

BUILDING A SMARTER APPLICATION

Tom Kraljevic

tomk@h2o.ai

H2O World 2015

Nov. 9, 2015

(+ help from Amy and Prithvi)



BUILDING A SMARTER APPLICATION

Q: What is a Smarter Application?

A: An application that learns from data
[From rules-based to model-based]

Topics:

- Combining applications with models
- Deploying models into production

Target audience:

- *Developers* adding Machine Learning to apps
- *Data Scientists/DevOps* putting models into production

RESOURCES ON THE WEB

- Slides and material for this talk
 - GitHub h2oai/h2o-world-2015-training
 - tutorials/building-a-smarter-application
- Source code
 - GitHub h2oai/app-consumer-loan
- H2O Tibshirani release (on USB)
 - <http://h2o.ai/download>
- Generated POJO model Javadoc
 - <http://h2o-release.s3.amazonaws.com/h2o/releases/tibshirani/2/docs-website/h2o-genmodel/javadoc/index.html>

A CONCRETE USE CASE

- We're building a consumer loan app
- The end-user is applying for a loan
- Imagine the website is a lender
- Should a loan be offered?
- Two predictive models
 - Is the loan predicted to be bad (yes/no)
 - If no, what is the interest rate to be offered?

STEPS TO BUILDING A SMARTER APP

- Step 1: Picking the question your model will 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

THE DATA

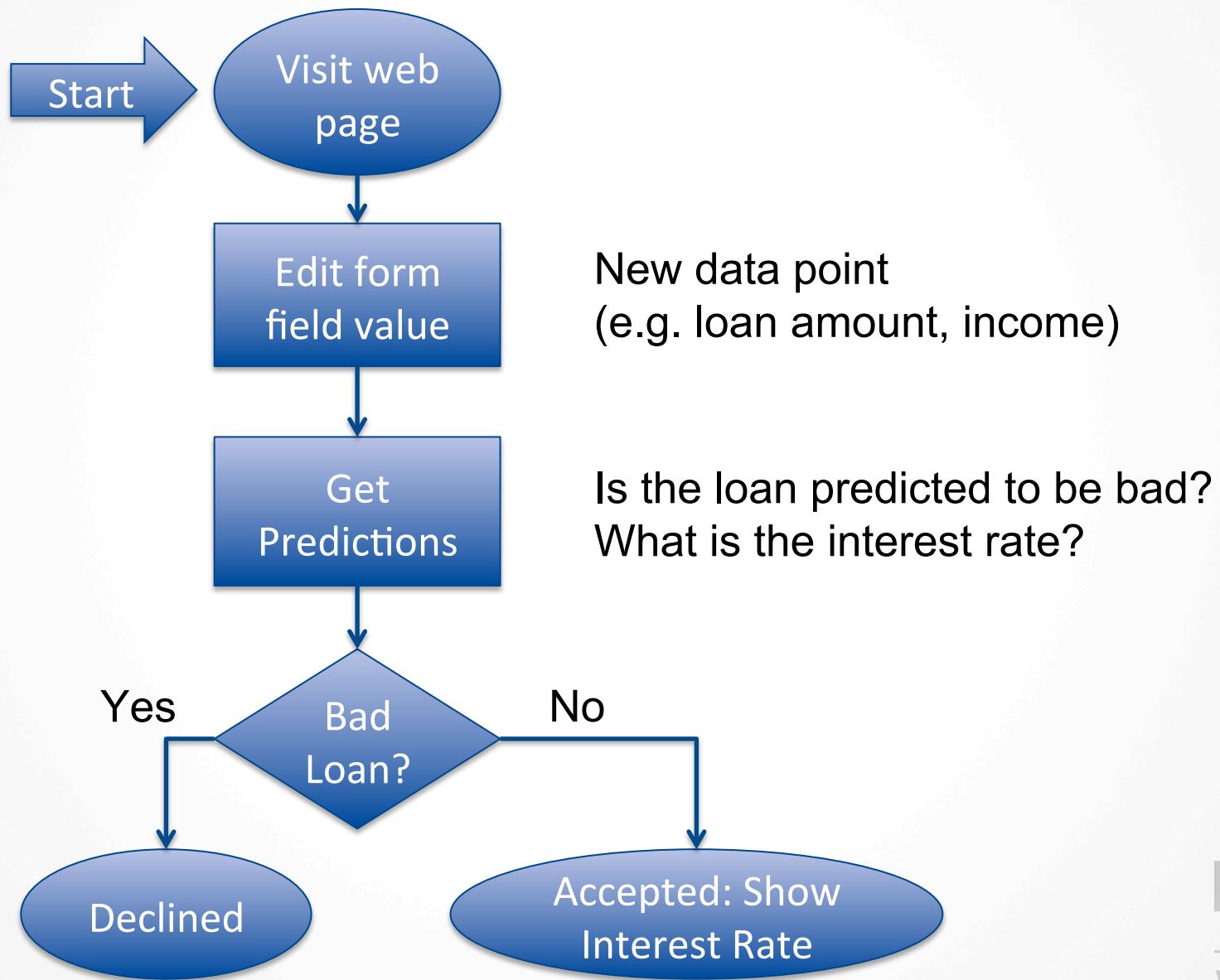
- Lending club loans from 2007 to June 2015
- Only loans that have a known good or bad outcome are used to build the model
- 163,987 rows
- 15 columns

DATA DICTIONARY

Predictor Variable	Description	Units
loan_amnt	Requested loan amount	US dollars
term	Loan term length	months
emp_length	Employment length	years
home_ownership	Housing status	categorical
annual_inc	Annual income	US dollars
verification_status	Income verification status	categorical
purpose	Purpose for the loan	categorical
addr_state	State of residence	categorical
dti	Debt to income ratio	%
delinq_2yrs	Number of delinquencies in the past 2 years	integer
revol_util	Revolving credit line utilized	%
total_acc	Total accounts (number of credit lines)	integer
longest_credit_length	Age of oldest active account	years

