UNIVERSITY OF WATERLOO

Big Data Infrastructure

CS 489/698 Big Data Infrastructure (Winter 2016)

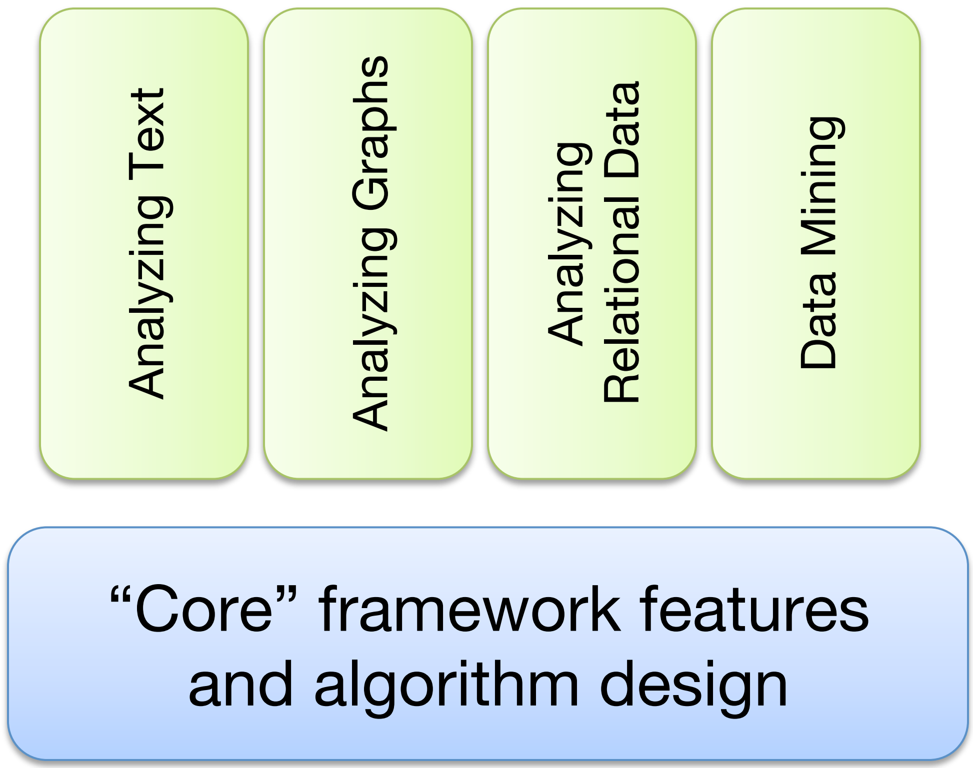
Week 8: Data Mining (1/4)

March 1, 2016

Jimmy Lin
David R. Cheriton School of Computer Science
University of Waterloo

These slides are available at http://lintool.github.io/bigdata-2016w/

Structure of the Course



Supervised Machine Learning

The generic problem of function induction given sample instances of input and output

Focus today

Classification: output draws from finite discrete labels

Regression: output is a continuous value

This is not meant to be an exhaustive treatment of machine learning!

Classification

Source: Wikipedia (Sorting)

