

# Building Custom Machine Learning Algorithms with Apache SystemML

Fred Reiss

Chief Architect, IBM Spark Technology Center

Member, IBM Academy of Technology



**SPARK SUMMIT 2016**  
DATA SCIENCE AND ENGINEERING AT SCALE  
JUNE 6-8, 2016 SAN FRANCISCO

# Roadmap

- What is Apache SystemML?
- Demo!
- How to get SystemML



# What is Apache SystemML?



**SPARK SUMMIT 2016**  
DATA SCIENCE AND ENGINEERING AT SCALE  
JUNE 6-8, 2016 SAN FRANCISCO

# Origins of the SystemML Project

You are  
here.

2015

2016

2011

2012

2013

2014



SPARK SUMMIT 2016

**2007-2008:** Multiple projects at IBM Research – Almaden involving machine learning on Hadoop.

**2009:** We form a **dedicated team** for scalable ML

**2009-2010:** Through engagements with customers, we observe how data scientists create **ML solutions**.

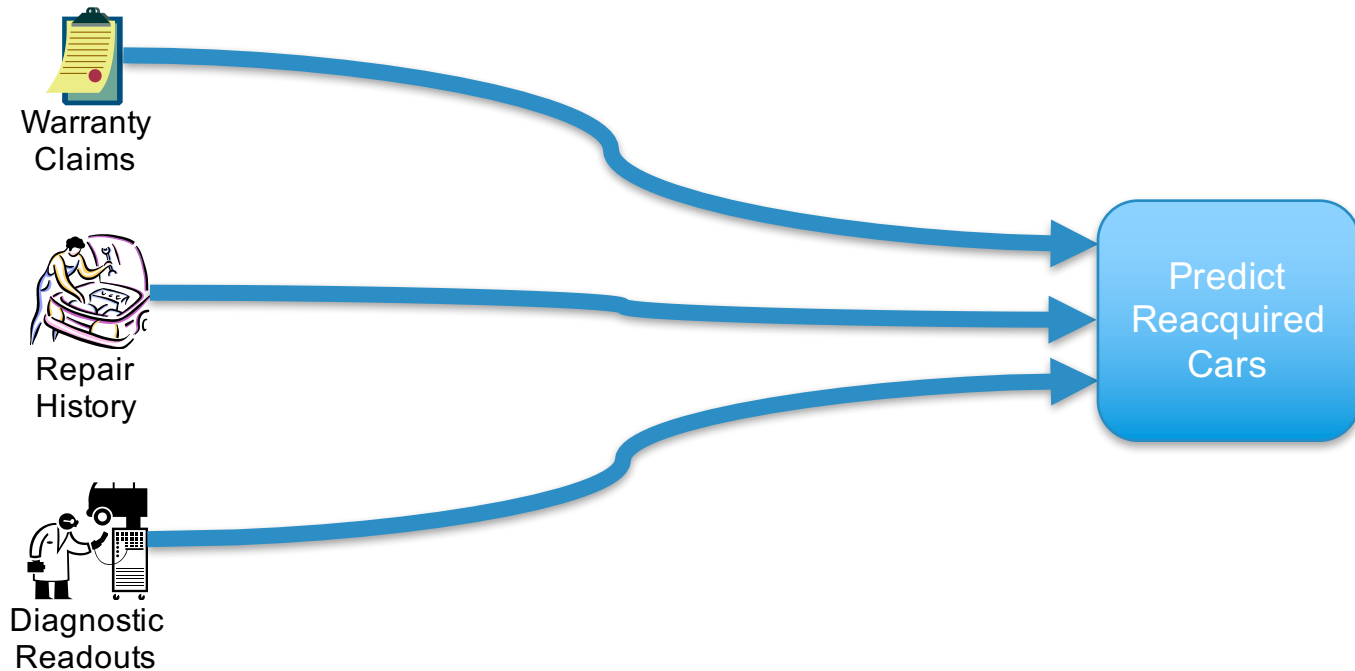
2007

2008

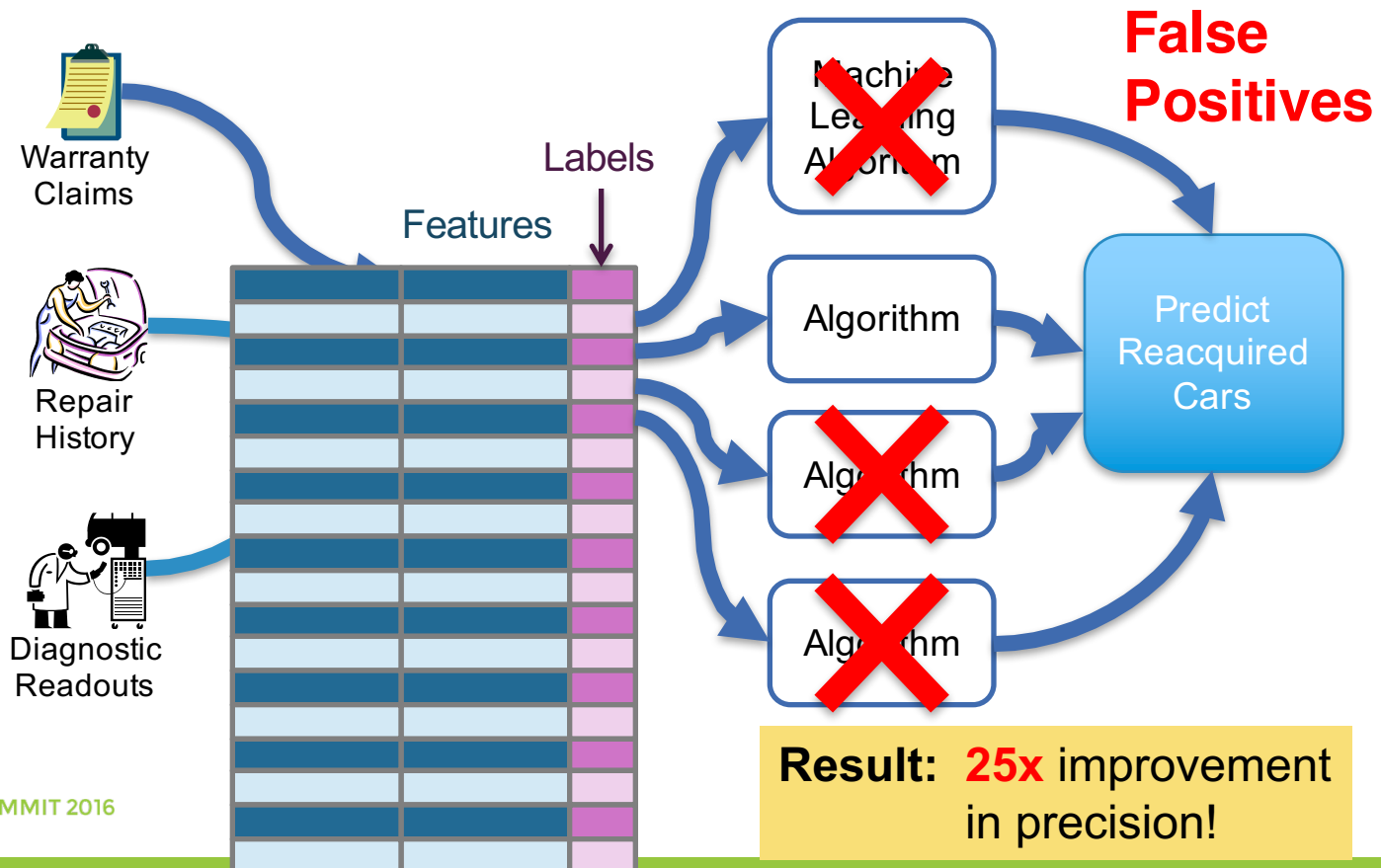
2009

2010

# Case Study: An Auto Manufacturer

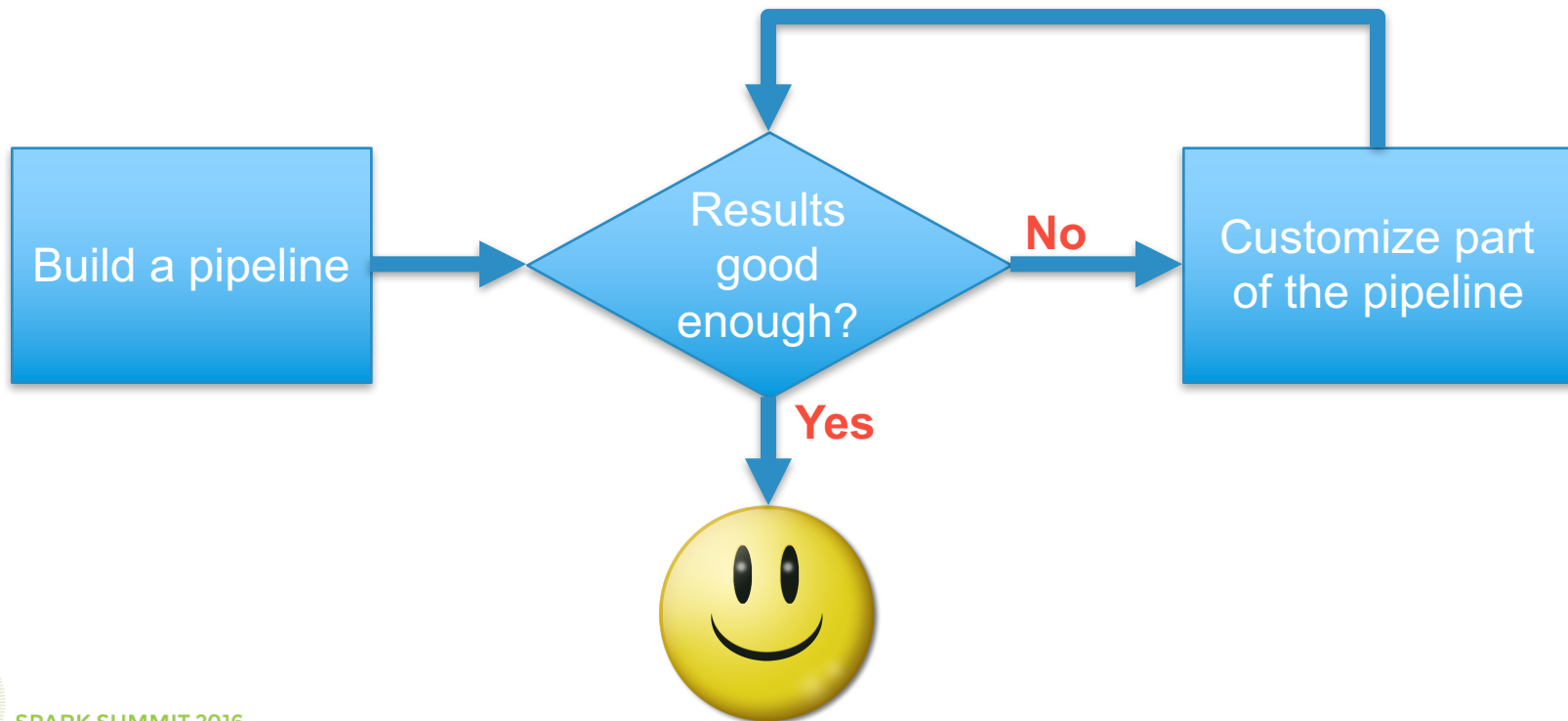


# Case Study: An Auto Manufacturer



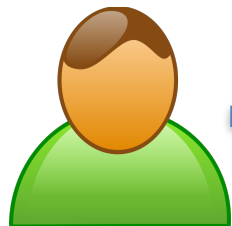


# The Iterative Development Process



# State-of-the-Art: Small Data

Data  
Scientist



```
4 X = read ($X); # explanatory variables
5 y = read ($Y); # predicted variables
6
7 n = nrow (X);
8 m = ncol (X);
9
10 # Reshape the columns of X if needed
11 scale_lambda = matrix (1, rows = 1, cols = m);
12 lambda = t(scale_lambda) * sreg;
13
14 # Constant a given by the user
15 A = t(X) %*% lambda;
16 b = t(X) %*% y;
17
18 beta = solve (A, b);
19 ...
20 write (beta, $B);
```

