IMPRO3

Logistic Regression

16.07.2014

Markus Holzemer
Jonas Traub
Timo Walther

Agenda

- Motivation
- Logistic Regression Algorithm
- Parallelization Strategy
- Implementation in Stratosphere and Spark
- Experiment Results

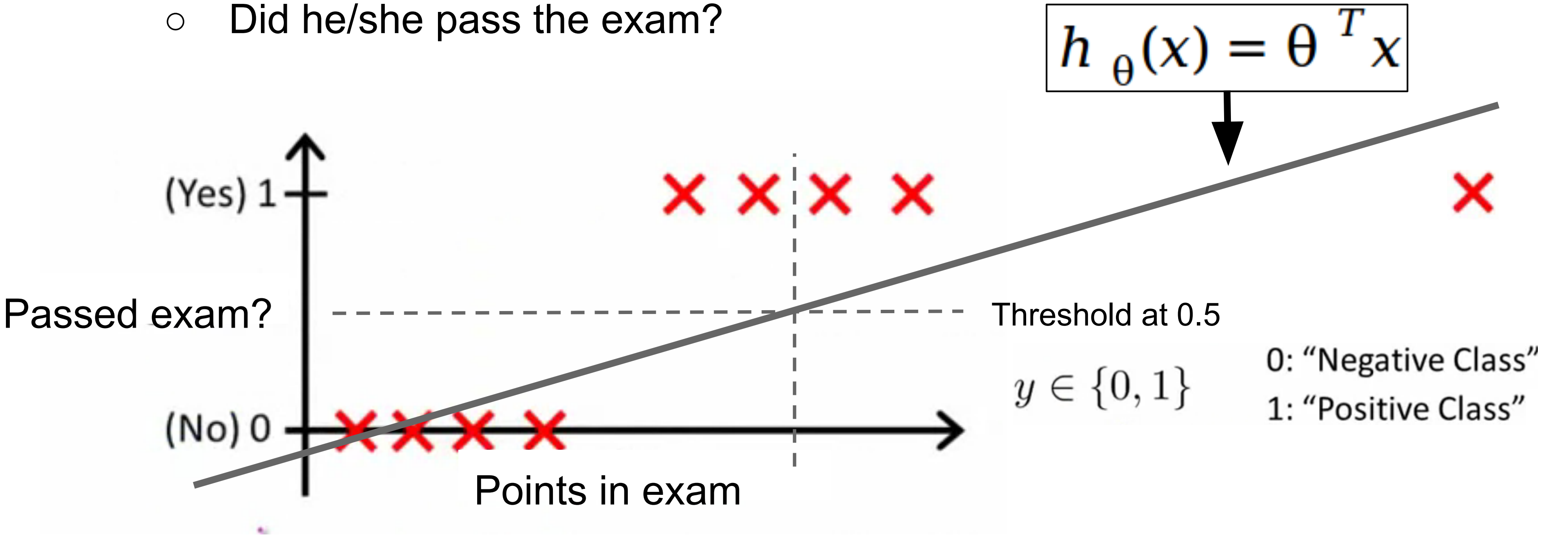
Motivation

- Logistic Regression is for **Classification**
- Typically binary classification
  - Is this mail spam?
  - Did he/she pass the exam?

source: coursera/Stanford Machine Learning by Andrew Ng

Motivation

- Logistic Regression is for **Classification**
- Typically binary classification
  - Is this mail spam?
  - Did he/she pass the exam?