Applications

Spam detection

Sentiment analysis

Content (e.g., genre) classification

Link prediction

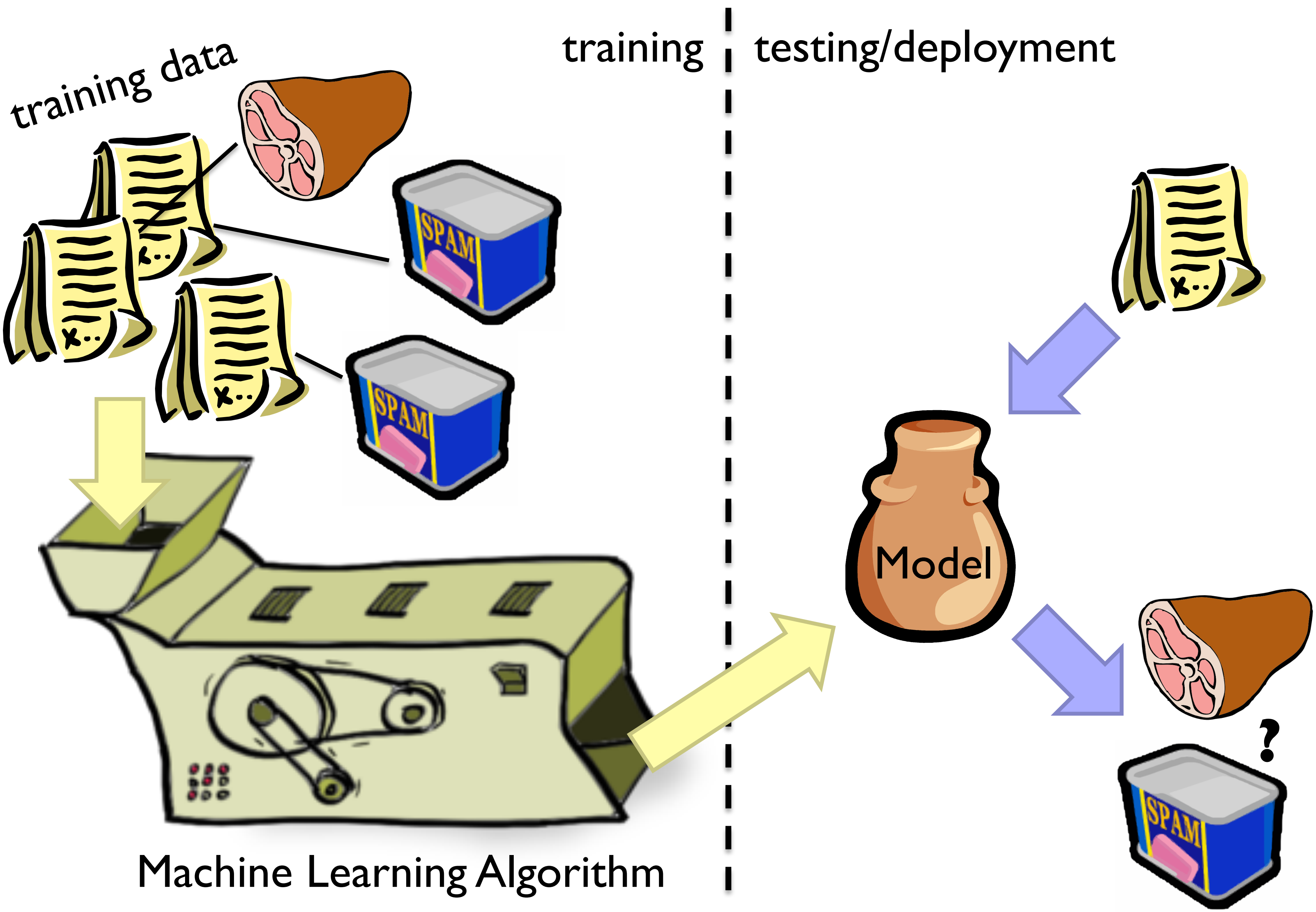
Document ranking

Object recognition

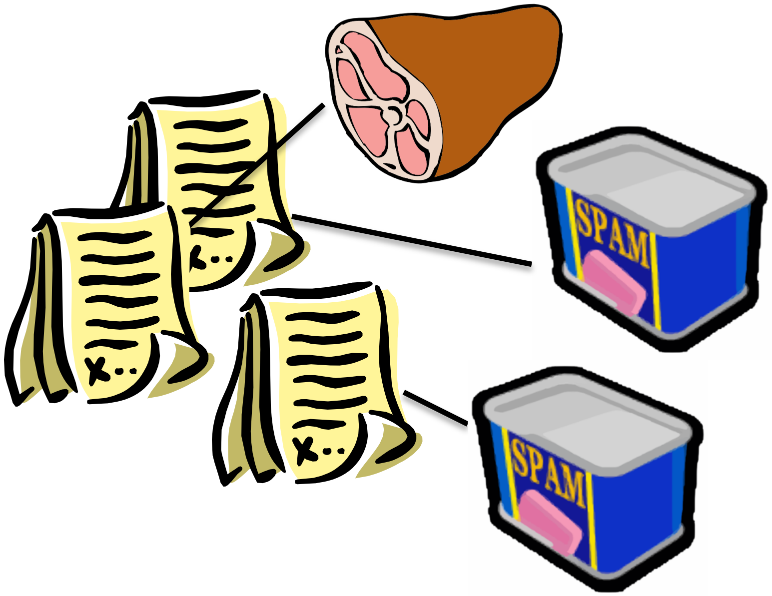
Fraud detection

And much much more!

Supervised Machine Learning



Feature Representations



Who comes up with the features?

How?

Objects are represented in terms of features:

"Dense" features: sender IP, timestamp, # of recipients, length of message, etc.

"Sparse" features: contains the term "viagra" in message, contains "URGENT" in subject, etc.

Applications

Spam detection

Sentiment analysis

Content (e.g., genre) classification

Link prediction

Document ranking

Object recognition

Fraud detection

And much much more!

Features are highly application-specific!