R or  
Python

Weather station data 2005

Weather station	Average temperatures (°C)	Min	Mean	Max	Daily rainfall (mm)	Mean	Max
RYOGGE	average temperatures	-18.5	7.5	23.8	average rainfall	2.0	34.8
NINIVEN - TOROKK	average temperatures	-18.4	4.8	23.8	average rainfall	1.2	35.9
TOROKK - FVB	average temperatures	-18.0	8.5	23.8	average rainfall	2.1	35.9
SOLA	average temperatures	-18.0	8.5	23.8	average rainfall	2.6	48.1
GABERMOEN	average temperatures	-15.4	5.4	19.4	average rainfall	2.0	49.5
BERGEN - FLORIDA	average temperatures	-5.2	19.2	34.2	average rainfall	8.4	190.5
LERDAL - SOLDO	average temperatures	-1.8	18.1	34.1	average rainfall	1.8	34.1
TAFROD	average temperatures	-1.8	18.1	34.1	average rainfall	5.2	64.5
VERNES	average temperatures	-1.8	18.1	34.1	average rainfall	2.5	47.0
RENA - RACEDALEN	average temperatures	-1.8	18.1	34.1	average rainfall	1.9	36.6
BERGVI	average temperatures	-1.8	18.1	34.1	average rainfall	4.0	39.5
TRONHO	average temperatures	-4.4	4.1	19.3	average rainfall	3.5	39.5
KAUTBERG	average temperatures	-20.9	-0.1	21.1	average rainfall	1.5	34.8
NY-ALBUND	average temperatures	-21.4	-0.4	19.2	average rainfall	0.9	39.1
JAN MAYEN	average temperatures	-11.6	0.8	11.1	average rainfall	2.0	31.3

Data




Personal  
Computer

Results

1	AAPL	31.05.2008	182.75	185.75
2	AAPL	08.08.2008	188.6	185.64
3	AAPL	13.08.2008	184.79	172.37
4	AAPL	20.08.2008	171.3	175.27
5	AAPL	27.08.2008	174.74	170.09
6	AAPL	03.07.2008	191.10	170.12
7	AAPL	03.07.2008	191.10	172.58
8	AAPL	03.07.2008	191.10	165.15
9	AAPL	03.07.2008	191.10	162.12
10	AAPL	01.08.2008	182.54	158.66
11	AAPL	08.08.2008	156.6	169.55
12	AAPL	15.08.2008	170.07	175.74
13	AAPL	22.08.2008	175.57	176.79
14	AAPL	29.08.2008	176.15	169.53



SPARK SUMMIT 2016

[illegible]

# R or Python


# Systems Programmer

# Scala

**Spark**

# Results

Rank	Symbol	Date	Value	Value
24	AAPL	30/05/2008	182.75	188.75
25	AAPL	06/06/2008	188.6	185.64
25	AAPL	13/06/2008	184.79	172.33
26	AAPL	20/06/2008	171.3	175.21
27	AAPL	27/06/2008	174.74	170.05
28	AAPL	03/07/2008	170.19	170.12
29	AAPL	09/07/2008	166.6	172.58
30	AAPL	16/07/2008	165.14	165.15
31	AAPL	25/07/2008	166.9	162.12
32	AAPL	01/08/2008	162.34	156.66
33	AAPL	08/08/2008	156.6	169.53
34	AAPL	15/08/2008	170.07	175.74
35	AAPL	22/08/2008	175.57	176.75
36	AAPL	29/08/2008	176.15	169.53

[illegible]

# Days or weeks per iteration




# Errors while translating algorithms



# Results

Rank	Symbol	Date	Value	Value
24	AAPL	30/05/2008	182.75	183.75
25	AAPL	08/06/2008	188.6	185.64
25	AAPL	13/06/2008	184.79	172.37
26	AAPL	20/06/2008	171.3	175.27
27	AAPL	27/06/2008	174.74	170.05
28	AAPL	03/07/2008	170.19	170.12
29	AAPL	09/07/2008	166.6	172.58
30	AAPL	16/07/2008	165.15	165.15
31	AAPL	25/07/2008	168.9	162.12
32	AAPL	01/08/2008	162.34	158.66
33	AAPL	08/08/2008	158.6	169.55
34	AAPL	15/08/2008	170.07	175.74
35	AAPL	22/08/2008	175.57	176.79
36	AAPL	29/08/2008	176.15	169.53



