Rajiv Shah's Projects Blog

Taking an H2O Model to Production

22 Aug 2016

One of the best feelings as a Data Scientist is when the model you have poured your heart and soul into, moves into production. Your model is now *grown-up* and you get to watch it mature.

This post shows how to take a H2O model and move it into a production environment. In this post, we will develop a simple H2O based predictive model, convert it into a Plain Old Java Object (POJO), compile it along with other Java packages, and package the compiled class files into a deployable JAR file so that it can readily be deployed onto any Java based application servers. This model will accept the input data set in the form of CSV file and return the predicted output in CSV format.

H2O is one of my favorite tools for building models because it is well designed from an algorithm perspective, easy to use, and can scale to larger datasets. However, H2O's documentation, though voluminous, doesn't have clear instructions for moving a POJO model into production. This post will discuss this approach in greater detail besides providing code for how to do this. (H2O does have a post on doing real time predictions with storm). Special thanks to Socrates Krishnamurthy who co-wrote this post with me.

Building H2O Model

As a starting point, lets use our favorite ice cream dataset to create a toy model in H2O:

```
library(h2o)
library(Ecdat)
```

```
  data(Icecream)
  h2o.init()
  train.h2o <- as.h2o(Icecream)
  rf <- h2o.randomForest(x=2:4, y=1, ntrees=2, tr
aining_frame=train.h2o)
```

Once you have developed your model in H2O, then the next step is downloading the POJO:

```
h2o.download_pojo(rf, getjar=TRUE, path="~/Code/h
2o-3.9.1.3459/test/")
# you must give a path to download a file
```

This will save two files, a H2O jar file about the model and an actual model file (that begins with `DRF` and ends with `.java`). Go ahead and open the model file in a text editor if you want to have a look at it.

Compiling the H2O Model

The next step is to compile and run the model (say, the downloaded model name is `DRF_model_R_1470416938086_15.java`), then type:

```
> javac -cp h2o-genmodel.jar -J-Xmx2g DRF_model_R
_1470416938086_15.java
```

This creates a bunch of java class files.

Scoring the Input Data

The final step is scoring some input data. Prior to running the model, it is necessary to have files created for the input and output. For the input, the default setting is to read the first row as a header. The assumption is that the csv is well formed (this approach is not using the H2O parser). Once that is done, run:

```
> java -cp .:h2o-genmodel.jar hex.genmodel.tools.
PredictCsv --header --model DRF_model_R_147041693
8086_15 --input input.csv --output output.csv
```

If you open the `output.csv` file, it can be noticed that the predicted values are in Hexadecimal and not in Numeric format. For example, the output will be something like this:

```
0x1.a24dd2p-2
```

Fixing the Hexadecimal Issue

The model is now predicting, but the predictions are in the wrong format. Yikes! To fix this issue requires some hacking of the java code. The rest of this post will show you how to hack the java code in PredictCsv, which can fix this issue and other unexpected issues with PredictCsv (for example, if your input comes tab separated).

If we take a deeper look at the PredictCsv java file located in the h2o github, the `myDoubleToString` method returns Hexadecimal string. But the challenge is this method being `static` in nature, cannot be overridden in a subclass or cannot be updated directly since it was provided by H2O jar file, to return regular numeric value in String format.

This can be fixed by creating a new java file (say, `NewPredictCsv.java`) by copying the entire content of `PredictCsv.java` from the above location and saving it locally. You then need to:

- comment out the first line, so it should be `//package hex.genmodel.tools;`
- change the name of the class name (~line 20) to read: `public class NewPredictCsv {`
- correct the hexadecimal issue by changing the return statement of `myDoubleToString` method to `.toString()` in lieu of `.toHexString()` (~line 131).

After creating `NewPredictCsv.java` , compile it using the following command:

```
> javac -cp h2o-genmodel.jar -J-Xmx2g NewPredictC
sv.java DRF_model_R_1470416938086_15.java
```

Run the compiled file by providing input and output CSV files using the following command (Ensure that the `input.csv` file is in the current folder where you will run this):

```
> java -cp .:h2o-genmodel.jar NewPredictCsv --hea
der --model DRF_model_R_1470416938086_15 --input
 input.csv --output output.csv
```

If you open the `output.csv` file now, it will be in the proper numeric format as follows:

```
0.40849998593330383
```

Deploying the Solution into Production:

At this point, we have a workable flow for using our model to score new data. But we can clean up the code to make it a little friendlier for our data engineers. First, create a jar file out of the class files created in previous steps. To do that, issue the following command:

```
> jar cf my-RF-model.jar *.class
```

This will place all the class files and our NewPredictCsv inside the jar. This is helpful when we have a model with say 500 trees. Now all we need is three files to run our scorer. So copy the above two jar files along with `input.csv` file in any folder/directory from where the program has to be executed. After copying, the folder should contain following files:

```
> my-RF-model.jar
> h2o-genmodel.jar
> input.csv
```

The above `input.csv` file contains the dataset for which the dependent variable has to be predicted. To compute/ predict the values, run the `java` command as below:

```
> java -cp .:my-RF-model.jar:h2o-genmodel.jar New
PredictCsv --header --model DRF_model_R_147041693
8086_15 --input input.csv --output output.csv
```

Note:

Replace `:` with `;` in above commands if you are working in Windows (yuck).

Related Posts

Using Unlabeled Data to Label Data 16 Jan 2018

Using Google's Quickdraw to create an MNIST style dataset! 14 Jul 2017

Deep Learning with R 04 Jun 2017

Comments **Community** Ava

Recommend 2 **Share** **Sort by Best**

Rajiv Shah's Projects requires you to verify your email address before posting. Send verification email to robert.l.mcpherson@gmail.com ×

Join the discussion…

RickBlaine • 6 months ago

How do I run the model I created if input.csv is in HDFS?

Reply • Share ›

Nitisha Nupur • 8 months ago

[illegible]

Hi Rajiv, the article is extremely helpful. Could you also provide a link to the PredictCsv.java file that you are using? I am getting errors when I use the latest file available.
TIA

^ | v • Reply • Share ›

Rajiv Shah Mod ➦ Nitisha Nupur
• 8 months ago

You can look at the history the PredictCSV.java file and find the one that I used at: https://github.com/h2oai/h2... But that isn't very satisfying, maybe if you try the H2O Stackoverflow, someone can help. I think this is a use case (scoring in java), that there needs to be good documentation on.

^ | v • Reply • Share ›

Ming Cong • 9 months ago

Hi Rajiv, I really love this article. Thanks!

I have a question. When I tried to predict my own test csv file (which has many rows), all the values in the output file are all the same. Why would this happen? Does the "PredictCsv.java" only do one prediction at a time or I did something wrong? Thanks you!

^ | v • Reply • Share ›

Edileusa Estefani ➦ Ming Cong
• 6 months ago

Hi Ming Cong I'm Edileusa Prado and I have the same problem than you. Could you explain me how you to do to solve this problem?
Thanks so much Edileusa.

^ | v • Reply • Share ›

Rajiv Shah Mod ➦ Ming Cong
• 9 months ago

I need to update this post with the hint that the input file should use the column names that are in the POJO model. If you don't do this, it is just treating your values as NA and that is why every row is getting the same prediction. For example, input.csv should look like this:
income,price,temp
1,1,100
202,2,68

202,2,08
3,3,3

^ | v • Reply • Share ›

Edileusa Estefani → Rajiv Shah
• 6 months ago

I'm with the same problem that Ming Cong the output file generated only one values how result my prediction. Do you have a new release of this file Rajiv? Couldo you pass me, please? Thanks a lot Edileusa Prado

^ | v • Reply • Share ›

Ming Cong → Rajiv Shah
• 9 months ago

I solved that problem. Thank you Rajiv.

Another problem is... when I tried to hack the "PredictCsv.java" file, I did exactly the three steps you suggested. After that, I put the "NewPredictCsv.java", "my-model.java" and "h2o-genmodel.jar" together and ran javac -cp h2o-genmodel.jar -J-Xmx2g NewPredictCsv.java my-model.java. Then I got some errors, which are about the import sumbol cannot be found.

NewPredictCsv.java:5: error: cannot find symbol

see more

^ | v • Reply • Share ›

Rajiv Shah Mod → Ming Cong • 9 months ago

Looking at your errors, you are using a newer version of the PredictCSV than I did. I know they have updated it to include support for the MOJO (and numeric support as well). Try asking on the h2o communities site. They still need to show a nice example like mine where you want to predict a csv while staying in java.

Reply • Share ›

Tammy Ernst Rajiv Shah
• 6 months ago

The latest Predict.csv has a