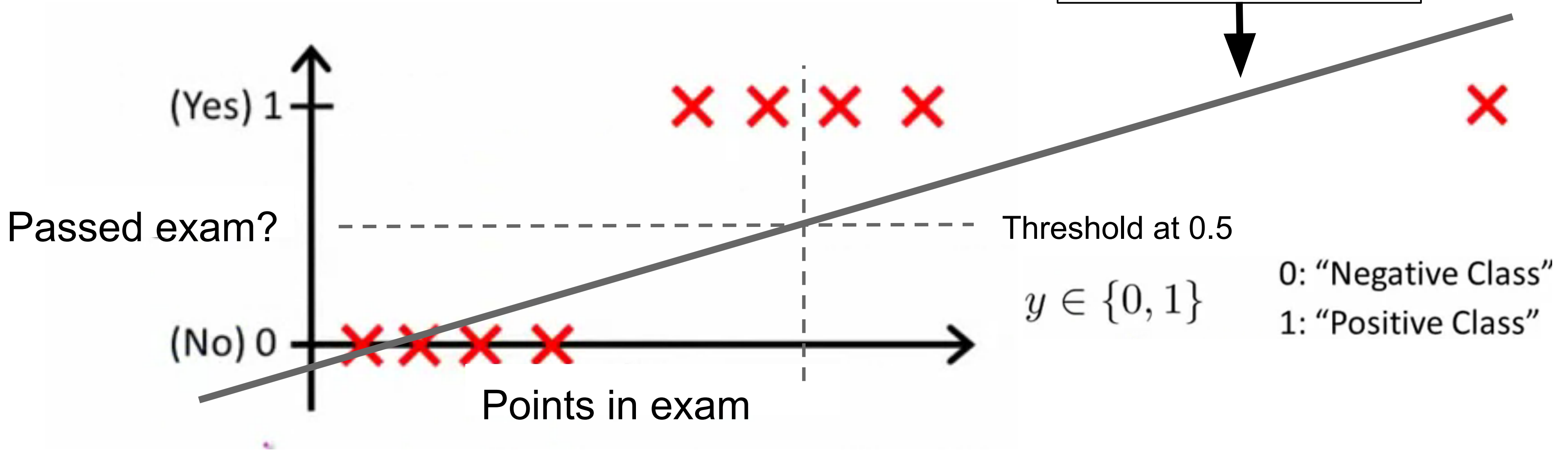


source: coursera/Stanford Machine Learning by Andrew Ng

Motivation

- Logistic Regression is for **Classification**
- Typically binary classification
  - Is this mail spam?
  - Did he/she pass the exam?

$$\boxed{h_\theta(x) = \theta^T x}$$



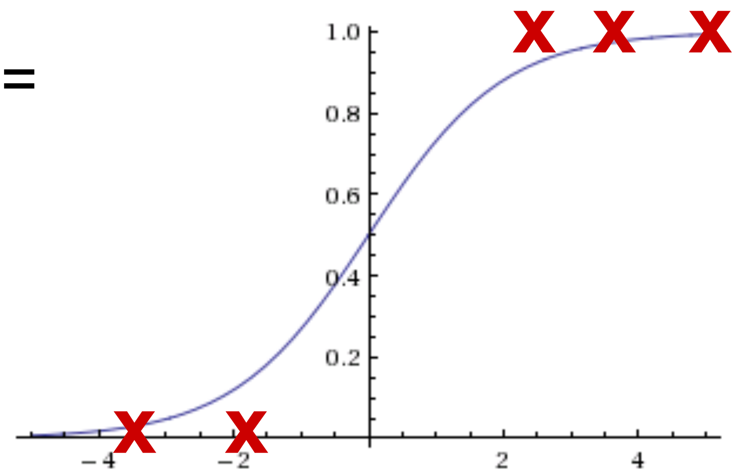
- In example: $h_\theta(x) < 0 \; and \; h_\theta(x) > 1 \; are \; possible$
- With Logistic Regression: $0 \leq h_\theta(x) \leq 1$

source: coursera/Stanford Machine Learning by Andrew Ng

Sigmoid Function = Logistic Function =

$$g(z) = \frac{1}{1+e^{-z}} \quad with\ h_\theta(x) = g(\theta^T x)$$

$$\Rightarrow h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$



wolframalpha.com: plot g(z)=(1/(1+e^(-z))) from z = -5 to +5

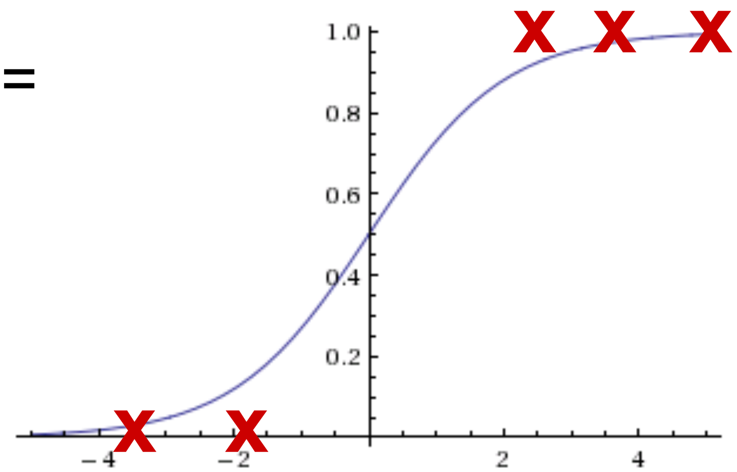
source: coursera/Stanford Machine Learning by Andrew Ng

Hypothesis and Cost Representation

Sigmoid Function = Logistic Function =

$$g(z) = \frac{1}{1+e^{-z}} \quad \text{with } h_\theta(x) = g(\theta^T x)$$

$$\Rightarrow h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$



wolframalpha.com: plot g(z)=(1/(1+e^(-z))) from z = -5 to +5

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}) \qquad Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & if\ y=1 \\ -log(1-h_\theta(x)) & if\ y=0 \end{cases}$$

