

Code Free Data Science for Hadoop and Spark

Remove the complexity of data prep and machine learning on Hadoop and Spark



Introduction to Hadoop

Hadoop offers great promise to organizations looking to gain a competitive advantage from data science. Hadoop lets organizations collect a massive amount of data that can later be used to extract insights of immense business value for use cases that include fraud detection, sentiment analysis, risk assessment, predictive maintenance, churn analysis, user segmentation, and many more. But deploying Hadoop can be extraordinarily complex and time consuming, making it difficult to gain the insights.

Hadoop is a collection of technologies and open source projects that form an ecosystem for storage and processing, requiring a host of specialized IT and analytics skills. Integrating these different Hadoop technologies is often complex and time consuming, so instead of focusing on generating business value organizations spend their time on the architecture. Data scientists spend most of their time learning the myriad of skills required to extract value from the Hadoop stack, instead of doing actual data science.

To overcome this challenge, data science teams need to find a solution that will remove the requirement for specialized Hadoop skills, so they can instead focus their efforts on creating predictive models that they can easily execute in Hadoop to extract immediate business value for the organization.

The Challenge

As previously mentioned, Hadoop is a set of loosely coupled software packages like HDFS (file system), Hive (data access and manipulation), Spark (parallel job management), Yarn (job scheduling), and others. All major Hadoop distributions have more than 20 integrated packages like those mentioned above, but there are dozens more available. It is a huge effort to simply keep track of these packages; it takes time and specialized knowledge to understand the Hadoop ecosystem.

When developing predictive analytic work flows on Hadoop, understanding and incorporating the following packages typically come into play:



Hive is the data warehouse SQL layer of Hadoop. While it is the most accessible entry point to Hadoop, it still comes with complexities. For example, HiveQL is not yet fully compatible with SQL standards and users often have a hard time working around the missing pieces. Advanced Hive usage typically builds on a decent understanding of Hadoop storage formats like Parquet or ORC and also requires the ability to program User Defined Functions (UDFs) in Java. Hive can work on MapReduce, Tez or Spark as execution back ends, making management and job monitoring even more complex.



Apache Spark has enormously grown in acceptance in the last years, and for good reason. It brings optimized memory and disk usage, support for iterative computations, a clean programming API for various popular programming languages like Python and R, and a large community – all making it the natural successor of MapReduce. However, Spark requires proficiency in either Python, R, Java or Scala. Additionally, developers need a good understanding of parallel programming to avoid writing programs with performance bottlenecks. Because Spark is developing so quickly, it is hard to keep track of all the changes.



Hadoop MapReduce: The original execution framework of Hadoop, MapReduce is abstracted and simplified by Hive and Pig. But sometimes you might still need to implement analytics in the MapReduce framework, which implies that you have a grasp on parallel programming. Though you can use many programming languages to write map and reduce functions, effectively translating your analytics to those functions is challenging and requires a lot of expertise. You will need to dive deep into data storage formats, compression, serialization, and lots of low-level tricks to achieve your analytics goals.

Each of these packages, as well as other Hadoop technologies, like the Pig programming language, requires detailed knowledge to use. The problem is amplified by the need to manually integrate these technologies for analysis. Passing data from one component to another involves transforming that data and converting it to the correct format.

Even a simple analytics process demands the data scientist to understand the HDFS formats for data storage, develop the necessary HiveQL scripts for data preparation (some of which may require additional coding for some specific functions like pivoting), then move the data as an RDD to Spark, where Python, R or Scala need to be leveraged to train models using the algorithms provided by Spark's MLlib. And that's only the prototyping phase, because, then, someone else has to think about operationalizing that model to make it available and usable for final users (Marketing, business users, analysts or others) and that requires yet more coding.

As a consequence, mastering analytics on Hadoop through programming not only requires a very broad yet specialized skill set, but also means repetitively solving many tasks that are a necessity of the technology rather than part of the actual analytics initiative. For these reasons, developing predictive analytics on Hadoop can be a complex and costly endeavor.