# R or Python

```

4 X = read.csv(Xfile) # explanatory variables
5 y = read.csv(Yfile) # predicted variables
6
7 n = nrow(X)
8 m = ncol(X)
9
10 # Rescale the columns of X 10^-4
11 scale_lambda = matrix(1, nrow = 1, cols = m)
12 lambda = t(scale_lambda) * sreg;
13
14 # Construct the matrix A
15 A = (X) %*% scale_lambda + lambda;
16 b = t(X) %*% y;
17
18 beta = solve(A, b);
19
20 write(beta, $B);

```

# SystemML

**Spark**

# Results

Rank	Symbol	Date	Value	Value
24	AAPL	30.05.2008	182.75	188.75
23	AAPL	08.06.2008	188.6	185.64
22	AAPL	13.06.2008	184.79	172.37
20	AAPL	20.06.2008	171.3	175.27
17	AAPL	27.06.2008	174.74	170.05
16	AAPL	03.07.2008	170.19	170.12
15	AAPL	07.07.2008	170.6	172.58
14	AAPL	09.07.2008	169.4	165.15
13	AAPL	25.07.2008	168.9	162.12
12	AAPL	01.08.2008	162.34	156.66
11	AAPL	08.08.2008	156.6	169.55
10	AAPL	15.08.2008	170.07	175.74
9	AAPL	22.08.2008	175.57	176.75
8	AAPL	29.08.2008	176.15	169.53



# R or Python

```

4 X = read.csv(X); # explanatory variables
5 y = read.csv(Y); # predicted variables
6
7 n = nrow(X);
8 m = ncol(X);
9
10 # Rescale the columns of X if needed
11 scale_lambda = matrix(1, rows = 1, cols = m);
12 lambda = t(scale_lambda) %>% rowSums()
13
14 # Construct the matrix A and the vector b
15 A = t(X) %>% rowSums(lambda) %>% as.numeric()
16 b = t(X) %>% rowSums(y) %>% as.numeric()
17
18 beta = solve(A, b);
19 ...
20 Write(beta, $B);

```



Fast iteration  
Same answer

# SystemML

**Spark**

# Results

24	AAPL	30/05/2008	182.75	188.75
25	AAPL	06/06/2008	188.6	185.64
25	AAPL	13/06/2008	184.79	172.37
26	AAPL	20/06/2008	171.3	175.27
27	AAPL	27/06/2008	174.74	170.05
28	AAPL	03/07/2008	170.19	170.12
29	AAPL	07/07/2008	176.6	172.58
30	AAPL	14/07/2008	165.4	165.15
31	AAPL	25/07/2008	168.9	162.12
32	AAPL	01/08/2008	162.34	156.66
33	AAPL	08/08/2008	156.6	169.55
34	AAPL	15/08/2008	170.07	175.74
35	AAPL	22/08/2008	175.57	176.75
36	AAPL	29/08/2008	176.15	169.53

**2007-2008:** Multiple projects at IBM Research – Almaden involving machine learning on Hadoop.

**2009:** We form a dedicated team for scalable ML

**2009-2010:** Through engagements with customers, we observe how data scientists create machine learning algorithms.

2007

2008

2009

2010

# Research

2011

2012

2013

2014



# Apache SystemML

**June 2015:** IBM  
Announces open-  
source SystemML

**September 2015:**  
Code available on  
Github

**November 2015:**  
SystemML enters  
Apache incubation

**February 2016:**  
First release (0.9) of  
Apache SystemML

**June 2016:**  
Second Apache  
release (0.10)

2015

2016

# SystemML at IBM Watson Health

- Built algorithms for predicting treatment outcomes
  - Substantial improvement in accuracy
- Moved from Hadoop MapReduce to Spark
  - SystemML supports both frameworks
  - **Exact same code**
  - **300X faster** on 1/40<sup>th</sup> as many nodes



# SystemML at Cadent Technology



Cadent is a leading provider of TV advertising and data solutions, reaching over 140 million homes and trusted by the world's largest service providers.