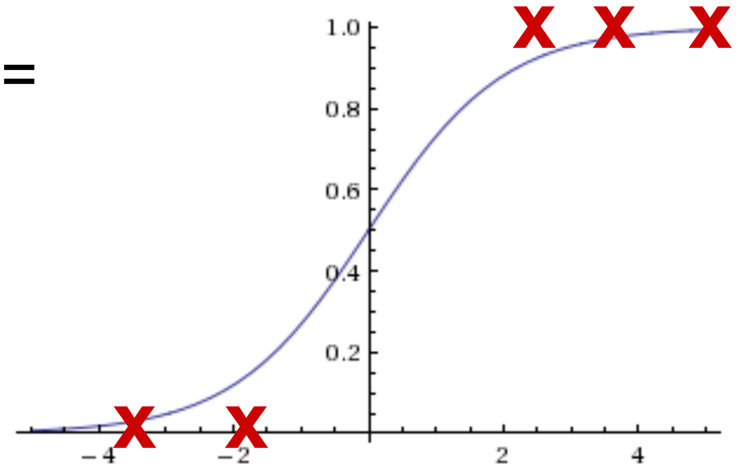
source: coursera/Stanford Machine Learning by Andrew Ng

Hypothesis and Cost Representation

Sigmoid Function = Logistic Function =

$$g(z) = \frac{1}{1+e^{-z}} \quad with\ h_\theta(x) = g(\theta^T x)$$

$$\Rightarrow h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$



wolframalpha.com: plot g(z)=(1/(1+e^(-z))) from z = -5 to +5

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}) \qquad Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & if\ y=1 \\ -log(1-h_\theta(x)) & if\ y=0 \end{cases}$$

=> We want to minimize cost J

=> Gradient Descent, repeat:

$$\theta_j = \theta_j - \alpha \frac{\Delta J(\theta)}{\Delta \theta_j} with\ \frac{\Delta J(\theta)}{\Delta \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\, x_j^{(i)}$$

source: coursera/Stanford Machine Learning by Andrew Ng

Pseudocode

```
X = [m, n]       // training set of features
y = [m]          // vector of classification
alpha = 1        // learning rate
theta = [n] -> all 0

Gradient descent:
for 1:number_iterations
   for i = 1:n
      grad(i) = 0;
      for j = 1:m
```

$$\texttt{grad(i) += }\overbrace{\texttt{(}\underbrace{\texttt{sigmoid(X(j,:)*theta)}}_{h(x)}\texttt{-y(j))*X(j,i))}}^{\textit{derivative of cost function}}\texttt{;}$$

```
      end
      grad(i) = grad(i)/m;
   end
   theta = theta - alpha * grad;
end
```

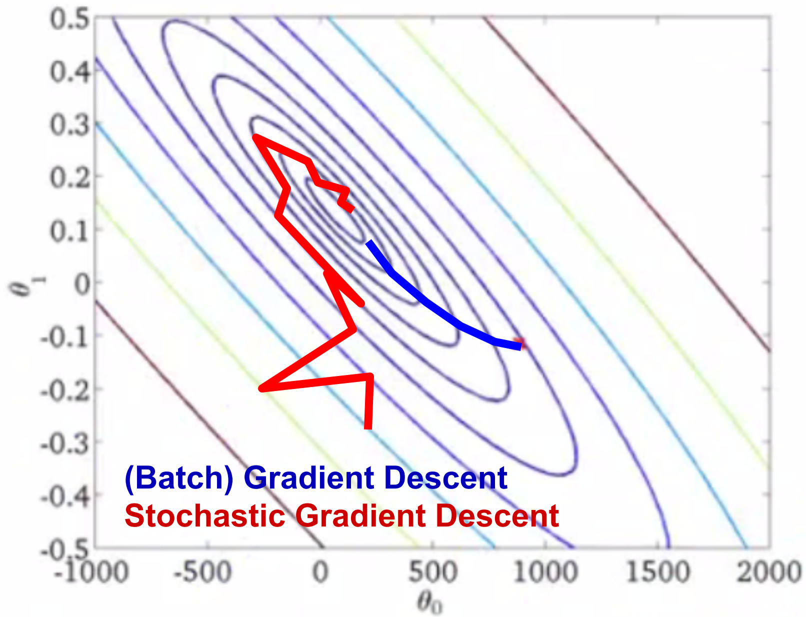
Very naive way, can be vectorized

Stochastic Gradient Descent (for large training sets)

```
X = [m, n]        // training set of features
y = [m]           // vector of classification
alpha = 1         // learning rate
theta = [n] -> all 0

Stochastic Gradient Descent:

Randomly_Shuffle_Training_Set(X,y)
repeat until theta converges
    for j = 1:m
      for i = 1:n
        grad(i) = (sigmoid(X(j,:)*theta)-y(j))*X(j,i));

      end
      theta = theta - alpha * grad;
    end
end
```

different cost function but same derivative

h(x)