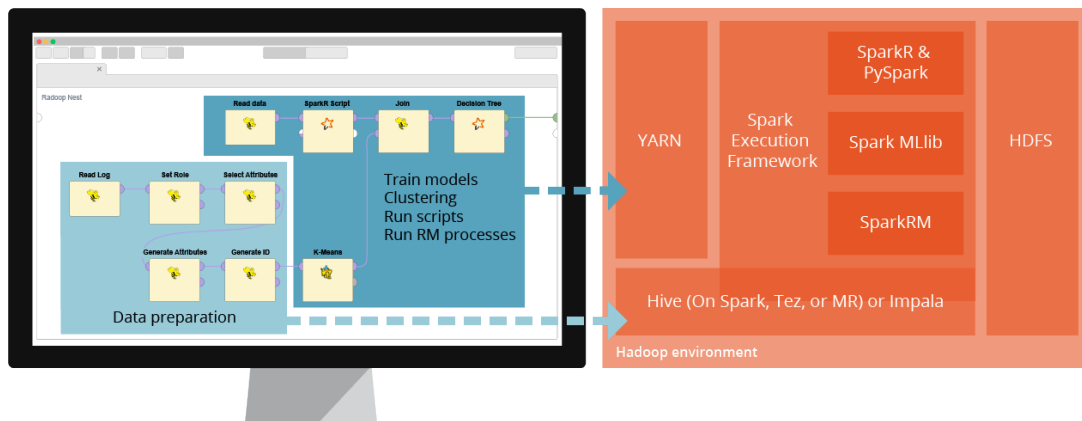
The RapidMiner Platform

RapidMiner eliminates the complexities of doing data science on Hadoop. RapidMiner Studio is a visual workflow designer that empowers data scientists to build and deploy predictive analytics without needing to write code (although users can easily incorporate R, Python and SQL scripts, too, if they desire, including SparkR, PySpark, Pig or HiveQL). RapidMiner Radoop extends RapidMiner Studio, adding built-in intelligence that automatically understands Hadoop complexities and makes them transparent to users.

With RapidMiner Studio, data scientists can visually create predictive analytics workflows, or processes, in a matter of minutes. RapidMiner Radoop then translates these predictive analytics workflows into Hadoop, where the analytics are executed across the entire cluster, taking advantage of the full processing power of Hadoop, and importantly, allowing analysis upon the full breadth and variety of data. This means that users can focus solely on the data science, rather than on programming each step required to execute predictive analytics across all the different technologies of Hadoop.

The Details

RapidMiner Radoop speaks native Hive and Spark (as well as MapReduce, and Pig), automatically translating predictive analytics workflows for native Hadoop execution. Each building block within a RapidMiner Studio workflow executes a specific operation, such as data loading, blending, transformation, machine learning, or scoring. RapidMiner Radoop automatically performs the necessary integration steps as the workflow is executed across Hadoop. It ensures that operations are seamlessly combined, and it performs all the necessary integration steps (e.g., data format conversions) behind the scenes.



For example, for a workflow that loads multiple data sets that reside across the Hadoop cluster, RapidMiner Radoop speaks native Hadoop to transform and combine them and trains a model on that data. It can orchestrate operations of a predictive analytics workflow that spans and uses all the Hadoop technologies mentioned above. That's how RapidMiner Radoop allows you to focus on your data science processes (preparing data, training models, applying, and operationalizing them), instead of on debugging code.

RapidMiner Radoop also enhances internal Hadoop technologies and automatically employs several optimizations in the background. For example, it extends the natural capabilities of Hive by submitting custom UDFs that improve performance or add functionality, i.e. the ability to calculate a pivot table, compute a correlation matrix or perform a principal component analysis. Additionally, it also supports Hive on Spark as a new method for extracting more performance from your Hadoop cluster. When using this configuration, resource management is also automatically optimized by re-using Spark containers and reducing the latency that's inherent in Hadoop's architecture.

RapidMiner Radoop also offers various data import operations and data conversions, which are internally submitted as custom jobs to the YARN ResourceManager of a Hadoop cluster. Spark machine learning libraries (MLlib) is also tightly integrated so that machine learning models are computed on the cluster in a distributed fashion and then returned to RapidMiner to be operationalized in business processes.

RapidMiner Radoop's capabilities don't stop there, as all RapidMiner Studio operators and supported extensions can be executed on the cluster via the RapidMiner SparkRM operator. With SparkRM, you can embed operators or subprocesses in the familiar RapidMiner Studio interface and push them down to the Hadoop cluster for its execution using native Spark.

