HOW TO RUN MACHINE LEARNING TRANSFORMS IN AWS GLUE

Here we show you how to do a machine learning transformation with Amazon Glue. Previous Glue tutorials include:

- How To Make a Crawler in Amazon Glue
- How To Join Tables in Amazon Glue
- How To Define and Run a Job in AWS Glue
- AWS Glue ETL Transformations

Now, let's get started.

Amazon's machine learning

A fully managed service from Amazon, AWS Glue handles data operations like ETL to get your data prepared and loaded for analytics activities. Glue can crawl S3, DynamoDB, and JDBC data sources.

Amazon called their offering **machine learning,** but they only have one ML-type function, **findMatches**. It uses an ML algorithm, but Amazon does not tell you which one. They even boast on their web page you don't need to know—but a data scientist would certainly want to know.

You can study their execution log to gain some insight into what their code is doing. Suffice it to say it is doing a type of clustering algorithm and using Apache Spark as a platform to execute that.

The process: Amazon Glue machine learning

Here is the general process for running machine learning transformations:

1. Upload a csv file to an S3 bucket. Then you set up a crawler to crawl all the files in the designated S3 bucket. For each file it finds, it will create a metadata (i.e., schema) file in Glue that contains the column names.
2. Set up a **FindMatches** machine learning task in Glue. It's an iterative process. It takes your input date, created in the crawler process, and makes a label file. These labels are like a k-means clustering algorithm. It looks at the input data and all of the columns in the data set. Then it put the data into groups, each labeled with a **labeling_set_id**.
3. Download the label file. There will be an empty column called **label**. You are invited to add your own label to classify data however you see fit. For example, it could be borrower risk rating, whether or not a patient has diabetes, or whatever. Labels should be a single value, like A, B, C or 1, 2, 3. A data scientist would say they must be **categorical**.
4. Upload the labelled file to a different S3 bucket. Do not use the same bucket where you put the original input data, as the crawler will attempt to crawl that and create another metadata file.
5. Rerun Step 2, above, and it creates another labelled file. Do this iteratively until it supplies the most accurate result. In this example, there was no improvement from one run to the next. Repeating machine learning runs is standard practice for improving accuracy. However, at some point, the gain in accuracy will level off.
6. Generate and then inspect the **Quality Metrics**. Perhaps change some of the parameters and run the **Tune** operation, which means to run the algorithm again.

Tutorial: Amazon Glue machine learning

Now, let's run an example to show you how it works.

I have copied the Pima Native American database from Kaggle and put it on GitHub, here. You have to add a primary key column to that data, which Glue requires. Download the data here. I have also copied the input data and the first and second label files here, in a Google Sheet, so that you can see the before and after process.

The data looks like this:

```
recordID,Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,BMI,Diabetes
PedigreeFunction,Age,Outcome
2,6,148,72,35,0,33.6,0.627,50,1
3,1,85,66,29,0,26.6,0.351,31,0
4,8,183,64,0,0,23.3,0.672,32,1
5,1,89,66,23,94,28.1,0.167,21,0
6,0,137,40,35,168,43.1,2.288,33,1
```

Then copy it to an Amazon S3 bucket as shown below. You need to have installed the Amazon CLI (command line interface) and run **aws configure** to configure your credentials. Importantly, the data must be in the same Amazon zone as the instance you are logged into.

```
aws s3 cp diabetes.csv s3://sagemakerwalkerml
```

Add a label

The diabetes data is already labelled in the column **outcome**. So, I used Google Sheets to copy that value into the label **column**. You do this after the first run, like this:

1. Upload the original data.
2. Run a training model
3. Download the resulting labels file.

At that point you can populate the label with some kind of categorical data. You might put the outcome of logistic regression on your input data set into this label, but that's optional. You don't need a label at all.

The algorithm does not require a label the first time it runs. Glue says:

> *As you can see, the scope of the labels is limited to the labeling_set_id. So, labels do not cross labeling_set_id boundaries.*

In other words, when there is no label, it groups records by **labeling_set_id** without regards to the **label** value. When there is a label then the **labeling_set_id** is within the label.