=> make progress in each iteration (modify the parameters to fit the training set a little bit better)

=> generally, move the parameters in the direction of the global minimum

Parallelization

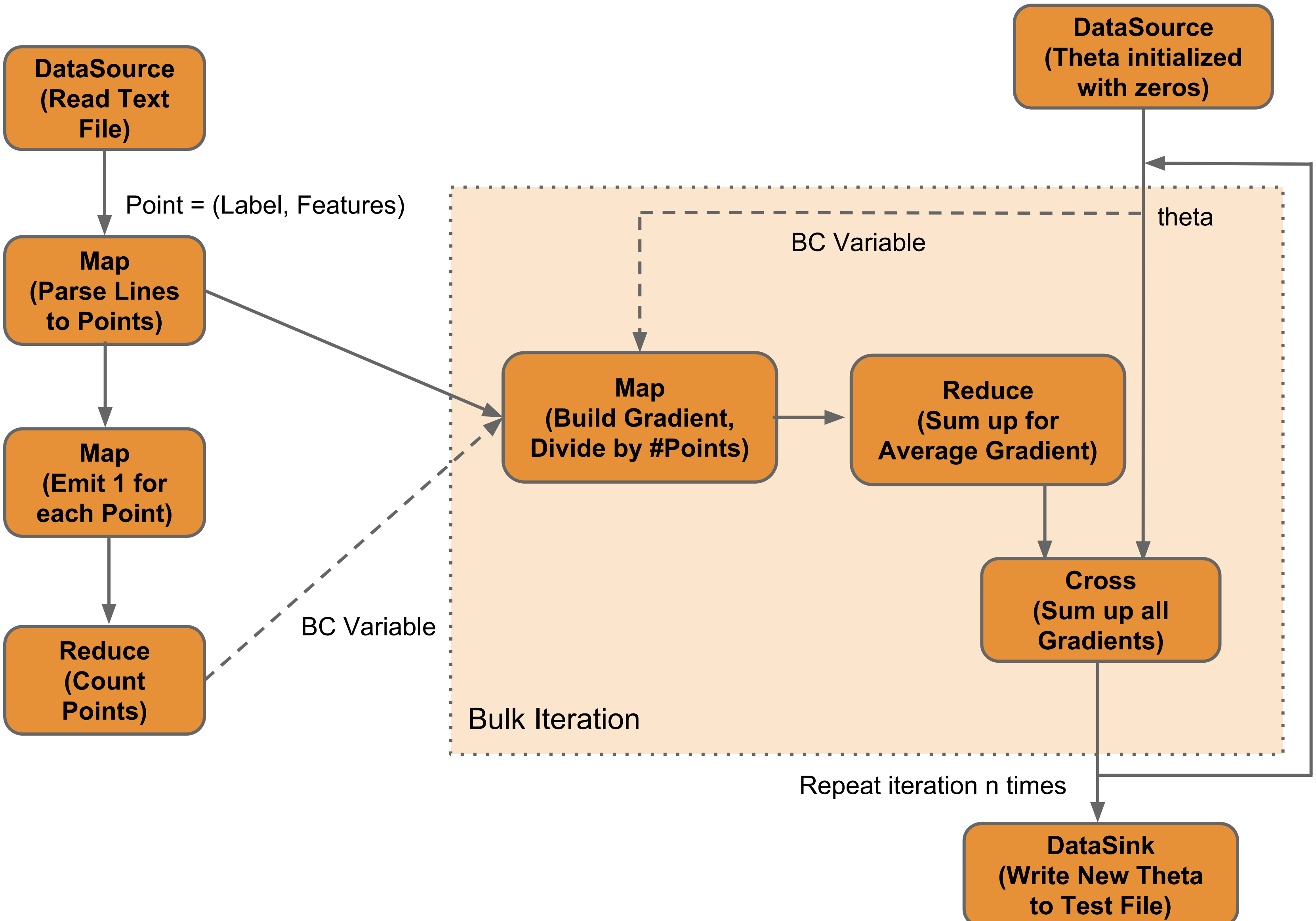
- **Stochastic Gradient Descent**
  - Inherently not parallelizable (theta needs to be adjusted after every point)
  - Parallelization over different alphas or different distributions of the training set and averaging? Research is ongoing.