Managing and configuring RapidMiner Radoop is easy. Simply connect to the cluster manager of your Hadoop distribution (Cloudera Manager, Apache Ambari or MapR) and the import wizard will retrieve all the required parameters and populate the connection quickly and easily.

Another aspect of Hadoop that adds complexity is security. Typically, a secured environment includes firewalls as perimeter security, Kerberos for authentication, Ranger or Sentry for authorization and, in some cases HDFS encryption within the cluster itself. RapidMiner Radoop easily integrates with all these elements providing a transparent user experience while keeping the tight security all companies need these days.

Use Case for Data Science on Hadoop

Let's now discuss a practical example and show the differences between a traditional, programming- based approach and a modern and effective approach using RapidMiner Radoop.

The use case we are going to cover is the prediction of flight delays. For any given flight, we want to predict if the flight is significantly delayed or canceled. Keep in mind that this happens to about 20% of all airline flights and creates a severe economic damage (and upset customers). We will build a supervised classification model based on machine learning which to predict the delay for a flight based on historical flight data and weather information.

This is a well understood dataset and classification problem. And it comes with the advantage that solutions based on Hadoop or Spark are publicly available and well documented. We will compare RapidMiner Radoop to a code-based approaches using Pig, Python, and Scikit-learn on a Hadoop cluster.

Details can be found here:


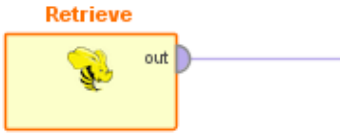
<http://nbviewer.jupyter.org/github/ofermend/IPython-notebooks/blob/master/blog-part-1.ipynb>

Alternatively, we could use Spark, Scala, and MLLib which would lead to a very similar approach

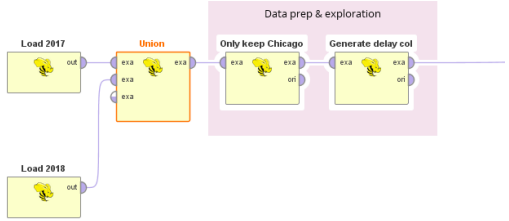
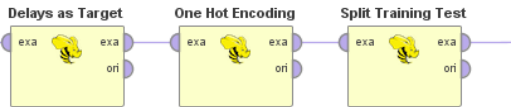
<http://nbviewer.jupyter.org/github/ofermend/IPython-notebooks/blob/master/blog-part-2.ipynb>

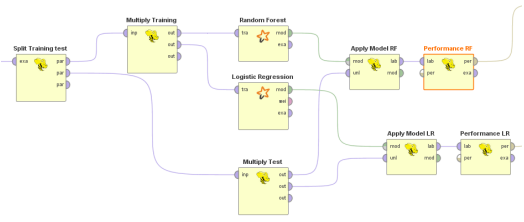
All links and additional information can be found at the end of this section.

To keep this comparison short, we will skip most of the code lines in our discussion. But we will show excerpts together with the total amount of lines of code. And while the latter is generally seen as a poor metric for productivity, it still can give us an idea about how much time it takes to create such a solution for a production environment. Studies have shown that good developers on average produce between 10 and 30 lines of production-ready lines of code – per day ([compare here](#)). This gives us a rough idea about the potential effort of building such a solution.