Components of a ML Solution

Data

Features

Model

Optimization

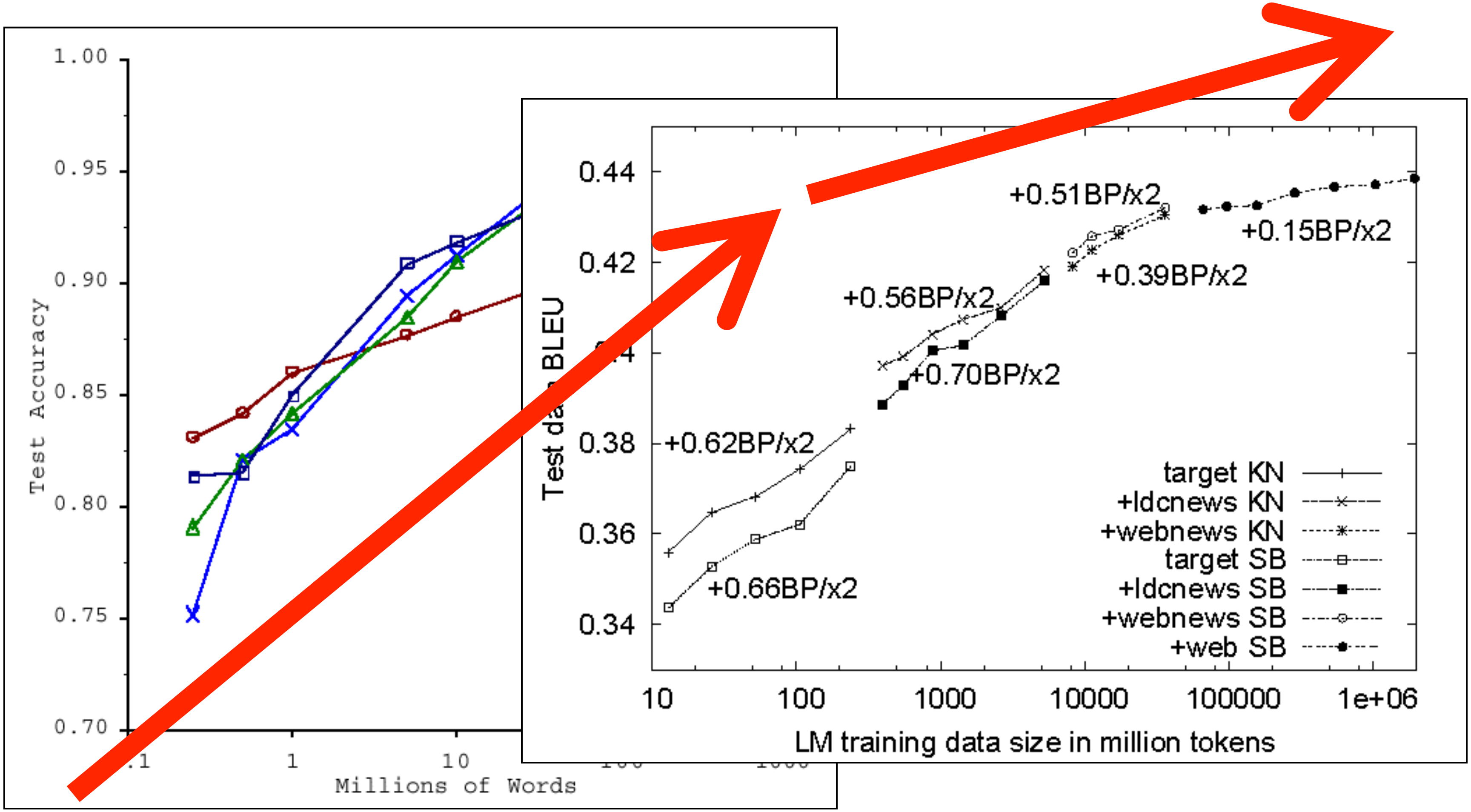
gradient descent, stochastic gradient descent, L-BFGS, etc.

logistic regression, naïve Bayes, SVM, random forests, perceptrons, neural network, etc.

What "matters" the most?

No data like more data!

s/knowledge/data/g;



(Banko and Brill, ACL 2001)
(Brants et al., EMNLP 2007)

Limits of Supervised Classification?

- Why is this a big data problem?
  - Isn't gathering labels a serious bottleneck?
- Solution: crowdsourcing
- Solution: bootstrapping, semi-supervised techniques
- Solution: user behavior logs
  - Learning to rank
  - Computational advertising
  - Link recommendation
- The virtuous cycle of data-driven products

Supervised Binary Classification

- Restrict output label to be *binary*
  - Yes/No
  - 1/0
- Binary classifiers form a primitive building block for multi-class problems
  - One vs. rest classifier ensembles
  - Classifier cascades

The Task

- Given $D = \{(\mathrm{x}_i, y_i)\}_i^n$

  label

  (sparse) feature vector

  $$\mathrm{x}_i = [x_1, x_2, x_3, \ldots, x_d]$$
  $$y \in \{0, 1\}$$

- Induce $f : X \to Y$
  - Such that loss is minimized

  $$\frac{1}{n} \sum_{i=0}^{n} \ell(f(\mathrm{x}_i), y_i)$$

  loss function

- Typically, consider functions of a parametric form:

  $$\arg\min_{\theta} \frac{1}{n} \sum_{i=0}^{n} \ell(f(x_i; \theta), y_i)$$

  model parameters

Key insight: machine learning as an optimization problem!

