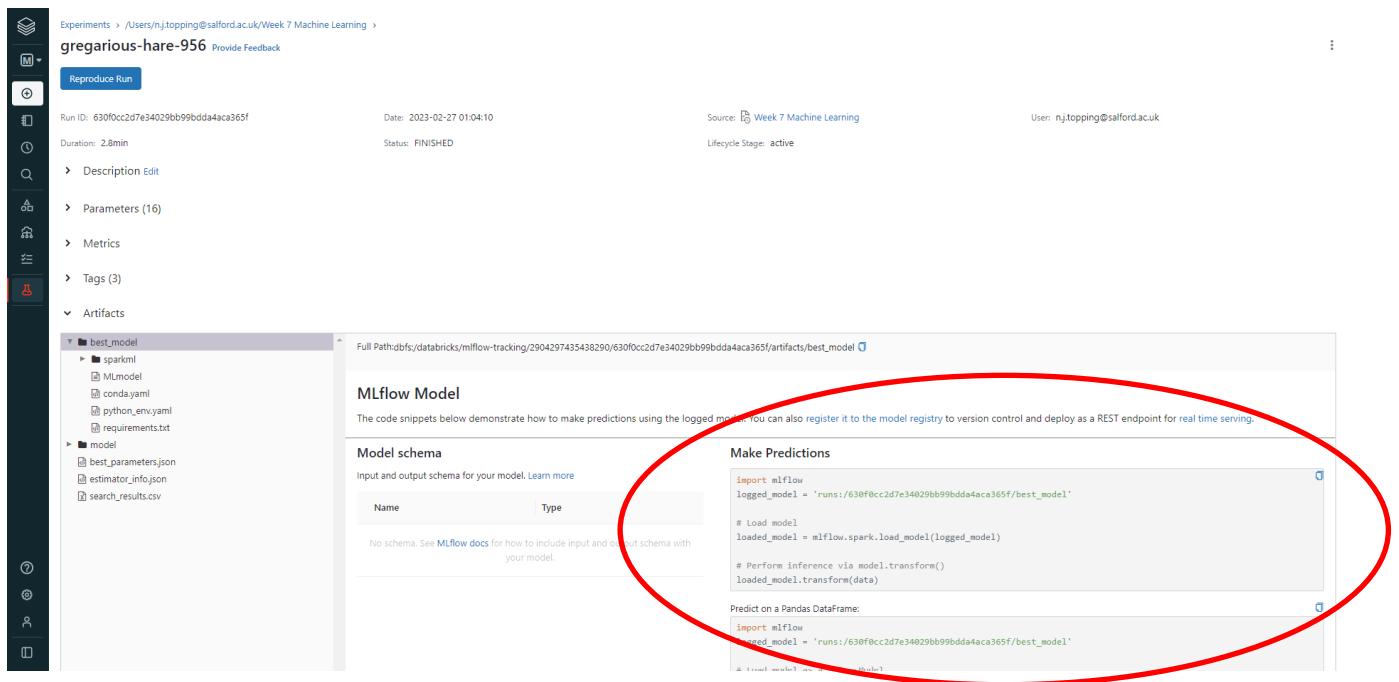
If you click on the 'accuracy' column header you can sort the results on the accuracy metric. You should verify that the parameter settings for the highest performing model as shown in the Experiment UI match those we identified in step 15 in the notebook.

One of the benefits of using MLflow tracking is that it allows us to load a model trained in a given MLflow run and use it to make predictions. This is useful if we want to use a model once we have trained it. For example, we can load this model using MLflow in another notebook and make batch predictions using it. If you are using the Enterprise Edition of Databricks, you can also deploy it as a REST endpoint and register it to the model registry, which provides version control as the model transitions through the different model stages (e.g., Staging for model testing and validating, and Production for models which have been deployed). For now, we will briefly demonstrate how to load the best model from the grid search and use it to make predictions. Note, while we are doing it from within the same notebook, using MLflow allows us to use the trained model in another notebook if we wish.

17. Click on the Run Name of the parent run shown in the Experiment UI (this will be the top one if you re-sort by the 'Created' column in the table).

