



UNIVERSITY OF
WATERLOO

Data-Intensive Distributed Computing

CS 451/651 431/631 (Winter 2018)

Part 6: Data Mining (1/4)

February 27, 2018

Jimmy Lin

David R. Cheriton School of Computer Science

University of Waterloo

These slides are available at <http://lintool.github.io/bigdata-2018w/>



This work is licensed under a Creative Commons Attribution-NonCommercial-Share Alike 3.0 United States
See <http://creativecommons.org/licenses/by-nc-sa/3.0/us/> for details

Structure of the Course

Analyzing Text

Analyzing Graphs

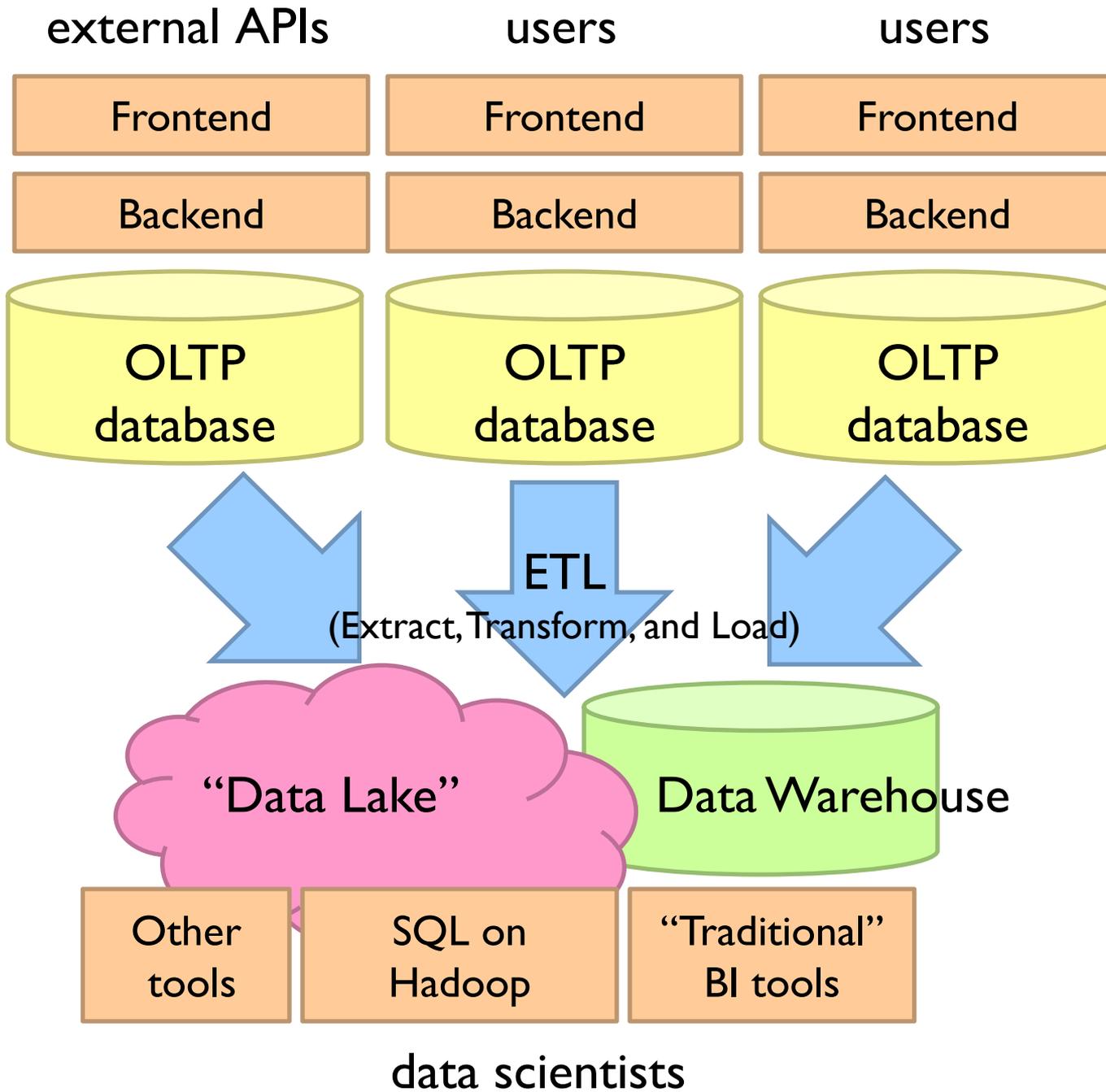
Analyzing
Relational Data

Data Mining

“Core” framework features
and algorithm design

Learn new buzzwords!