(closed form solutions generally not possible)

Gradient Descent: Preliminaries

- Rewrite:

$$\arg\min_{\theta} \frac{1}{n} \sum_{i=0}^{n} \ell(f(\mathrm{x}_i;\theta), y_i) \qquad \arg\min_{\theta} L(\theta)$$

- Compute gradient:
  - "Points" to fastest increasing "direction"

$$\nabla L(\theta) = \left[\frac{\partial L(\theta)}{\partial w_0}, \frac{\partial L(\theta)}{\partial w_1}, \ldots \frac{\partial L(\theta)}{\partial w_d} \right]$$

- So, at any point: *

$$\mathrm{b} = \mathrm{a} - \gamma \nabla L(\mathrm{a})$$

$$L(\mathrm{a}) \geq L(\mathrm{b})$$

* caveats

Gradient Descent: Iterative Update

- Start at an arbitrary point, iteratively update:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \nabla L(\theta^{(t)})$$

- We have:

$$L(\theta^{(0)}) \geq L(\theta^{(1)}) \geq L(\theta^{(2)}) \ldots$$

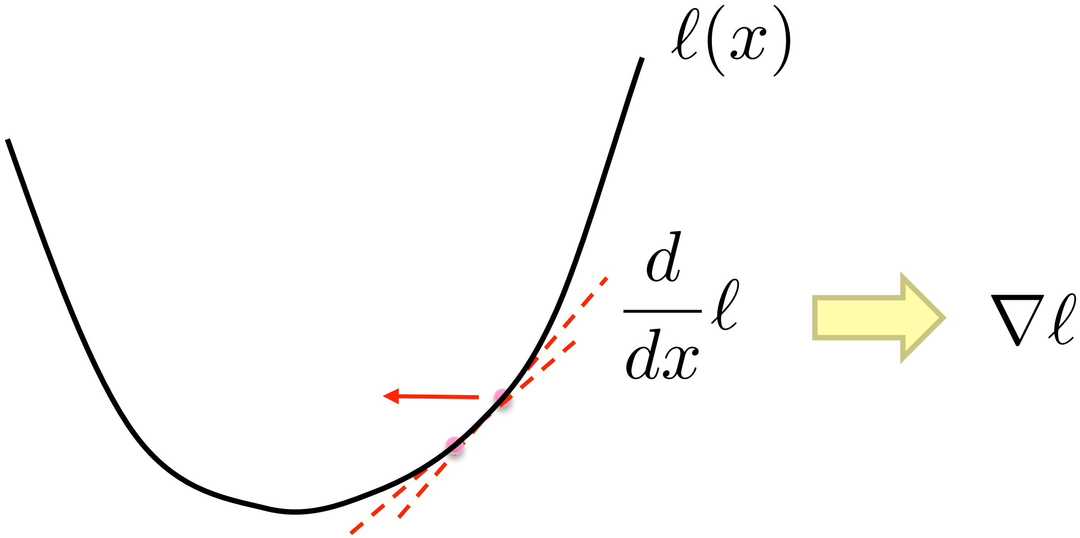
- Lots of details:
  - Figuring out the step size
  - Getting stuck in local minima
  - Convergence rate
  - …

Gradient Descent

Repeat until convergence:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \frac{1}{n} \sum_{i=0}^{n} \nabla \ell(f(\mathrm{x}_i; \theta^{(t)}), y_i)$$

