Building a Smarter WebApp with a Generated POJO Model

H2O.ai

Tom Kraljevic

September 17, 2015

Target Audience

- JavaScript WebApp developer who wants to incorporate machine learning
- More code, less PowerPoint

Building a Smarter Application

- Q: What is a smarter application?
- A: An app that learns from data

  [From rules-based to model-based]

Software Pieces

- Front-end
  - Web browser
  - JavaScript application (run in the browser)
- Back-end
  - Jetty servlet container
  - H2O-generated model POJO (hosted by servlet container)
- Out-of-band
  - H2O

Overview

Step 1: Picking the question you want your model to answer

Step 2: Using your data to build a model

Step 3: Exporting the generated model as a Java POJO

Step 4: Compiling the model

Step 5: Hosting the model in a servlet container

Step 6: Running the JavaScript app in a browser

Step 7: Using a REST API to make predictions

Step 8: Incorporating the prediction into your application

Resources

- This meetup
  - https://github.com/h2oai/h2o-meetups/tree/master/2015_09_17_PojoWebApp
- H2O-3 Slater Release
  - http://h2o-release.s3.amazonaws.com/h2o/rel-slater/1/index.html
- Generated POJO Model Javadoc
  - http://h2o-release.s3.amazonaws.com/h2o/rel-slater/1/docs-website/h2o-genmodel/javadoc/index.html
- POJO REST API Example
  - https://github.com/h2oai/h2o-world-2015-training/tree/master/tutorials/pojo_webapp

Demonstration

Next Steps for a Real Use Case

- Scoring (judging how good the predictions really are)
  - Need to get the correct answers from somewhere
- Storing predictions (and the correct answers)
  - Often Hadoop. Seems easy, but this can be real work to organize
- Model update frequency
  - From the stored data above
  - Need depends on the problem
  - Hourly, daily, monthly?
  - Cost of training the model becomes a factor
- Hot swapping the model
  - Separating front-end and back-end makes this easier
  - Java reflection for in-process hot-swap
  - Load balancer for servlet container hot-swap