In other words, given this:

labeling_set_id	labeling_set_id	other columns
123	blank	
123	blank	
456	blank	

The first two rows are grouped together. But if we add a label:

labeling_set_id	label	other columns
123	A	
123	A	
456	B	
456	B	
789	A	

Then the first two rows are matched together while rows 456, even if they had matched without the label, are groups separately. Remember Amazon said they "don't cross boundaries."

Of course, that does not mean only the label determines what is considered to be a match. (That would be of little use.) It's the other columns that determine what matches. The label just confines that matching to records with that label. So, it's matching within a subset of records. It's like having n number of files with no label instead of one file with n labels, so you can run the process one time and not n times.

Anyway, that's the conclusion I draw from this design. Perhaps yours will differ.

Crawl S3

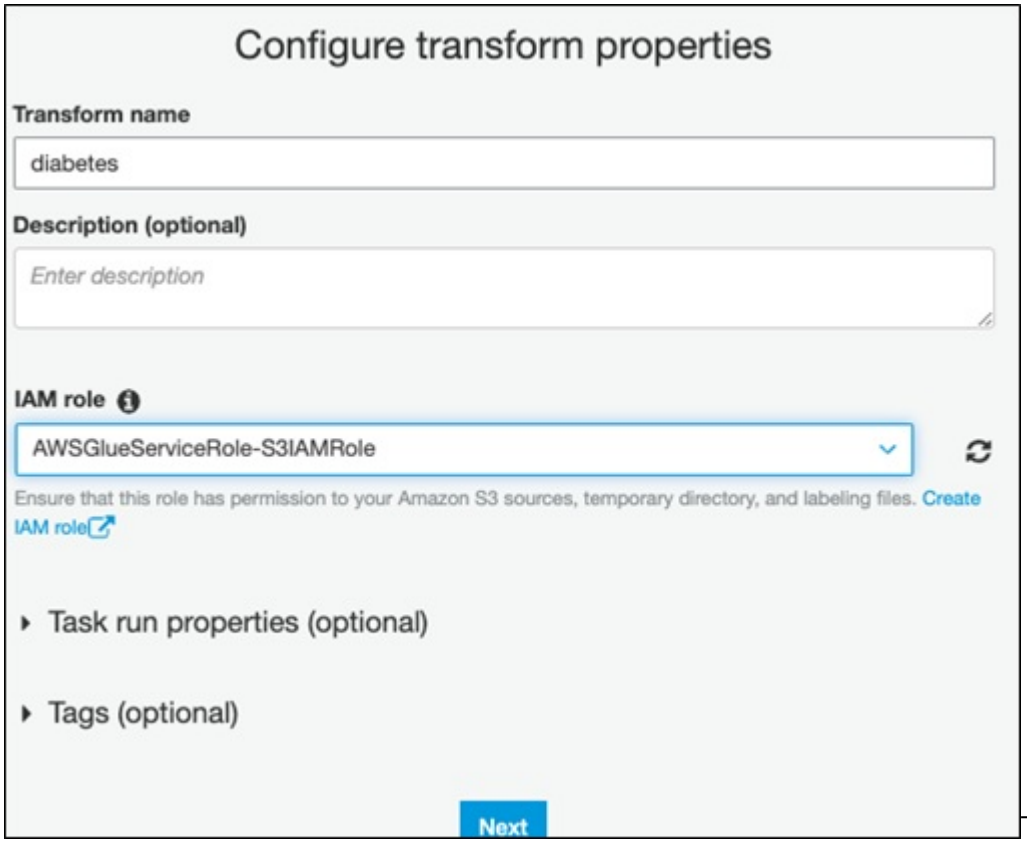
We start with the crawlers. Here is the metadata extracted from the diabetes.csv file in S3:

Schema

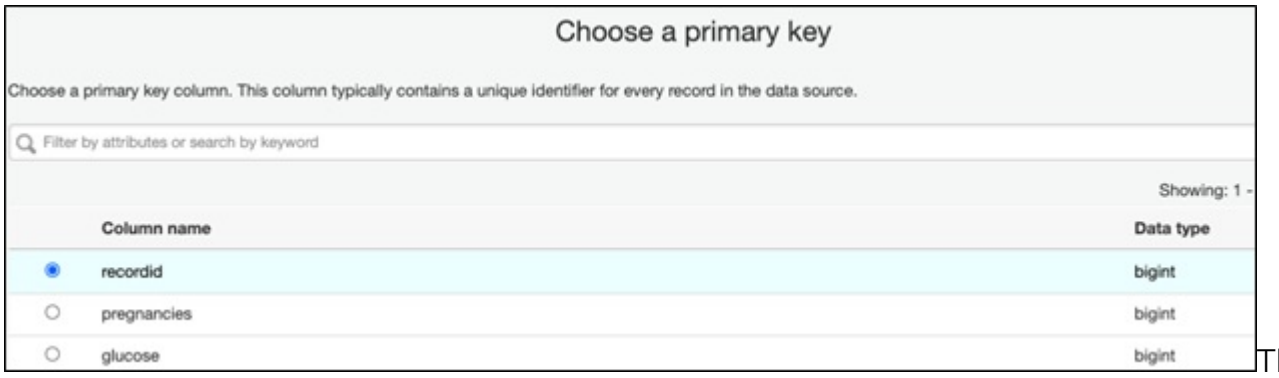
	Column name	Data type	Partition k
1	recordid	bigint	
2	pregnancies	bigint	
3	glucose	bigint	
4	bloodpressure	bigint	
5	skinthickness	bigint	
6	insulin	bigint	
7	bmi	double	
8	diabetespedigreefunction	double	
9	age	bigint	
10	outcome	bigint	

It created these tables in the database.

Pick an IAM role that has access to S3 and give the transformation a name.



The data must have a primary key. The matching algorithm requires that to do its matching logic.



Then it asks you to tune the transformation. These are tradeoffs between cost and accuracy:

- **Cost** is financial.
- **Cost function** is data science and computing.

(Pricing is based on resources (DPUs) you consume, which I cover below.)

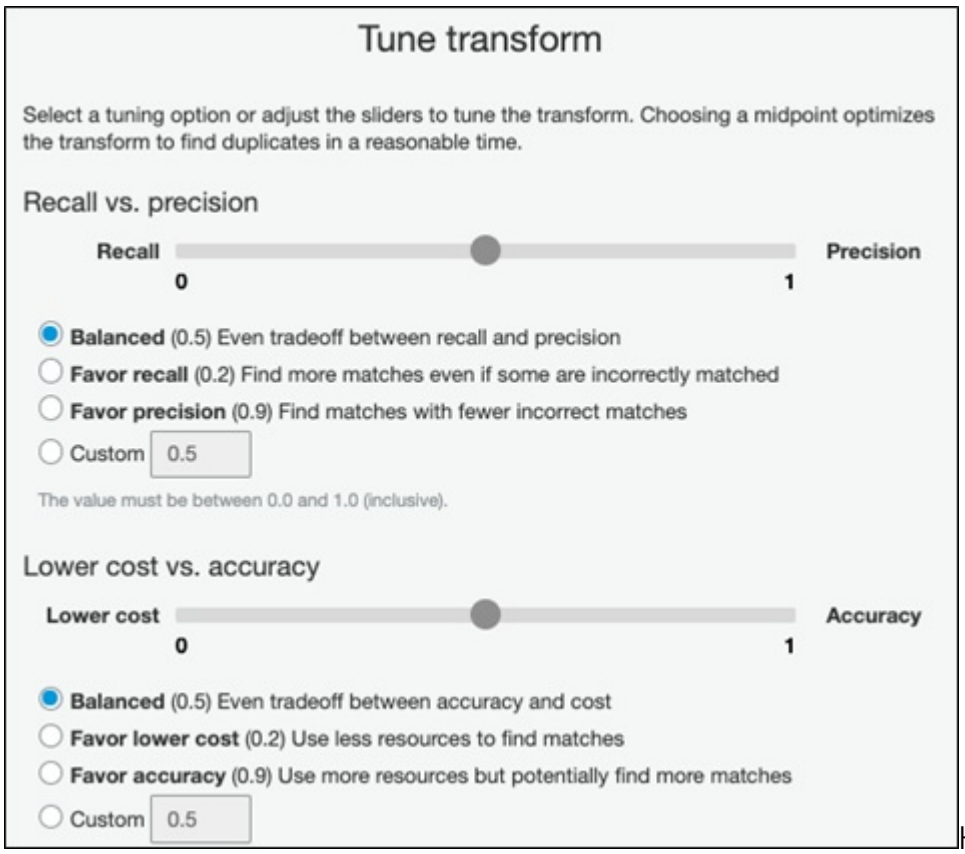
The data science-related tuning parameters are between **recall** and **precision**.

