Migrating & Scaling Machine Learning Models to Azure Databricks for Cloud-Powered AI

Colby Ford

Needing to scale up your predictive power and data processing capabilities, but a bit apprehensive about moving awesome machine learning models to a new platform? No need to worry! In today's post, I'll show you the easy way to migrate and scale machine learning and deep learning models from Python over to Azure Databricks. Plus, I'll also talk about why reworking your existing models using MLlib in Spark might be a good idea.

Data scientists spend a lot of time training models and tuning them to optimize their performance for whatever use case is at hand. Traditionally, this is done on local workstations using machine learning libraries such as scikit-learn, H_2O, or XGBoost in Python.

Many AI teams are making the shift to begin developing on Spark and Databricks, which allows for embarrassingly parallel model training, tuning, and cross-validation on a cluster. However, this doesn't necessarily mean that we have to throw away the Python models that we've already built and have put to use. Migrating them to Databricks is easy!

Migrating and Scaling from Python (scikit-learn)

Anyone who has used Python for machine learning has heard of scikit-learn. It's one of the most popular libraries for machine learning, consisting of a plethora of clustering, classification, regression, and dimensionality reduction algorithms.

In 2016, the team at Databricks saw the need for users to be able to migrate their scikit-learn models to Spark. Thus, they released the spark-sklearn package.

This package allows users to:

1. Train and evaluate models in parallel.
2. Spread the work across multiple machines with no changes required in the code between the single-machine case and the cluster case.
3. Convert Spark DataFrames seamlessly into Numpy ndarrays or sparse matrices.
4. Distribute SciPy's sparse matrices as a dataset of sparse vectors.

These features allow for better scalable processing of machine learning models without making users leave their scikit-learn comfort zone.

```
from sklearn import svm, datasets
from spark_sklearn import GridSearchCV
iris = datasets.load_iris()
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
svr = svm.SVC(gamma = 'auto')
clf = GridSearchCV(sc, svr, parameters)
clf.fit(iris.data, iris.target)
```

In the example code above, you can see the use of both the normal scikit-learn package to bring in the desired algorithm, as well as the spark-sklearn package to perform a grid search across parameter and CV folds.

Scikit-learn isn't the only library that can be used in Databricks. Other packages such as H_2O and XGBoost have Spark counterparts as well. To learn how to use other third-party libraries in Databricks, click here.

Importing Trained Neural Networks (ONNX)

ONNX, or the Open Neural Network Exchange, is a format for representing deep learning models such as neural networks. This format allows for users to train models on popular frameworks such as Cognitive Toolkit, TensorFlow, PyTorch, and MXNet, and save them for distribution and use in other places.

For example, let's say we've created an awesome deep learning model on our local GPU-based workstation using Cognitive Toolkit. Saving the model in the ONNX format is easy.

```
import cntk as C

x = C.input_variable(<input shape>)
z = create_model(x) #your create model function
z.save(<path of where to save your ONNX model>, format=C.ModelFormat.ONNX)
```

Once we've saved our model in a location accessible by Databricks (Blob Storage or Data Lake Store), we can import the model just as easily.

```
import cntk as C
z = C.Function.load(<path of your ONNX model>, format=C.ModelFormat.ONNX)
```

ONNX exporting and importing not only works with Cognitive Toolkit, but a variety of other frameworks as well. For a list of tutorials on how to get started, click here.

In addition to using ONNX, you can also import from MLeap, which is a common serialization format for machine learning pipelines. To learn how, click here.

Retraining using MLlib

Don't forget that Spark includes a really powerful set of algorithms in MLlib, Apache Spark's scalable machine learning library. Personally, I've used MLlib for quite a few clients here at BlueGranite and am starting to love it.

MLlib includes the following classes of algorithms and functions:

- Classification - logistic regression, naïve Bayes, decision trees, and random forests
- Regression - generalized linear regression and survival regression
- Recommendation - alternating least squares (ALS)
- Clustering - K-means and Gaussian mixtures (GMMs)
- Topic modeling - latent Dirichlet allocation (LDA)
- Frequent itemsets, association rules, and sequential pattern mining
- Distributed linear algebra - singular value decomposition (SVD), principal component analysis (PCA)
- Statistics - summary statistics, hypothesis testing, standardication, normalization, and much more.

