USING SCALA WITH APACHE IGNITE MACHINE LEARNING

This article is an expansion of How to Use Apache Ignite for Machine Learning. In this article, we'll show how to code the same program, but using Scala instead. We focus on how to get Ignite Machine Learning in Scala.

The code for this example is here.

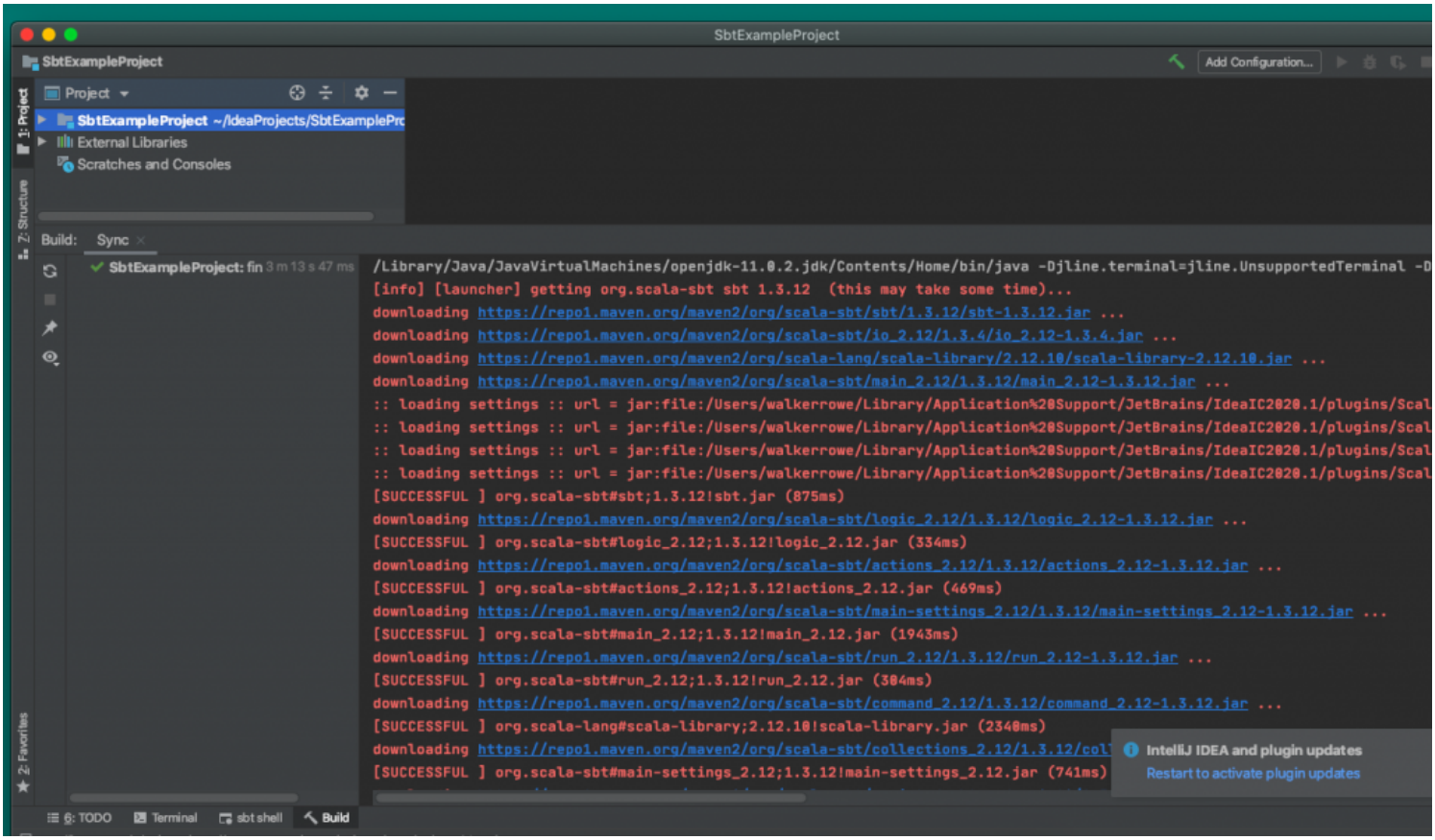
Use Intellij IDEA to convert code

By far the easiest way to convert the previous Java code to Scala is to use the Intellij IDEA, a Java-based IDE (integrated development environment). This lets you literally copy, line by line, the Java code into Intellij and it will convert the Java code to Scala for you.

You first install the Scala plugin to Intellij. If you've worked with Scala before, you know that it will take a while to set up SBT (Scala build tool) the first time.

Working in Scala

Next, create a Scala project.