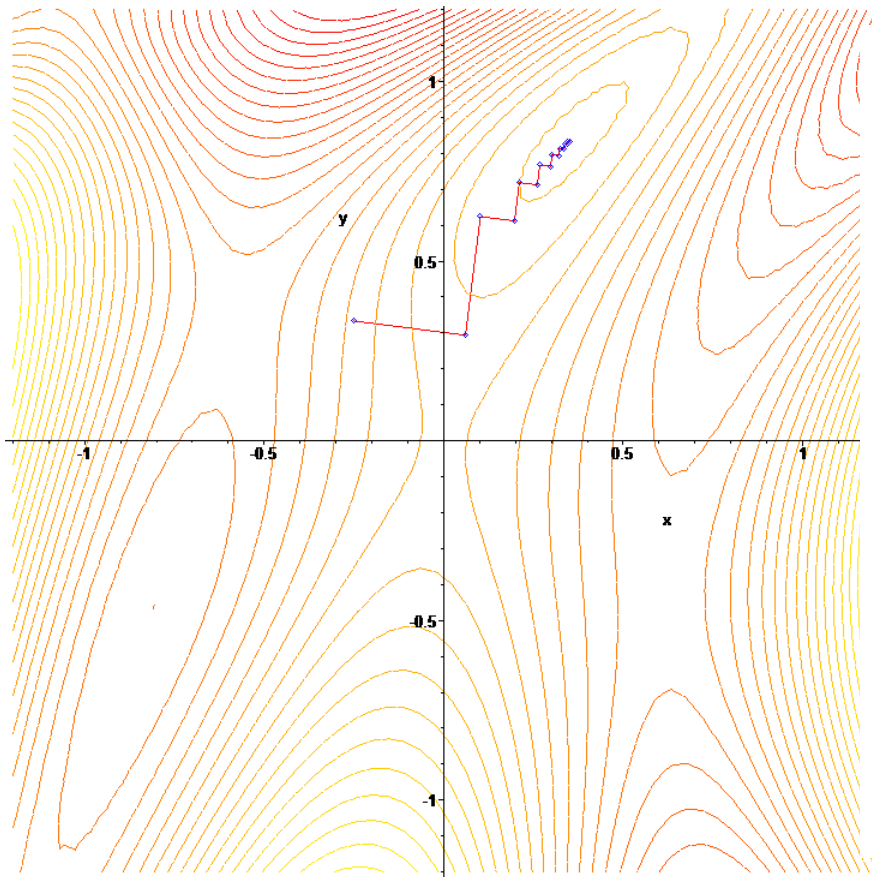
Intuition behind the math...



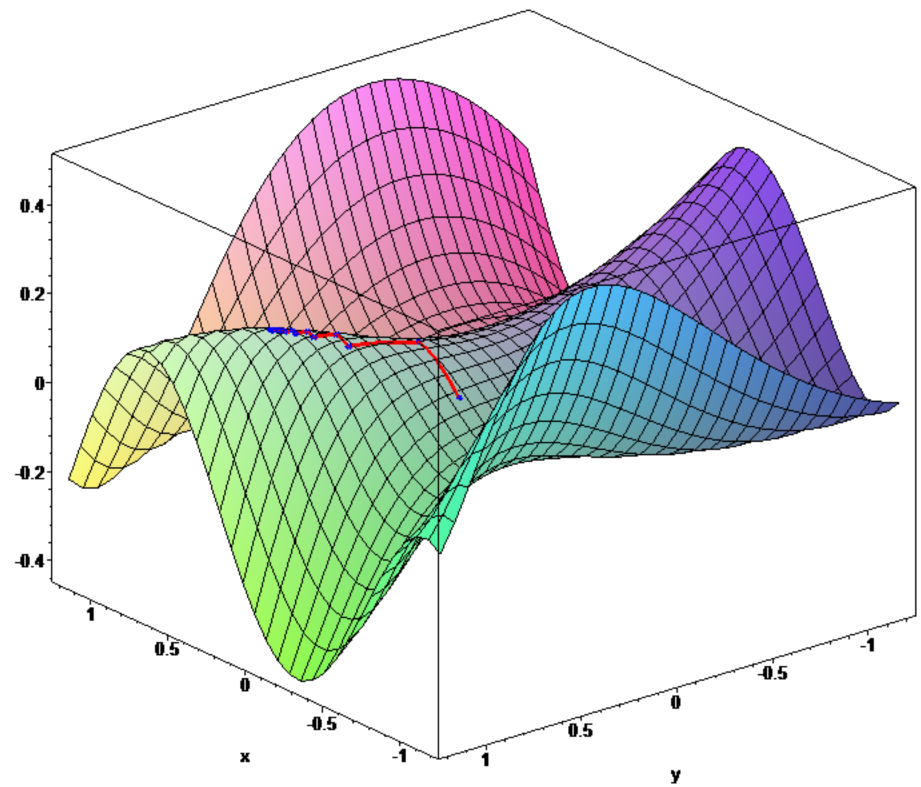
$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \frac{1}{n} \sum_{i=0}^{n} \nabla \ell(f(\mathrm{x}_i; \theta^{(t)}), y_i)$$

New weights Old weights

Update based on gradient



y
1
0.5
-0.5
-1
-1
-0.5
0.5
1
x



0.4
0.2
0
-0.2
-0.4
1
0.5
0
-0.5
-1
x
-1
-0.5
0
0.5
1
y

Gradient Descent

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \frac{1}{n} \sum_{i=0}^{n} \nabla \ell(f(\mathrm{x}_i; \theta^{(t)}), y_i)$$

Source: Wikipedia (Hills)

Lots More Details...