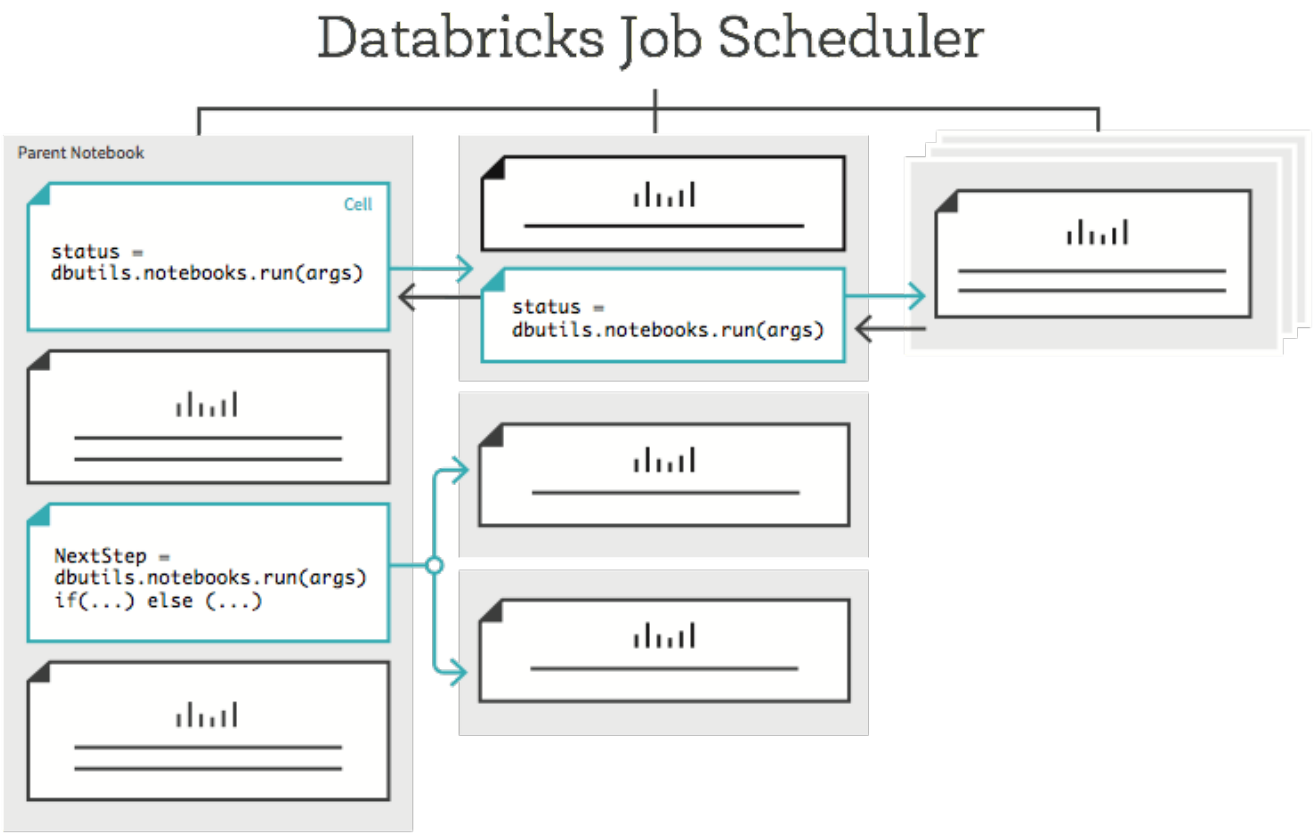
If you have a machine learning model that you've trained outside of Spark/Databricks, you can always retrain the model using MLlib to sweep through additional parameter combinations or perform a more robust cross-validation of the model. This can help in situations where you are getting lackluster performance from your previous model, but it takes too long on your local workstation to optimize the model any further.

For some examples to get started using MLlib, click here.

Putting your Model to Work

As you can see, scaling up your AI practice is easier than ever thanks to Azure Databricks. Whether you're creating new machine learning solutions or wanting to operationalize your existing models, Azure Databricks is the premier platform for AI in the cloud.

One common issue I hear from clients is that they find it difficult to operationalize their models. In other words, despite having great data science teams creating robust machine learning models, organizations still struggle to use their models in an automated way.



Azure Databricks Job Scheduler makes operationalizing your models super easy. Within a couple clicks, you can have a notebook in Azure Databricks scheduled to run and score your new incoming data every day. So, even if you aren't having training or performance issues with your models, automating the use of the model may be reason enough to give Azure Databricks a try!