# Machine Learning with Spark MLIB and PySpark

## Step 1: Log in to Worker Nodes and Install the Library

1. Log in to Worker Node 1:  
   docker-compose exec worker1 bash
2. Install the Library using pip3:  
   pip3 install numpy happybase
3. Exit Worker Node 1:  
   exit
4. Log in to Worker Node 2:  
   docker-compose exec worker2 bash
5. Install the Library using pip3:  
   pip3 install numpy happybase
6. Exit Worker Node 2:  
   exit

## Step 2: Log in to the Master Node and Install the Library

1. Log in to the Master Node:  
   docker-compose exec master bash
2. Install the Library using pip3:  
   pip3 install numpy happybase

## Step 3: Load grades.csv into HDFS

**This dataset is only provided as an example. You are not allowed to use it for your Final Project. Submitting with this dataset will result in an automatic grade of zero. For your project, you must select your own dataset and use NiFi to load it into HDFS.**

**Use a small dataset. The objective is to build a pipeline with the tools not to test scale on your very small virtual machine.**

hdfs dfs -put /data/grades.csv /

## Step 3: Load grades.csv into Hive

1. Open Hive Shell:

hive

1. Create the Hive Table:

CREATE TABLE gradesml (  
 last\_name STRING,  
 first\_name STRING,  
 ssn STRING,  
 test1 INT,  
 test2 INT,  
 test3 INT,  
 test4 INT,  
 final\_score INT,  
 grade STRING  
)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY ','  
STORED AS TEXTFILE;

1. Load Data from CSV into the Hive Table:

LOAD DATA INPATH '/grades.csv' INTO TABLE gradesml;

1. Exit hive shell

exit;

## Step 4: Create an HBase table to load your metrics from Spark ML

1. Open the HBase shell:  
     
   hbase shell
2. Create a table in HBase named my\_table with a column family cf:  
     
   create 'my\_table', 'cf'
3. Exit the hbase shell

exit

## Step 4: Start the HBase Thrift Server

1. Start the HBase Thrift server to allow Python to connect via happybase. This command will be run in the master docker container.  
     
   nohup hbase thrift start &  
     
   The & runs the Thrift server in the background.

## Step 5: Use Spark to Run Machine Learning (MLlib)

1. Use VSCode or your IDE and github to locally write your python code and push to the master docker container. This was setup in week 4.
2. Here’s a PySpark example that uses test scores to predict the final score with MLlib (Linear Regression) and writes the results to HBase. **You’ll need to adapt this code to your own environment by pointing it to your dataset, your Hive table, and your HBase table**. I called this file sparkml.py

from pyspark.sql import SparkSession

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.regression import LinearRegression

import happybase

# Step 1: Create a Spark session

spark = SparkSession.builder.appName("MLlib GradesML Prediction").enableHiveSupport().getOrCreate()

# Step 2: Load the data from the Hive table 'gradesml' into a Spark DataFrame

grades\_df = spark.sql("SELECT test1, test2, test3, test4, final\_score FROM gradesml")

# Step 3: Handle null values by either dropping or filling them

grades\_df = grades\_df.na.drop() # Drop rows with null values

# Step 4: Prepare the data for MLlib by assembling features into a vector

assembler = VectorAssembler(

inputCols=["test1", "test2", "test3", "test4"],

outputCol="features",

handleInvalid="skip" # Skip rows with null values

)

assembled\_df = assembler.transform(grades\_df).select("features", "final\_score")

# Step 5: Split the data into training and testing sets

train\_data, test\_data = assembled\_df.randomSplit([0.7, 0.3])

# Step 6: Initialize and train a Linear Regression model

lr = LinearRegression(labelCol="final\_score")

lr\_model = lr.fit(train\_data)

# Step 7: Evaluate the model on the test data

test\_results = lr\_model.evaluate(test\_data)

# Step 8: Print the model performance metrics

print(f"RMSE: {test\_results.rootMeanSquaredError}")

print(f"R^2: {test\_results.r2}")

# ---- Write metrics to HBase with happybase (using the provided pattern) ----

# Example data (row\_key, column\_family:column, value) populated with the metrics

data = [

('metrics1', 'cf:rmse', str(test\_results.rootMeanSquaredError)),

('metrics1', 'cf:r2', str(test\_results.r2)),

]

# Function to write data to HBase inside each partition

def write\_to\_hbase\_partition(partition):

connection = happybase.Connection('master')

connection.open()

table = connection.table('my\_table') # Update table name

for row in partition:

row\_key, column, value = row

table.put(row\_key, {column: value})

connection.close()

# Parallelize data and apply the function with foreachPartition

rdd = spark.sparkContext.parallelize(data)

rdd.foreachPartition(write\_to\_hbase\_partition)

# Step 9: Stop the Spark session

spark.stop()

1. Submit your job like this.

spark-submit \

--master yarn \

--deploy-mode cluster \

--name GradesML\_to\_HBase \

sparkml.py

## Step 6: Check the Application logs

1. Find application id

When the spark job runs, you’ll see an output like this:

client token: N/A

diagnostics: N/A

ApplicationMaster host: N/A

ApplicationMaster RPC port: -1

queue: root.root

start time: 1758161692199

final status: UNDEFINED

tracking URL: http://master:8088/proxy/**application\_1758160134642\_0005**/

user: root

1. After the job has finished, get the logs (this will be helpful for debugging your code)

yarn logs -applicationId <APP\_ID>

in my case its

yarn logs -applicationId **application\_1758160134642\_0005**

## Step 7: Verify Data in HBase

1. After the script runs, you can verify the data written to HBase by using the HBase shell:  
     
   hbase shell  
     
   scan 'my\_table'