- Gradient descent is a "first order" optimization technique
  - Often, slow convergence
  - Conjugate techniques accelerate convergence
- Newton and quasi-Newton methods:
  - Intuition: Taylor expansion

$$f(x + \Delta x) = f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$$

  - Requires the Hessian (square matrix of second order partial derivatives): impractical to fully compute

Logistic Regression

Source: Wikipedia (Hammer)

Logistic Regression: Preliminaries

- Given $D = \{(\mathrm{x}_i, y_i)\}_i^n$

$$\mathrm{x}_i = [x_1, x_2, x_3, \ldots, x_d]$$

$$y \in \{0, 1\}$$

- Let's define:

$$f(\mathrm{x}; \mathrm{w}) : \mathbb{R}^d \to \{0, 1\}$$

$$f(\mathrm{x}; \mathrm{w}) = \begin{cases} 1 \text{ if } \mathrm{w} \cdot \mathrm{x} \geq t \\ 0 \text{ if } \mathrm{w} \cdot \mathrm{x} < t \end{cases}$$

- Interpretation:

$$\ln \left[\frac{\Pr(y = 1|\mathrm{x})}{\Pr(y = 0|\mathrm{x})} \right] = \mathrm{w} \cdot \mathrm{x}$$

$$\ln \left[\frac{\Pr(y = 1|\mathrm{x})}{1 - \Pr(y = 1|\mathrm{x})} \right] = \mathrm{w} \cdot \mathrm{x}$$

Relation to the Logistic Function

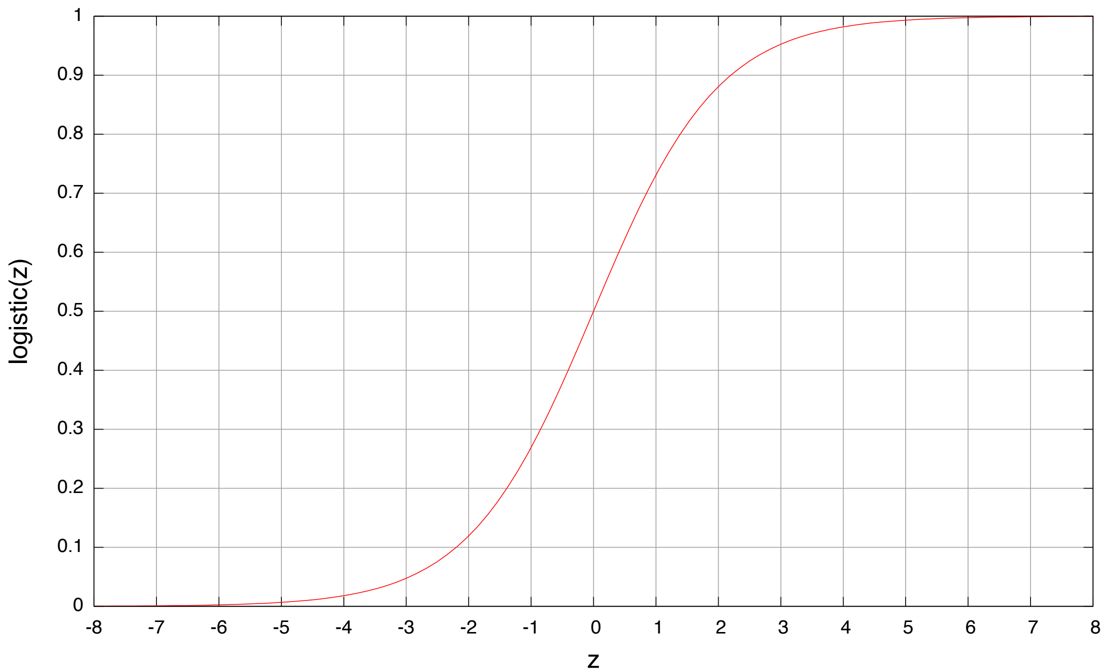
- After some algebra:

$$\Pr(y = 1|x) = \frac{e^{\mathbf{w}\cdot\mathbf{x}}}{1 + e^{\mathbf{w}\cdot\mathbf{x}}}$$

$$\Pr(y = 0|x) = \frac{1}{1 + e^{\mathbf{w}\cdot\mathbf{x}}}$$

- The logistic function:

$$f(z) = \frac{e^z}{e^z + 1}$$



Training an LR Classifier

- *Maximize* the conditional likelihood:

$$\arg\max_{\mathrm{w}} \prod_{i=1}^{n} \Pr(y_i|\mathrm{x}_i, \mathrm{w})$$

- Define the objective in terms of conditional log likelihood:

$$L(\mathrm{w}) = \sum_{i=1}^{n} \ln \Pr(y_i|\mathrm{x}_i, \mathrm{w})$$

  - We know $y \in \{0, 1\}$ so:

$$Pr(y|\mathrm{x}, \mathrm{w}) = Pr(y = 1|\mathrm{x}, \mathrm{w})^y Pr(y = 0|\mathrm{x}, \mathrm{w})^{(1-y)}$$