- **Batch Gradient Descent**
  - Parallel computation of the average gradient over all points possible (see next slide)
  - But: Not clear if it is profitable in comparison to a local SGD

=> Both SGD and BGD has been implemented in Scala

=> We use Batch Gradient Descent for Stratosphere and Spark to enable parallel performance measures

Parallel Batch Gradient Descent



Comparison of Implementations

StratoSphere
Above the Clouds

1. Explicit iteration operator
2. Usage of broadcast variables
3. Data represented as POJOs extending from Tuple

Spark
Lightning-Fast Cluster Computing

1. Iteration as Java for-loop
2. Operator output represented by Java variables
3. Data represented as POJOs

Issues during the Project

1. Issues reported to Jira/Git

   a. **GIT #905** - Using broadcast variables in UDFs within iterations leads to CompilerException
   ==> Solved with 0.5.1-SNAPSHOT

   b. **FLINK-929** - Impossible to pass double with configuration
   ==> Solved with Pull Request #13

   c. **FLINK-1018** - Logistic Regression deadlocks
   ==> Work in progress / Workaround is present

   ==> Needs stability and robustness

1. Java 6 on the cluster sucks!
   a. No JDK6 from Oracle available any more
   b. No Lambda Rules...

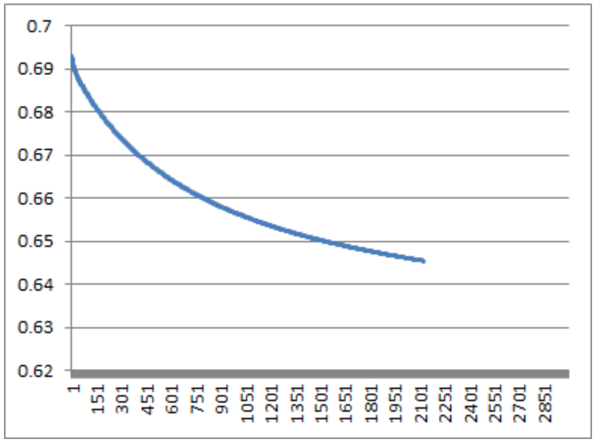
Performance Test Setup

- Cluster
  - 4 Nodes á 16 Cores, 32 GB RAM
  - Hadoop 1.2.1
  - Stratosphere 0.5.1
  - Spark 1.0
  - Java(TM) SE Runtime Environment (build 1.6.0_26-b03)

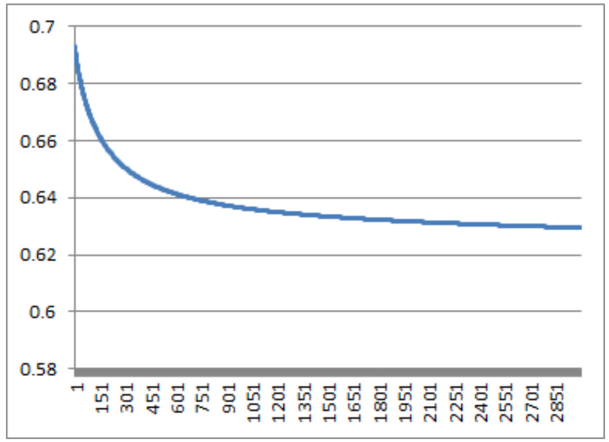
- Testruns
  - Every experiment repeated 7 times
  - Run with different datasize
  -
- Datasets
  - We used the Higgs Dataset from the UCI Repository
  - binary classified (0/1)
  - 28 dimensions with double numbers
  - S size: subsample of ~75MB
  - XL size: full dataset of ~7.5GB

Hyperparameter Finding

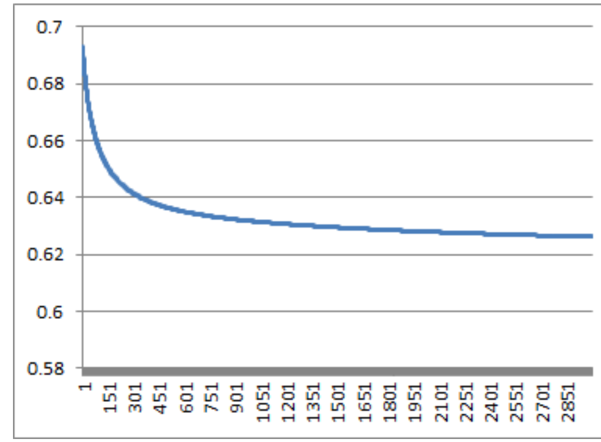
Goal: Find good learning rate alpha and reasonable number of iterations
Approach: Test and print costs of different rates locally by using a sample