Recall is:

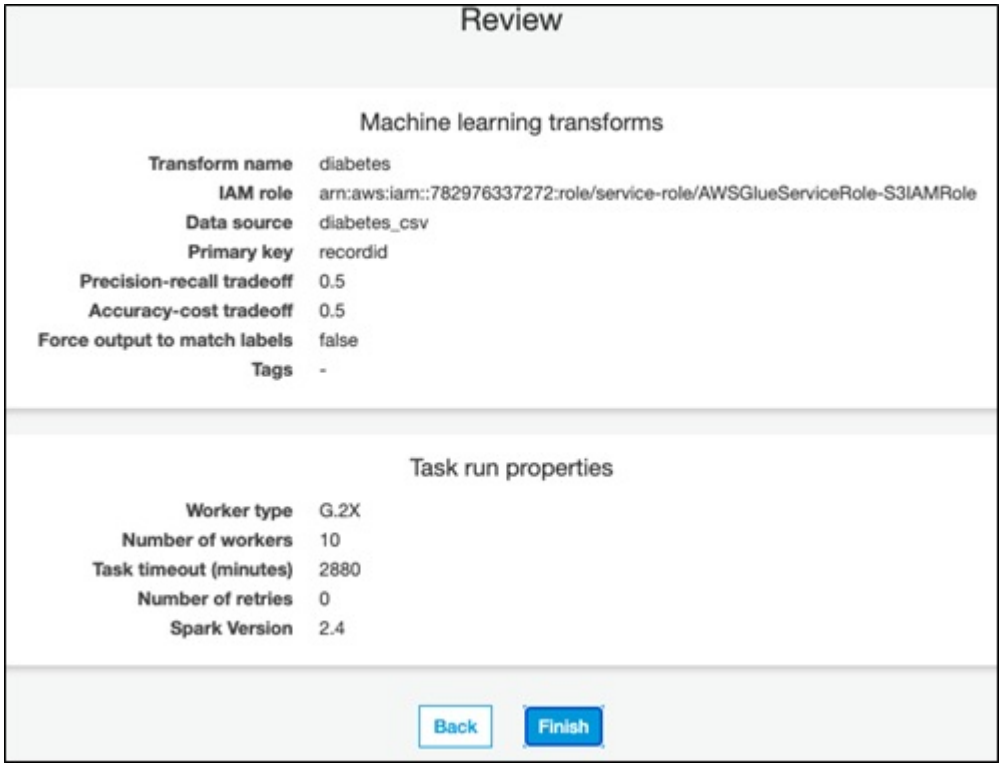
$$\frac{true\ positives}{predicted\ results} = \frac{true\ positives}{true\ positives + false\ positives}$$

Precision is:

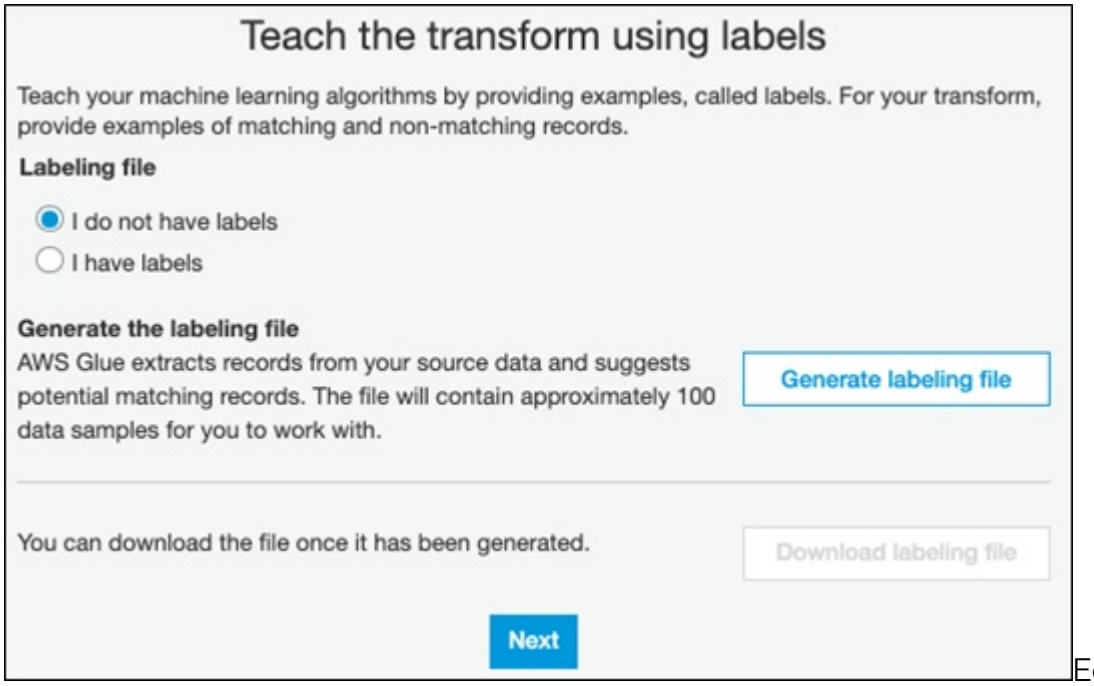
$$\frac{true\ positives}{actual\ results} = \frac{true\ positives}{true\ positive + false\ negative}$$