  - Substituting:

$$L(\mathrm{w}) = \sum_{i=1}^{n} \Big(y_i \ln \Pr(y_i = 1|\mathrm{x}_i, \mathrm{w}) + (1 - y_i) \ln \Pr(y_i = 0|\mathrm{x}_i, \mathrm{w}) \Big)$$

LR Classifier Update Rule

- Take the derivative:

$$L(\mathrm{w}) = \sum_{i=1}^{n} \Big(y_i \ln \Pr(y_i = 1 | \mathrm{x}_i, \mathrm{w}) + (1 - y_i) \ln \Pr(y_i = 0 | \mathrm{x}_i, \mathrm{w}) \Big)$$

$$\frac{\partial}{\partial \mathrm{w}} L(\mathrm{w}) = \sum_{i=0}^{n} \mathrm{x}_i \Big(y_i - \Pr(y_i = 1 | \mathrm{x}_i, \mathrm{w}) \Big)$$

- General form for update rule:

$$\mathrm{w}^{(t+1)} \leftarrow \mathrm{w}^{(t)} + \gamma^{(t)} \nabla_{\mathrm{w}} L(\mathrm{w}^{(t)})$$

$$\nabla L(\mathrm{w}) = \left[\frac{\partial L(\mathrm{w})}{\partial w_0}, \frac{\partial L(\mathrm{w})}{\partial w_1}, \ldots \frac{\partial L(\mathrm{w})}{\partial w_d} \right]$$

- Final update rule:

$$\mathrm{w}_i^{(t+1)} \leftarrow \mathrm{w}_i^{(t)} + \gamma^{(t)} \sum_{j=0}^{n} x_{j,i} \Big(y_j - \Pr(y_j = 1 | \mathrm{x}_j, \mathrm{w}^{(t)}) \Big)$$

Lots more details…

- Regularization
- Different loss functions
- …

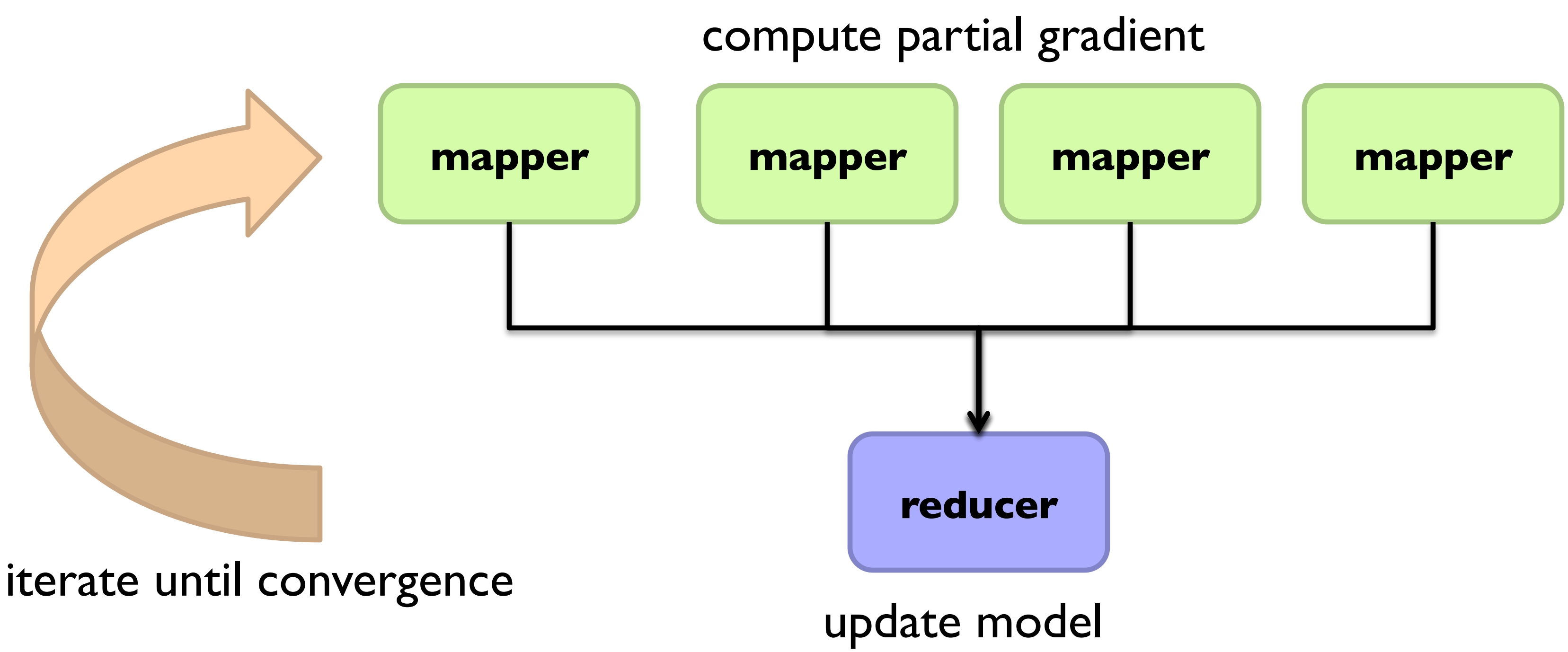
Want more details?
Take a real machine-learning course!