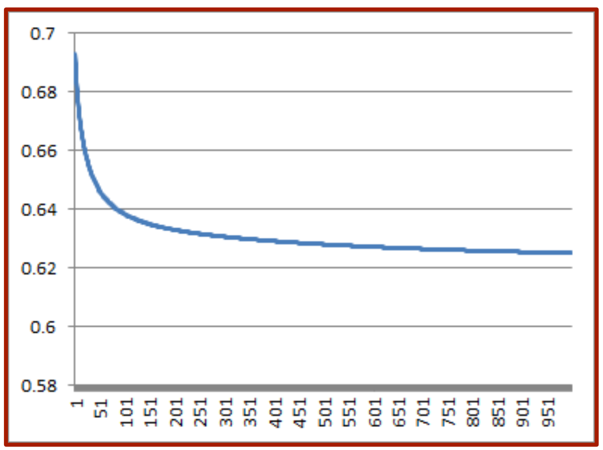
alpha = 0.01



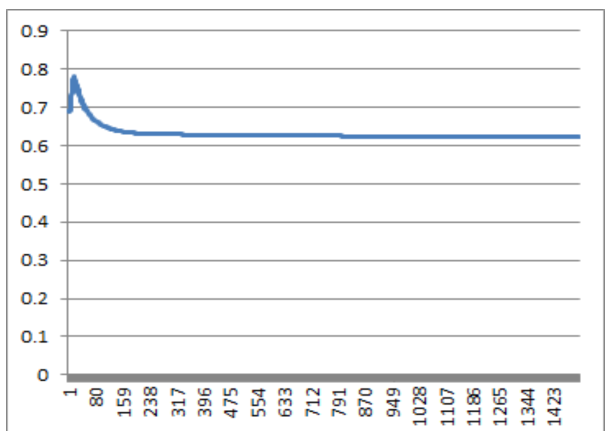
alpha = 0.05



alpha = 0.1



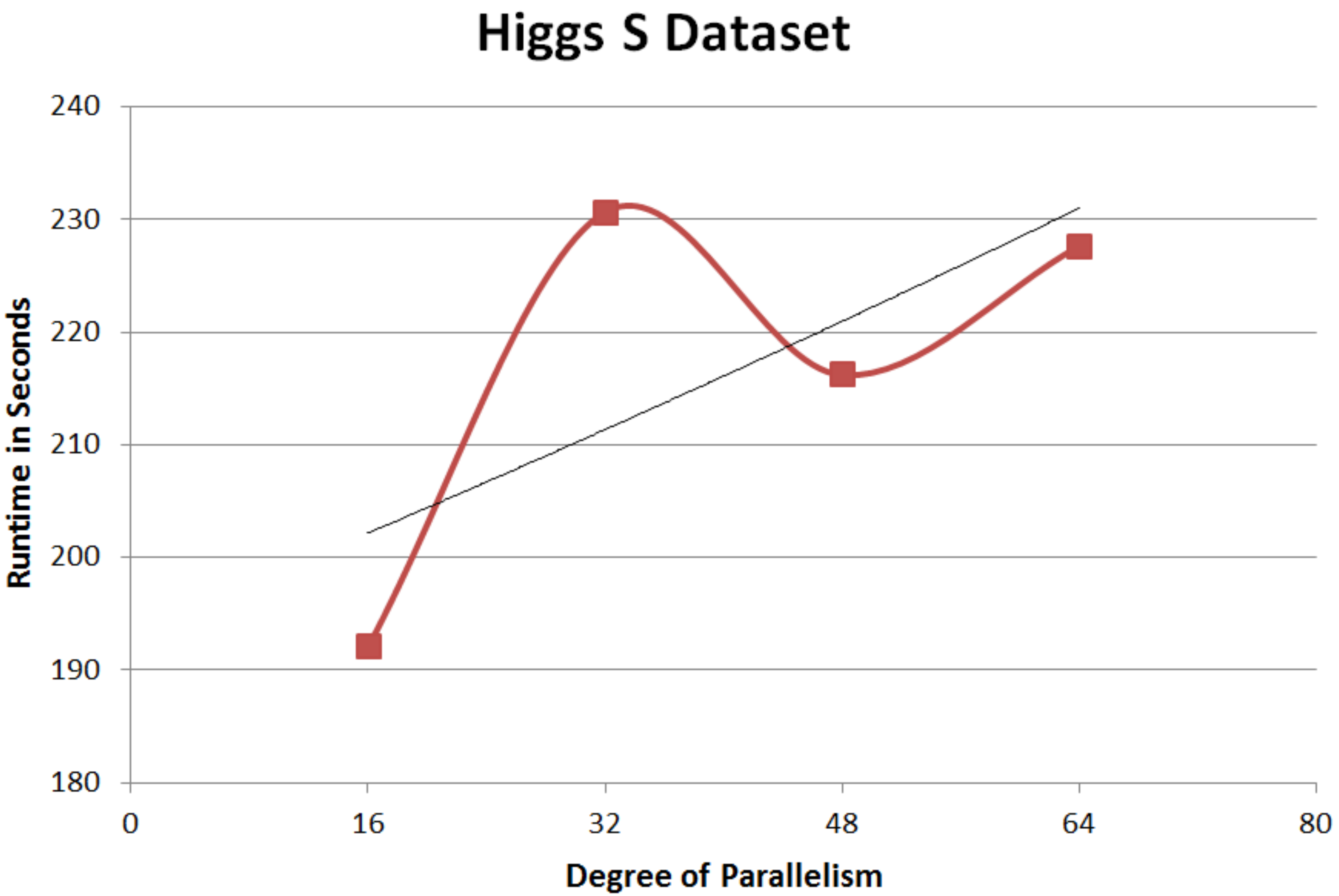
alpha = 0.4



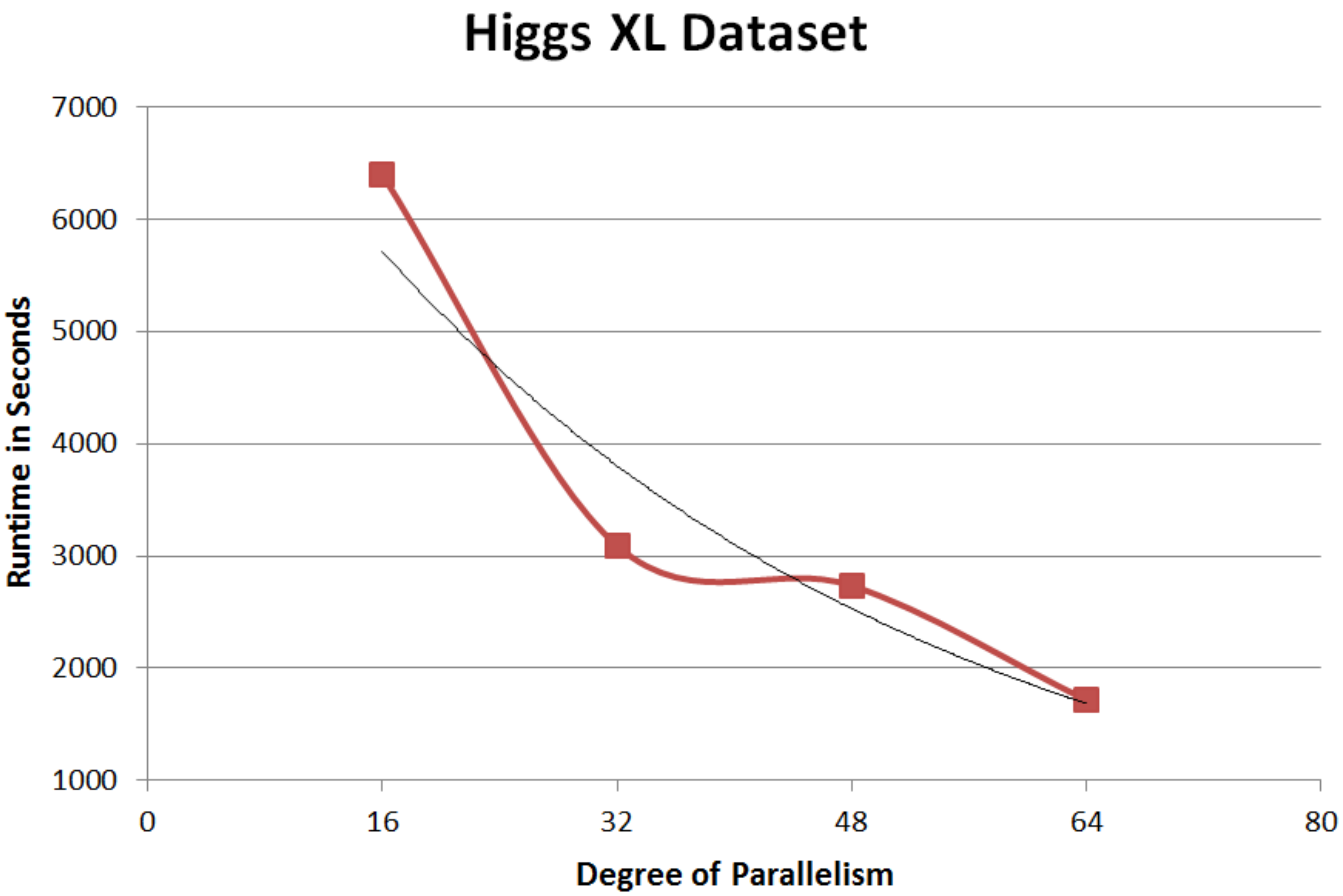
alpha = 0.5

=> alpha = 0.4
=> 750 iterations

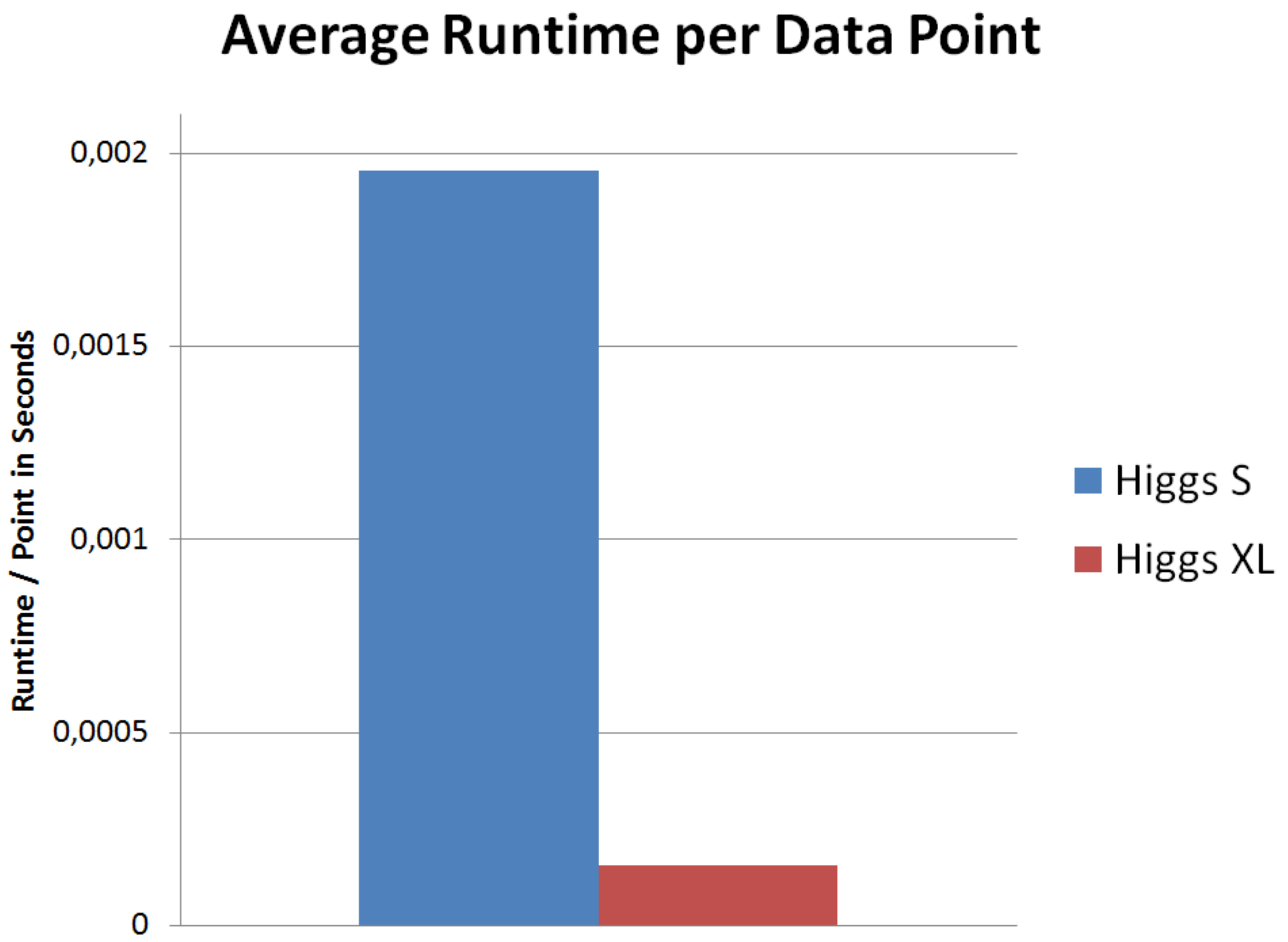
Test Results Stratosphere



Test Results Stratosphere



Test Results Stratosphere



Resume

System	StratoSphere Above the Clouds	Spark Lightning-Fast Cluster Computing
Test Results	+ Well scaling observed + Huge speedup through BC vars - bad performance on small data - sometimes unreliable	Run with the XL dataset for 1h and then aborted. Only the master node was used for the computation. Further investigations are necessary

=> Stratosphere gives good results! For Spark we don’t know...

General Impressions	+ Fast support via Jira/Git + Easy to use data model - Several bugs found - Hard to get it running	+ really nice Java API + Easy to use data model - Java 8 dependent documentation - Even harder to get it running

=> Both tools provide a nice programming abstraction
=> but the runtime needs to get more stable

Questions?