	Run Name	Created	Duration	Source	Models	Metrics		Parameters		
						accuracy	accuracy_testDF	impurity	maxBins	maxDepth
<input type="checkbox"/>	gregarious-hare-956	15 minutes ago	2.8min	Week 7 ...	spark, 1 more	-	-	-	-	-
<input type="checkbox"/>	capable-eel-669	15 minutes ago	42.5s	Week 7 ...	-	0.846	-	gini	64	7
<input type="checkbox"/>	skillful-kite-220	15 minutes ago	42.5s	Week 7 ...	-	0.84	-	gini	32	5
<input type="checkbox"/>	clean-ant-836	15 minutes ago	42.5s	Week 7 ...	-	0.817	-	gini	32	3
<input type="checkbox"/>	bald-fox-537	15 minutes ago	42.5s	Week 7 ...	-	0.834	-	entropy	16	5
<input type="checkbox"/>	able-smelt-969	15 minutes ago	42.5s	Week 7 ...	-	0.852	-	entropy	16	7
<input type="checkbox"/>	fearless-hare-928	15 minutes ago	42.5s	Week 7 ...	-	0.858	-	gini	16	7
<input type="checkbox"/>	nimble-wolf-580	15 minutes ago	42.5s	Week 7 ...	-	0.811	-	entropy	16	3
<input type="checkbox"/>	smiling-frog-447	15 minutes ago	42.5s	Week 7 ...	-	0.811	-	entropy	32	3
<input type="checkbox"/>	amazing-conch-660	15 minutes ago	42.5s	Week 7 ...	-	0.811	-	entropy	64	3
<input type="checkbox"/>	big-donkey-897	15 minutes ago	42.5s	Week 7 ...	-	0.852	-	gini	32	7
<input type="checkbox"/>	melodic-bee-313	15 minutes ago	42.5s	Week 7 ...	-	0.822	-	gini	16	3

18. This will take you to the page which provides details of this run as well as all the artifacts associated with it. Towards the bottom right of the page is a code snippet which you can use to load the model and make predictions.



Experiments > /Users/nj.topping@salford.ac.uk/Week 7 Machine Learning > gregarious-hare-956 Provide Feedback

Reproduce Run

Run ID: 630f0cc2d7e34029bb99bdda4aca365f Date: 2023-02-27 01:04:10 Source: Week 7 Machine Learning User: nj.topping@salford.ac.uk

Duration: 2.8min Status: FINISHED Lifecycle Stage: active

> Description Edit

> Parameters (16)

> Metrics

> Tags (3)

> Artifacts

Full Path: dbfs:/databricks/mlflow-tracking/2904297435438290/630f0cc2d7e34029bb99bdda4aca365f/artifacts/best_model

MLflow Model

The code snippets below demonstrate how to make predictions using the logged model. You can also register it to the model registry to version control and deploy as a REST endpoint for real time serving.

Model schema

Input and output schema for your model. [Learn more](#)

Name	Type
No schema. See MLflow docs for how to include input and output schema with your model.	