Goal	Pig + Python + Scikit-learn	RapidMiner Radoop
Preparing the system	<p>Lines of Code: 18 Code Example:</p> <pre>[...] import warnings warnings.filterwarnings('ignore') import sys import random import numpy as np from sklearn import linear_model, cross_validation, metrics, svm [...] import pandas as pd import matplotlib.pyplot as plt %matplotlib inline</pre>	<p>Just add the operator to your workflow and select the right connection:</p>  <p>Besides this, nothing is necessary. The RapidMiner system is immediately prepared for doing data preparation and machine learning.</p>
Read Data	<p>Lines of Code: 19 Code Example:</p> <pre>[...] import pydoop.hdfs as hdfs def read_csv_from_hdfs(path, cols, col_types=None): files = hdfs.ls(path); pieces = [] for f in files: fhandle = hdfs.open(f) pieces.append(pd.read_csv(fhandle, names=cols, dtype=col_types)) fhandle.close() return pd.concat(pieces, ignore_index=True) [...] cols = ['year', 'month', 'day', 'dow', 'DepTime', 'CRSDepTime', 'ArrTime', 'CRSArrTime', 'Carrier', 'FlightNum', 'TailNum', 'ActualElapsedTime', 'CRSElapsedTime', 'AirTime', 'ArrDelay', 'DepDelay', 'Origin', 'Dest', 'Distance', 'TaxiIn', 'TaxiOut', 'Cancelled', 'CancellationCode', 'Diverted', 'CarrierDelay', 'WeatherDelay', 'NasDelay', 'SecurityDelay', 'LateAircraftDelay']; [...]</pre>	<p>Data loading can be done with a single operator which automatically detects the correct column specifications:</p> 

Goal	Pig + Python + Scikit-learn	RapidMiner Radoop
Explore the Data	<p>Lines of Code: 27 Code Example:</p> <pre>df = flt_2007[flt_2007['Origin']=='ORD'].dropna(subset=['DepDelay']) df['DepDelayed'] = df['DepDelay'].apply(lambda x: x>=15) print "total flights: " + str(df.shape[0]) print "total delays: " + str(df['DepDelayed'].sum()) # Select a Pandas dataframe with flight originating from ORD # Compute average number of delayed flights per month grouped = df[['DepDelayed', 'month']].groupby('month').mean() # plot average delays by month grouped.plot(kind='bar') [...]</pre>	<p>We can simply use operators to filter the data and generate a new column indicating if a flight is delayed or not based on the delay time:</p>  <p>All statistics and chart visualizations are then immediately available:</p> 

Goal	Pig + Python + Scikit-learn	RapidMiner Radoop
Prepare the data	<pre> Lines of Code: 73 Code Example: # Python UDFs for our PIG script from datetime import date # get hour-of-day from HHMM field @outputSchema("value: int") def get_hour(val): return int(val.zfill(4)[:2]) # this array defines the dates of holiday in 2007 and 2008 holidays = [date(2007, 1, 1), date(2007, 1, 15), date(2007, 2, 19), date(2007, 5, 28), date(2007, 6, 7), date(2007, 7, 4), \date(2007, 9, 3), date(2007, 10, 8), date(2007, 11, 11), date(2007, 11, 22), date(2007, 12, 25), \ date(2008, 1, 1), date(2008, 1, 21), date(2008, 2, 18), date(2008, 5, 22), date(2008, 5, 26), date(2008, 7, 4), \ date(2008, 9, 1), date(2008, 10, 13), date(2008, 11, 11), date(2008, 11, 27), date(2008, 12, 25) \] # get number of days from nearest holiday @outputSchema("days: int") def days_from_nearest_holiday(year, month, day): d = date(year, month, day) x = [(abs(d-h)).days for h in holidays] return min(x) [...] while True: line = pig_out.readline() if not line: break sys.stdout.write("%s" % line) sys.stdout.flush() </pre>	<p>We can easily re-use the preparation process from above and run this process for the data from 2007 and 2008:</p> 
Prepare Modeling	<pre> Lines of Code: 59 Code Example: # read files cols = ['delay', 'month', 'day', 'dow', 'hour', 'distance', 'carrier', 'dest', 'days_from_holiday'] col_types = {'delay': int, 'month': int, 'day': int, 'dow': int, 'hour': int, 'distance': int, 'carrier': str, 'dest': str, 'days_from_holiday': int} data_2007 = read_csv_from_hdfs('airline/fm/ord_2007_1', cols, col_types) data_2008 = read_csv_from_hdfs('airline/fm/ord_2008_1', cols, col_types) # Create training set and test set cols = ['month', 'day', 'dow', 'hour', 'distance', 'days_from_holiday'] train_y = data_2007['delay'] >= 15 train_x = data_2007[cols] test_y = data_2008['delay'] >= 15 test_x = data_2008[cols] [...] from sklearn.preprocessing import OneHotEncoder [...] </pre>	<p>We can now make the final preparations for using machine learning models. First, we define the target column. This is the column which should be predicted, in our case "delays". Then we improve the data set with a technique called One Hot Encoding. And finally, we split the data into a training and a test data set:</p>  <p>Prepare data for modeling: define target, hot once encoding and splitting in training and test set</p>