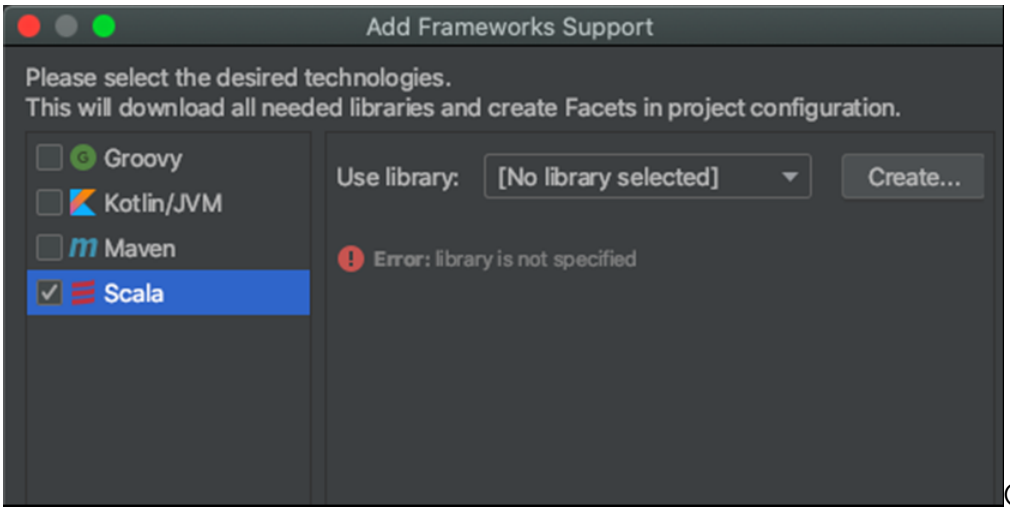
Add the Ignite dependencies to **build.sbt**:

```
{
libraryDependencies ++= Seq(
  "org.apache.ignite" % "ignite-core" % "2.8.1",
"org.apache.ignite" % "ignite-ml" % "2.8.1" )
}
```

Download the Scala SDK. For example:

```
wget https://downloads.lightbend.com/scala/2.13.2/scala-2.13.2.tgz
```

To create this simple linear regression example, right click on project and select Add Framework Support and the lib folder in the SDK you just downloaded.