Here is a summary of the parameters:



- **The first time** let it generate a label file for you. It will match records based on all of the data points taken together.
- **The second time** it will incorporate labels in its matching algorithm should you choose to add one.



Edit the file in Excel or Google Sheets to both review it and optionally add a label. Copy it back to S3, putting it in a different bucket than the original upload file. Then run transformation again (called **train**). It will produce yet another label file which is the results of the matching aka grouping process.

It asks for the bucket name:

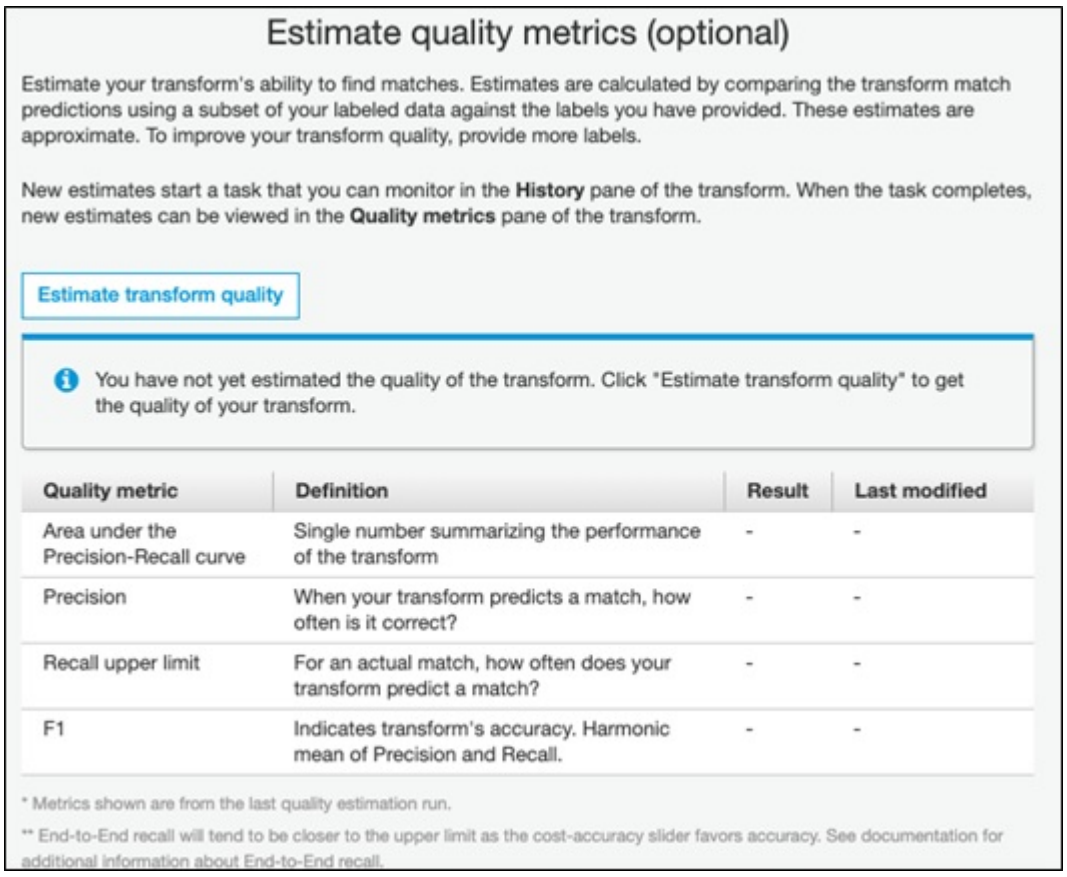
You download the labels from this screen.