*“SystemML allows Cadent to implement advanced numerical programming methods in Apache Spark, empowering us to leverage specialized algorithms in our predictive analytics software.”*

Michael Zargham  
Chief Scientist



SPARK SUMMIT 2016

# Demo!



**SPARK SUMMIT 2016**  
DATA SCIENCE AND ENGINEERING AT SCALE  
JUNE 6-8, 2016 SAN FRANCISCO

# Demo Scenario

- **Application:** Targeted ads using demographic information tied to cookies
- **Problem:** The information is incomplete
- **Solution:** Estimate the missing values
  - Treat the problem as a **matrix completion** problem



# Data

- The U.S. Census Public Use Microdata Sample (PUMS) data set for 2010
- 10% sample of the U.S. population
  - We'll use just California today
- Use this full data set to generate synthetic incomplete data

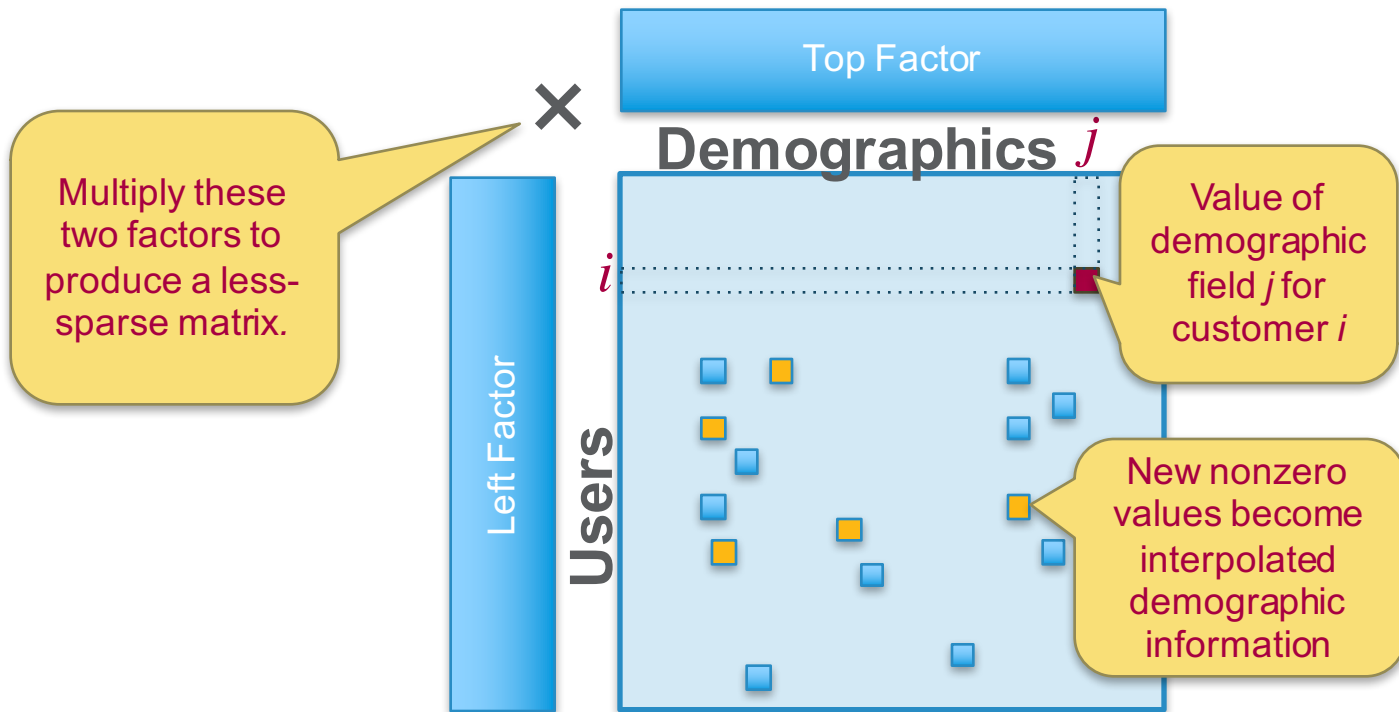


# Demo Scenario

- **Application:** Identify products that are complementary (often purchased together)
- **Problem:** Customers are not currently buying the best complements at the same time
- **Solution:** Suggest new product pairings
  - Treat the problem as a **matrix completion** problem