Make Predictions

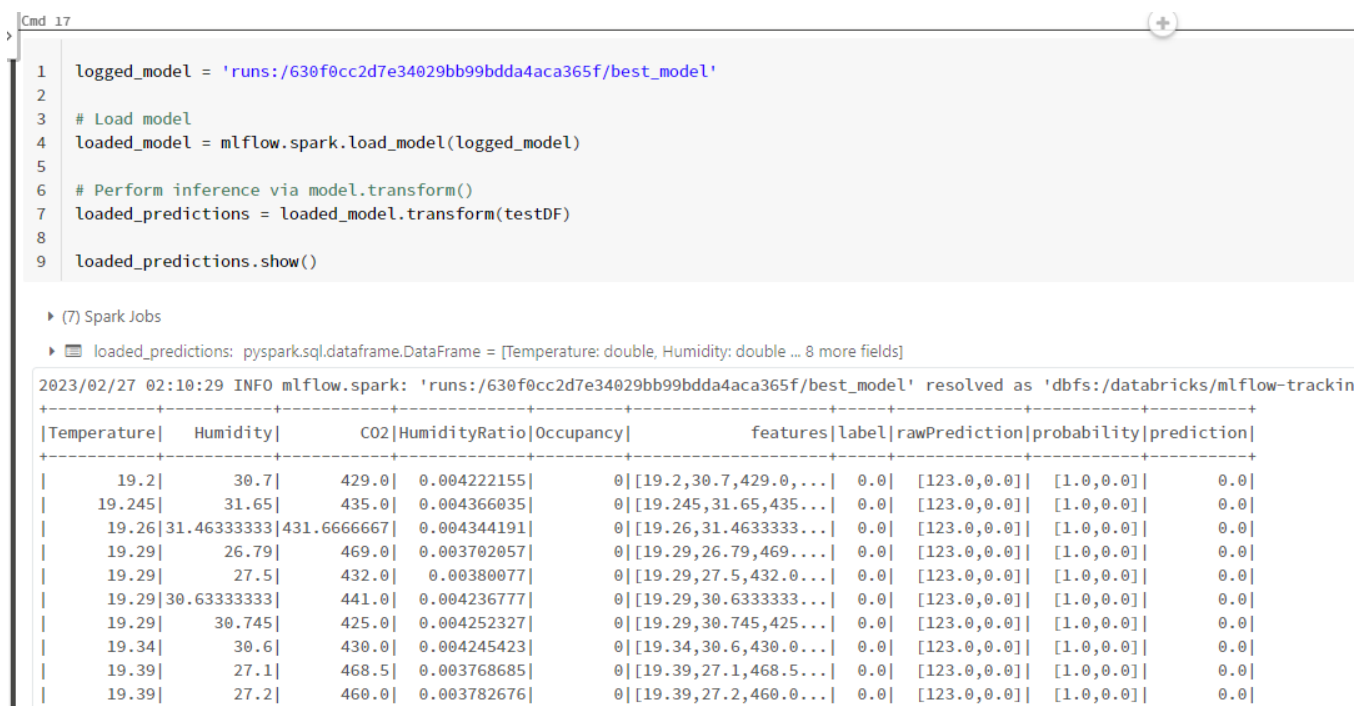
```
import mlflow
logged_model = 'runs:/630f0cc2d7e34029bb99bdda4aca365f/best_model'

# Load model
loaded_model = mlflow.spark.load_model(logged_model)

# Perform inference via model.transform()
loaded_model.transform(data)

Predict on a Pandas DataFrame:
import mlflow
logged_model = 'runs:/630f0cc2d7e34029bb99bdda4aca365f/best_model'
```

19. You can paste that code into a new code cell in your notebook. In the last line of code update this to `loaded_predictions = loaded_model.transform(testDF)` and add another line: `loaded_predictions.show()`. Run this code to generate the predictions from the loaded model and show the resulting predictions.



```
1 logged_model = 'runs:/630f0cc2d7e34029bb99bdda4aca365f/best_model'
2
3 # Load model
4 loaded_model = mlflow.spark.load_model(logged_model)
5
6 # Perform inference via model.transform()
7 loaded_predictions = loaded_model.transform(testDF)
8
9 loaded_predictions.show()
```

(7) Spark Jobs

loaded_predictions: pyspark.sql.dataframe.DataFrame = [Temperature: double, Humidity: double ... 8 more fields]

2023/02/27 02:10:29 INFO mlflow.spark: 'runs:/630f0cc2d7e34029bb99bdda4aca365f/best_model' resolved as 'dbfs:/databricks/mlflow-trackin

Temperature	Humidity	CO2	HumidityRatio	Occupancy	features	label	rawPrediction	probability	prediction
19.2	30.7	429.0	0.004222155	0	[19.2,30.7,429.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.245	31.65	435.0	0.004366035	0	[19.245,31.65,435.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.26	31.46333333	431.6666667	0.004344191	0	[19.26,31.46333333,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.29	26.79	469.0	0.003702057	0	[19.29,26.79,469.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.29	27.5	432.0	0.00380077	0	[19.29,27.5,432.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.29	30.63333333	441.0	0.004236777	0	[19.29,30.63333333,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.29	30.745	425.0	0.004252327	0	[19.29,30.745,425.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.34	30.6	430.0	0.004245423	0	[19.34,30.6,430.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.39	27.1	468.5	0.003768685	0	[19.39,27.1,468.5,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0
19.39	27.2	460.0	0.003782676	0	[19.39,27.2,460.0,...]	0.0	[123.0,0.0]	[1.0,0.0]	0.0