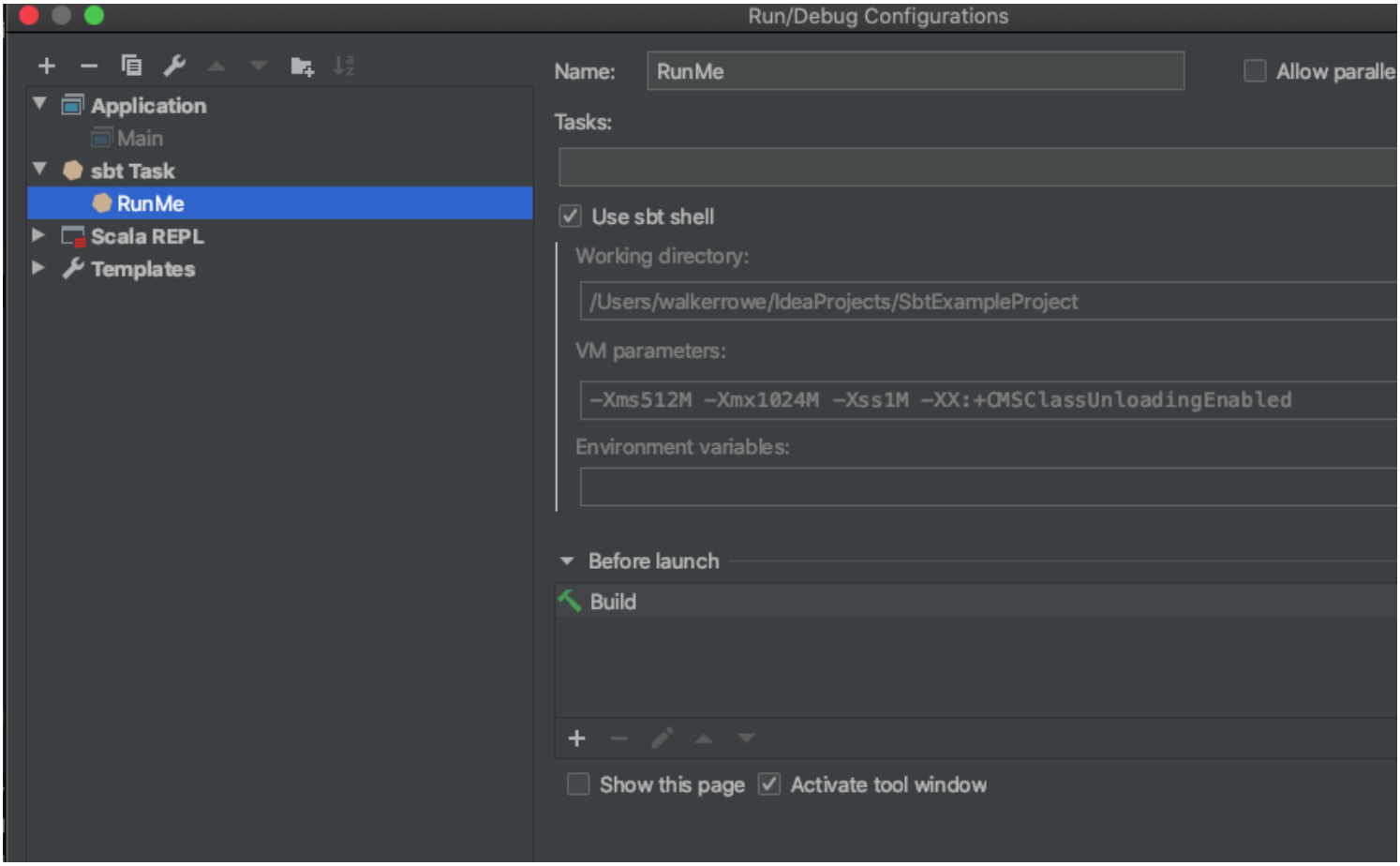


Create a Scala class file, but change it to an **Object**, so that you can run it by just clicking the green arrow.

```
package com.bmc.ignite

import org.apache.ignite.Ignite
import org.apache.ignite.Ignition
import org.apache.ignite.configuration.IgniteConfiguration
import java.io.IOException
import java.util
import java.util.UUID
import org.apache.ignite.Ignite
import org.apache.ignite.IgniteCache
import org.apache.ignite.Ignition
import org.apache.ignite.cache.affinity.rendezvous.RendezvousAffinityFunction
import org.apache.ignite.configuration.CacheConfiguration
import org.apache.ignite.configuration.IgniteConfiguration
import org.apache.ignite.ml.dataset.feature.extractor.Vectorizer
import org.apache.ignite.ml.dataset.feature.extractor.impl.DummyVectorizer
import org.apache.ignite.ml.math.primitives.vector.Vector
import org.apache.ignite.ml.math.primitives.vector.VectorUtils
import org.apache.ignite.ml.regressions.linear.LinearRegressionLSQRTrainer
import org.apache.ignite.ml.regressions.linear.LinearRegressionModel
import org.apache.ignite.ml.selection.scoring.evaluator.Evaluator
import org.apache.ignite.ml.selection.scoring.metric.MetricName



object Main extends App {


val PersistencePath = "/Users/walkerrowe/Downloads/ignite"
val WalPath = "/Users/walkerrowe/Downloads/wal"

val config = new IgniteConfiguration()


val ignite = Ignition.start(config)

val dataCache = getCache(ignite)


dataCache.put(1, VectorUtils.of(1, 1.8))
dataCache.put(2, VectorUtils.of(2, 4.3))
dataCache.put(3, VectorUtils.of(3, 6.2))
dataCache.put(4, VectorUtils.of(4, 5))
dataCache.put(5, VectorUtils.of(5, 11))
dataCache.put(6, VectorUtils.of(6, 11))
dataCache.put(7, VectorUtils.of(7, 15))
```

```
println("data created")


val vectorizer = new
DummyVectorizer().labeled(Vectorizer.LabelCoordinate.FIRST)

val trainer = new LinearRegressionLSQRTrainer

val mdl = trainer.fit(ignite, dataCache, vectorizer)

val rmse = Evaluator.evaluate(dataCache, mdl, new
DummyVectorizer().labeled(Vectorizer.LabelCoordinate.FIRST), MetricName.RMSE)

print("rmse" + rmse)

System.out.println("intercept = " + mdl.getIntercept)


System.out.println("Weights = ")

val weights = mdl.getWeights
val w = weights.asArray
for (v <- w) {
System.out.println(v)
}




private def getCache(ignite: Ignite) = {
val cacheConfiguration = new CacheConfiguration
cacheConfiguration.setName("ML_EXAMPLE_" + UUID.randomUUID)
cacheConfiguration.setAffinity(new RendezvousAffinityFunction(false, 10))
ignite.createCache(cacheConfiguration)
}
}
```

You also have to create an SBT build task in Intellij:



Now click the green arrow next to **object Main Extends App {**

This will start Apache Ignite and then run linear regression across the six data variables that we mocked. The program will output:

- Mean squared error (MSE)
- Intercept
- Coefficient (weights)

```
mse 0.5963051208050183
intercept = 0.5762626813570405
Weights = 0.4413657685174977
```