Here is the first label file it created. You can't see all of the columns because it's too wide. But you can see the labeling_set_id, thus how it grouped the data:

	A	B	C	D	E	F	G	H	I
1	labeling_set_id	label	pregnancies	bmi	outcome	age	diabetespedigre	bloodpressure	skinthickness
2	244fcf87-d3a6-36cb-81be-43a29dece209	0	0	40.5	0	44	1.781	82	0
3	244fcf87-d3a6-36cb-81be-43a29dece209	0	1	22.4	0	27	0.207	122	0
4	244fcf87-d3a6-36cb-81be-43a29dece209	1	10	32.4	1	42	0.272	66	0
5	244fcf87-d3a6-36cb-81be-43a29dece209	0	1	24.3	0	21	0.187	58	0
6	244fcf87-d3a6-36cb-81be-43a29dece209	0	0	18.4	0	27	0.582	76	0
7	244fcf87-d3a6-36cb-81be-43a29dece209	0	2	25.3	0	22	0.881	62	10
8	244fcf87-d3a6-36cb-81be-43a29dece209	0	6	30.8	0	37	0.122	70	32
9	244fcf87-d3a6-36cb-81be-43a29dece209	0	1	32	0	22	0.389	68	35
10	244fcf87-d3a6-36cb-81be-43a29dece209	0	0	20	0	22	0.236	68	22
11	244fcf87-d3a6-36cb-81be-43a29dece209	1	13	42.3	1	44	0.257	114	0
12	28603e1d-6ae9-38af-9801-c5bb2d7ff7c3	0	7	27.4	0	40	0.294	64	0
13	28603e1d-6ae9-38af-9801-c5bb2d7ff7c3	1	7	30.4	1	36	0.383	66	0
14	28603e1d-6ae9-38af-9801-c5bb2d7ff7c3	0	2	28.7	0	25	0.092	68	22
15	28603e1d-6ae9-38af-9801-c5bb2d7ff7c3	1	1	43.3	1	41	0.282	0	0
16	28603e1d-6ae9-38af-9801-c5bb2d7ff7c3	0	0	22.1	0	21	0.207	62	17
17	28603e1d-6ae9-38af-9801-c5bb2d7ff7c3	1	3	31.6	1	28	0.268	70	18
18	28603e1d-6ae9-38af-9801-c5bb2d7ff7c3	0	3	26.3	0	24	0.107	74	15
19	28603e1d-6ae9-38af-9801-c5bb2d7ff7c3	0	0	36.3	0	23	0.804	80	37

Evaluation metrics

This screen lets you calculate accuracy. I have yet to figure out where you can see the results as the screen mentioned in the documentation does not exist. (I will update this tutorial once I get a response on the user forum.)



Pricing

Price is by DPU. I used 10 DPUs for about 30 minutes. It's $0.44 for each multiple or fraction of an hour. So presumably I spent $0.44*10=$4.40.

These ETL jobs run on Amazon's Spark and Yarn infrastructure. If you want to write code to do transformations you need to set up a **Development Endpoint**. Basically, the development endpoint is a virtual machine configured to run Spark and Glue. We explained how to use a Development Endpoint here. Then you can run Python or Scala and optionally use a Jupyter Notebook.

Important note: When you don't need your development endpoint, be sure to delete it—it gets expensive quickly! (I spent $1,200 on that in a month.)

Additional resources

For more tutorials like this, explore these resources:

- BMC Machine Learning & Big Data Blog
- Apache Spark Guide, with 15 articles and tutorials
- AWS Guide
- Amazon Braket Quantum Computing: How To Get Started