Goal	Pig + Python + Scikit-learn	RapidMiner Radoop
Build Logistic and Random Forrest and Compare Accuracy	<pre> Lines of Code: 35 Code Example: # Create logistic regression model with L2 regularization clf_lr = linear_model.LogisticRegression(penalty='l2', class_weight='auto') clf_lr.fit(train_x, train_y) # Predict output labels on test set pr = clf_lr.predict(test_x) # display evaluation metrics cm = confusion_matrix(test_y, pr) print("Confusion matrix") print(pd.DataFrame(cm)) report_lr = precision_recall_fscore_support(list(test_y), list(pr), average='micro') print "\nprecision = %0.2f, recall = %0.2f, F1 = %0.2f, accuracy = %0.2f\n" % \ (report_lr[0], report_lr[1], report_lr[2], accuracy_score(list(test_y), list(pr))) # Create Random Forest classifier with 50 trees [...] </pre>	<p>As the final step, we can now train two different machine learning models (Logistic Regression and Random Forest) on the prepared data and test those models on unseen data points:</p>  <p>The diagram illustrates a workflow starting with 'Split Training test' data. This data is used for 'Multiple Training' of two models: 'Random Forest' and 'Logistic Regression'. The trained models are then used to 'Apply Model RF' and 'Apply Model LR' respectively. Finally, the results are compared using 'Performance RF' and 'Performance LR' operators.</p>

	<p>Total Lines of Code: 231</p> <p>At an average of 20 lines of production-ready code (tested and well-documented), the creation of this solution will take a minimum of 11 days.</p>	<p>Total Number of Operators: 21</p> <p>Thanks to the visual workflow approach the complete solution is production-ready within a few hours.</p>
--	--	---

We can conclude that on average the code-based solution will require at least 11 development days which can only be delivered by highly specialized coders who are familiar with the underlying infrastructures and needed languages such as Python, Pig, or Scala.

In contrast to this, a data scientist using RapidMiner Radoop only needs to drag 19 self-documented operators into a visual workflow. This does not require any specialized skills and can be achieved within a few hours. Since the building blocks are standardized, this will lead to a manageable and well-tested solution in a much shorter time. Consequently, your data science team will be much more productive and will be able to deliver more robust data science solutions on big data infrastructures in a fraction of time.

Links:

Pig, Python, Scikit-learn:

<https://hortonworks.com/blog/data-science-apacheh-hadoop-predicting-airline-delays/>

Jupyter Notebook with complete code:

<http://nbviewer.jupyter.org/github/ofermend/IPython-notebooks/blob/master/blog-part-1.ipynb>

Spark, Scala, MLlib:

<https://hortonworks.com/blog/data-science-hadoop-spark-scala-part-2/>

Jupyter Notebook with complete code:

<http://nbviewer.jupyter.org/github/ofermend/IPython-notebooks/blob/master/blog-part-2.ipynb>

Conclusion

RapidMiner makes it fast and simple for data scientists to create and execute predictive analytics on Hadoop, empowering organizations to quickly unlock the immense value buried in their Hadoop cluster.

Users build predictive analytics workflows using the visual workflow designer of RapidMiner Studio, eliminating the need to write code while still allowing data scientists to embed their favorite R, Python, and SQL scripts within a predictive analytics workflow. With RapidMiner Radoop data scientists are now able to focus their full attention on predictive analytics, rather than also having to solve the technical challenges of both coding and integrating Hadoop technologies. And the RapidMiner workflows can be reused for non-Hadoop use cases, allowing organizations to standardize on a single platform for all their data science projects.

As a result, organizations dramatically reduce the complexity and risk of Hadoop-based analytics initiatives, and greatly improve the probability of delivering business value from their Big Data projects.



10 Milk Street, 11th Floor
Boston, MA 02108

RapidMiner builds a software platform for data science teams that unites data prep, machine learning, and predictive model deployment. RapidMiner products are used by over 300,000 data scientists in over 150 countries.

For more information, visit www.rapidminer.com