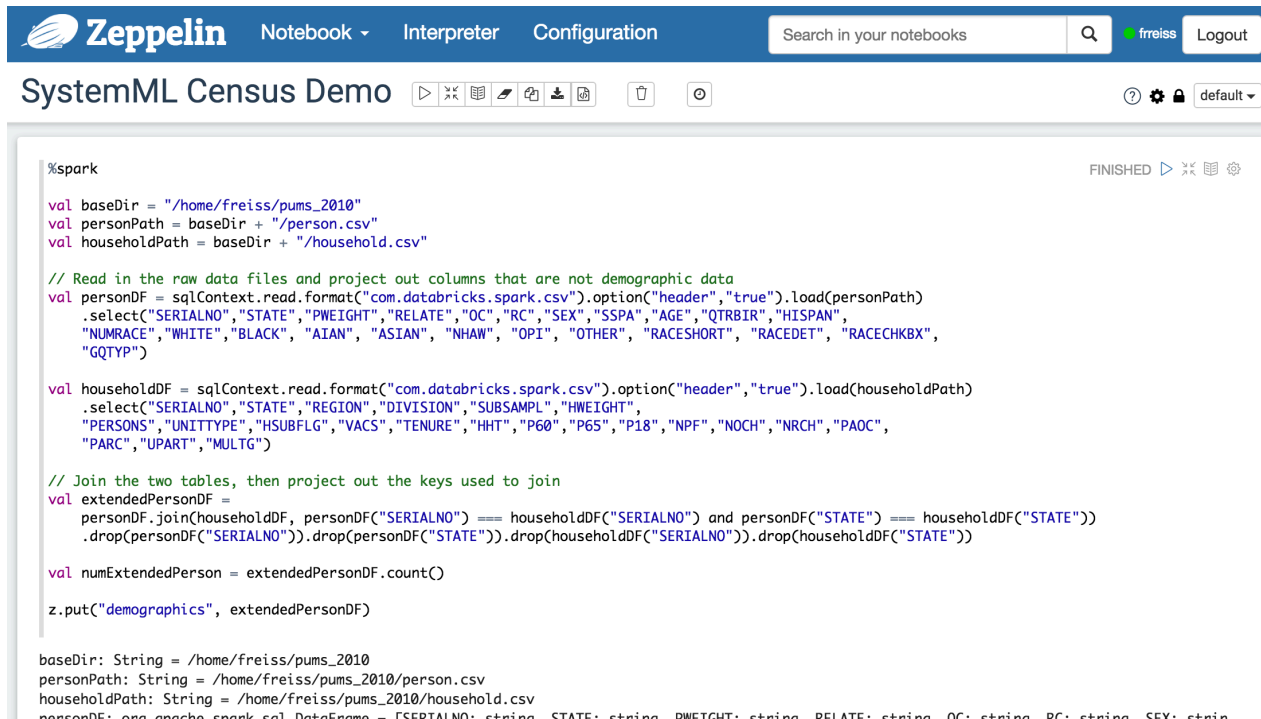


# Matrix Factorization





# Demo Part 1: Data wrangling



The image shows a Zeppelin Notebook interface. The top bar is blue with the Zeppelin logo, 'Notebook' dropdown, 'Interpreter', 'Configuration', a search bar, a user profile 'freiss', and a 'Logout' button. Below the bar, the notebook title is 'SystemML Census Demo'. The main area contains a Scala script for loading and joining census data. The script defines baseDir, personPath, and householdPath. It reads person and household data into DataFrames, selects specific columns, joins them on state, and counts the result. The status 'FINISHED' is shown in the top right of the code editor.

```
%spark
val baseDir = "/home/freiss/pums_2010"
val personPath = baseDir + "/person.csv"
val householdPath = baseDir + "/household.csv"

// Read in the raw data files and project out columns that are not demographic data
val personDF = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").load(personPath)
  .select("SERIALNO", "STATE", "PWEIGHT", "RELATE", "OC", "RC", "SEX", "SSPA", "AGE", "QTRBIR", "HISPAN",
    "NUMRACE", "WHITE", "BLACK", "AIAN", "ASIAN", "NHAW", "OPI", "OTHER", "RACESHORT", "RACEDET", "RACECHKBX",
    "GQTP")

val householdDF = sqlContext.read.format("com.databricks.spark.csv").option("header", "true").load(householdPath)
  .select("SERIALNO", "STATE", "REGION", "DIVISION", "SUBSAMPL", "HWEIGHT",
    "PERSONS", "UNITTYPE", "HSUBFLG", "VACS", "TENURE", "HHT", "P60", "P65", "P18", "NPF", "NOCH", "NRCH", "PAOC",
    "PARC", "UPART", "MULTG")

// Join the two tables, then project out the keys used to join
val extendedPersonDF =
  personDF.join(householdDF, personDF("SERIALNO") === householdDF("SERIALNO") and personDF("STATE") === householdDF("STATE"))
  .drop(personDF("SERIALNO")).drop(personDF("STATE")).drop(householdDF("SERIALNO")).drop(householdDF("STATE"))

val numExtendedPerson = extendedPersonDF.count()

z.put("demographics", extendedPersonDF)

baseDir: String = /home/freiss/pums_2010
personPath: String = /home/freiss/pums_2010/person.csv
householdPath: String = /home/freiss/pums_2010/household.csv
personDF: org.apache.spark.sql.DataFrame = [SERIALNO: string, STATE: string, PWEIGHT: string, RELATE: string, OC: string, RC: string, SEX: string,
```



# Demo Part 2: Custom algorithm

## Algorithm Customizability

ML algorithms are expressed in an R-like or Python-like syntax that includes linear algebra primitives, statistical functions, and ML-specific constructs. This high-level language significantly increases the productivity of data scientists as it provides (1) full flexibility in expressing custom analytics, and (2) data independence from the underlying input formats and physical data representations. Automatic optimization according to data and cluster characteristics ensures both efficiency and scalability.