Response Variable	Description	Model Category
bad_loan	Is this loan likely to be bad?	Binomial classification
int_rate	What should the interest rate be?	Regression

WORKFLOW FOR THIS APP



APP ARCHITECTURE DIAGRAM

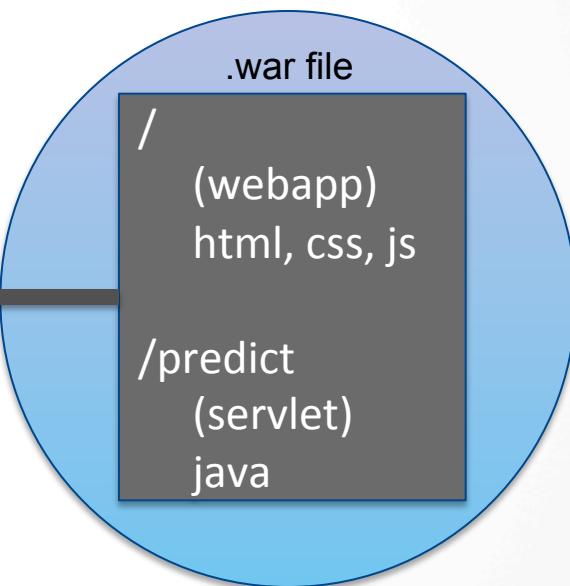
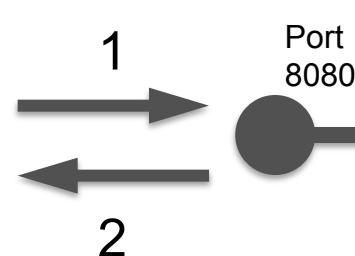
Front-end

Web browser



Back-end

Jetty servlet container



1. HTTP GET with query parameters (loan_amt, annual_inc, etc.)
2. JSON response with predictions

MODEL INFORMATION

Bad Loan Model

Algorithm: GBM

Model category: Binary
Classification

ntrees: 100

max_depth: 5

learn_rate: 0.05

AUC on valid: .685

max F1: 0.202

Interest Rate Model

Algorithm: GBM

Model category: Regression

ntrees: 100

max_depth: 5

learn_rate: 0.05

MSE: 11.1

R2: 0.424

SOFTWARE PIECES

- Offline
 - Gradle (build)
 - R + H2O (model building)
- Front-end
 - Web browser
 - JavaScript application (run in the browser)
- Back-end
 - Jetty servlet container
 - H2O-generated model POJO (hosted by servlet container)

WHAT YOU NEED FOR HANDS-ON



I have R
installed



I have H2O
Installed

- R and the H2O R package (from USB)
- gradle (from USB)
- app-consumer-loan (from USB)
 - Code and data is self-contained in this directory

HANDS-ON DEMONSTRATION

If you are already running H2O on your laptop, please stop it so the gradle script runs properly!

STEP 1: Compile and run (From the command line)

```
C:> cd path\to\H2OWorld2015\USB\image  
C:> cd app-consumer-loan  
C:> ..\gradle-2.8\bin\gradle jettyRunWar
```

STEP 2: Use the app (In a web browser)

<http://localhost:8080>

(Future) STEP 3: Rerun without rebuilding the models or recompiling

```
C:> ..\gradle-2.8\bin\gradle jettyRunWar -x war
```



KEY FILES

- Offline
 - `build.gradle`
 - `data/loan.csv`
 - `script.R`
- Front-end
 - `src/main/webapp/index.html`
 - `src/main/webapp/app.js`
- Back-end
 - `src/main/java/org/gradle/PredictServlet.java`
 - `lib/h2o-genmodel.jar` (*downloaded*)
 - `src/main/java/org/gradle/BadLoanModel.java` (*generated*)
 - `src/main/java/org/gradle/InterestRateModel.java` (*generated*)

POST-DEMO POINTERS

- POJO Javadoc
 - <http://h2o-release.s3.amazonaws.com/h2o/releases/tibshirani/2/docs-website/h2o-genmodel/javadoc/index.html>

NEXT STEPS: CLOSING THE FEEDBACK LOOP

- Scoring
 - Judging how good the predictions really are
 - Need to get the correct answers from somewhere
- Storing predictions (and the correct answers)
 - Often Hadoop
 - This can be a lot of work to organize

NEXT STEPS: RETRAINING AND DEPLOYING

- Model update frequency
 - Need depends on the use case
 - Hourly, daily, monthly?
 - Time cost of training the model is 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

RELATED EXAMPLES

- H2O Generated Model POJO in a Storm bolt
 - GitHub: [h2oai/h2o-world-2015-training](#)
 - [tutorials/streaming/storm](#)
- H2O Generated Model POJO in Spark Streaming
 - GitHub: [h2oai/sparkling-water](#)
 - [examples/src/main/scala/org/apache/spark/examples/h2o/CraigslistJobTitlesStreamingApp.scala](#)

BONUS APP

- “Ask Craig” app
- <http://classify.h2o.ai>
- Uses Spark word2vec, H2O gbm
- Multinomial classification problem
 - Given the words for a new job posting, figure out the right category for the job
- See H2O.ai blog about this Sparkling Water app
 - Do a web search for “h2o ask craig”
- GitHub: h2oai/app-ask-craig

Q & A

- Thanks for attending!
- Send follow up questions to:

Tom Kraljevic
tomk@h2o.ai