MapReduce Implementation

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \frac{1}{n} \sum_{i=0}^{n} \nabla \ell(f(\mathrm{x}_i; \theta^{(t)}), y_i)$$

mappers

single reducer



Shortcomings

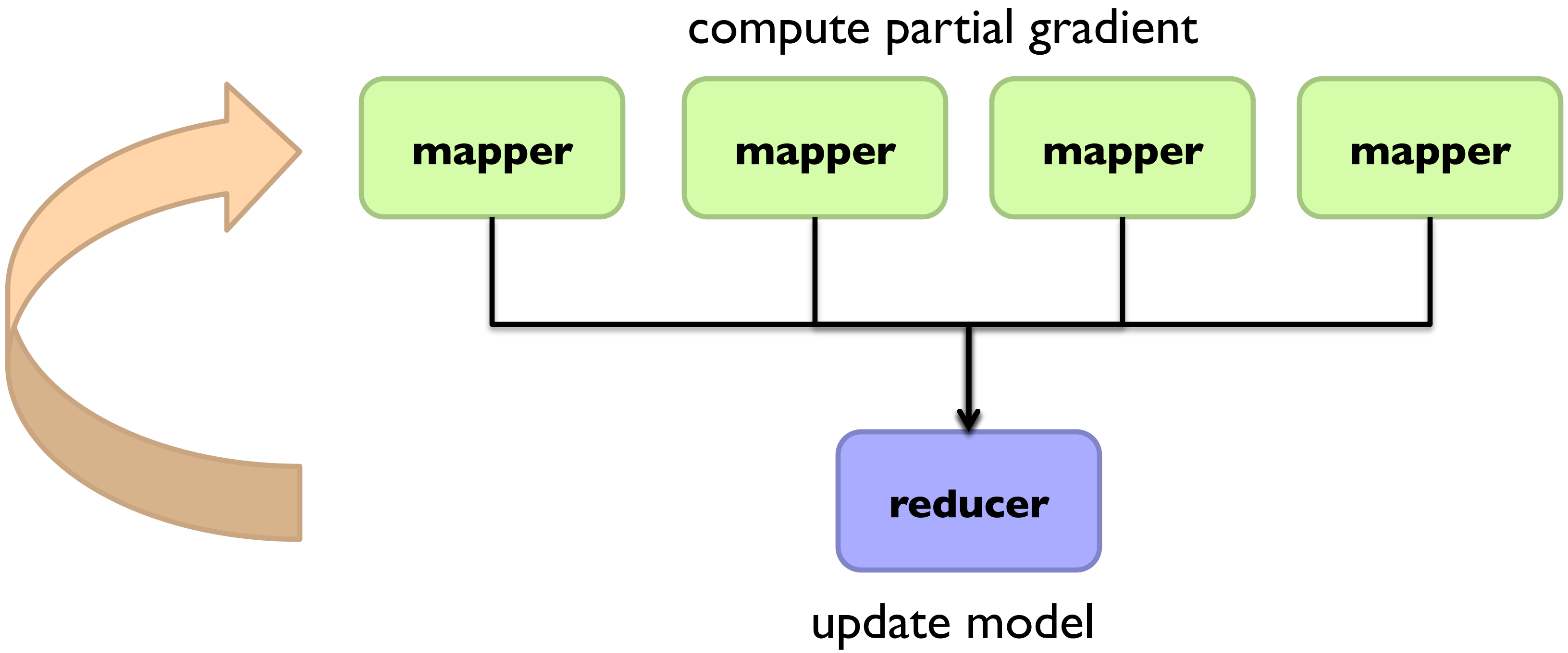
- Hadoop is bad at iterative algorithms
  - High job startup costs
  - Awkward to retain state across iterations
- High sensitivity to skew
  - Iteration speed bounded by slowest task
- Potentially poor cluster utilization
  - Must shuffle all data to a single reducer
- Some possible tradeoffs
  - Number of iterations vs. complexity of computation per iteration
  - E.g., L-BFGS: faster convergence, but more to compute

Spark Implementation

```
val points = spark.textFile(...).map(parsePoint).persist()

var w = // random initial vector
for (i <- 1 to ITERATIONS) {
  val gradient = points.map{ p =>
    p.x * (1/(1+exp(-p.y*(w dot p.x)))-1)*p.y
  }.reduce((a,b) => a+b)
  w -= gradient
}
```

What's the difference?



Source: Wikipedia (Japanese rock garden)