### Poisson Nonnegative Matrix Factorization in SystemML's R-like Syntax

```
while (iter < max_iterations) {  
  iter = iter + 1;  
  H = (H * (t(W) %*% (V/(W%*%H)))) / t(colSums(W));  
  W = (W * ((V/(W%*%H)) %*% t(H))) / t(rowSums(H));  
  obj = as.scalar(colSums(W) %*% rowSums(H)) - sum(V * log(W%*%H));  
  print("iter=" + iter + " obj=" + obj);  
}
```



# Key Points

- SystemML, Spark, and Zeppelin work together
- Linear algebra is great for data science
- Customization is important



# How to get Apache SystemML



**SPARK SUMMIT 2016**  
DATA SCIENCE AND ENGINEERING AT SCALE  
JUNE 6-8, 2016 SAN FRANCISCO

# The Apache SystemML Web Site

<http://systemml.apache.org>

The screenshot shows the Apache SystemML website interface. At the top left is the Apache SystemML logo, consisting of a red circle with 'ML' inside. To its right is the text 'Apache SystemML'. Below this, the word 'Apache' is partially visible in large white letters. A dark blue navigation bar at the top contains links for 'Community', 'GitHub', 'Documentation', 'Download', and 'Apache'. The 'Community' link is highlighted with a blue background. A dropdown menu for 'Community' is open, showing 'Get Involved' and 'Who we are'. A yellow callout bubble points to the 'Download' link with the text 'Download the binary release!'. Another yellow callout bubble points to the 'GitHub' link with the text 'Browse the source!'. A third yellow callout bubble points to the 'Documentation' link with the text 'Try out some tutorials!'. A fourth yellow callout bubble points to the 'Get Involved' link with the text 'Contribute to the project!'. A blue button labeled 'Get SystemML' is located at the bottom left. The main content area below the navigation bar contains the text 'Apache SystemML is a distributed and declarative machine learning platform.'

ML Apache SystemML

Community

GitHub Documentation Download Apache

Get Involved  
Who we are

Download the binary release!

Browse the source!

Try out some tutorials!

Contribute to the project!

Get SystemML

Apache SystemML is a distributed and declarative machine learning platform.

# THANK YOU.

Please try out Apache SystemML!

<http://systemml.apache.org>

Special thanks to Nakul Jindal and Mike Dusenberry for helping with the demo!



**SPARK SUMMIT 2016**  
DATA SCIENCE AND ENGINEERING AT SCALE  
JUNE 6-8, 2016 SAN FRANCISCO