Descriptive vs. Predictive Analytics



external APIs

users

users

Frontend

Frontend

Frontend

Backend

Backend

Backend

OLTP
database

OLTP
database

OLTP
database

ETL

(Extract, Transform, and Load)

“Data Lake”

Data Warehouse

Other
tools

SQL on
Hadoop

“Traditional”
BI tools

data scientists

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



Applications

Spam detection

Sentiment analysis

Content (e.g., topic) classification

Link prediction

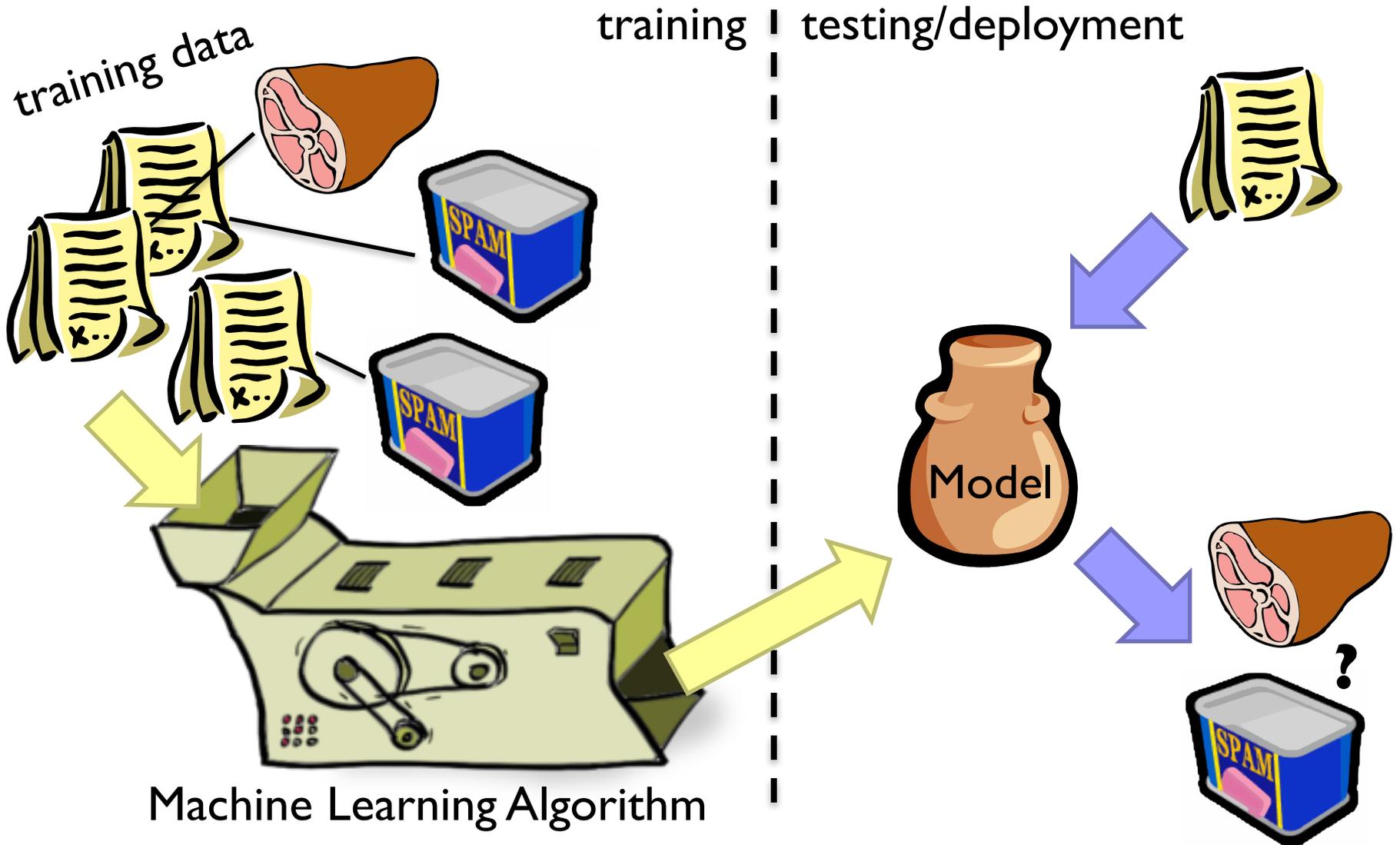
Document ranking

Object recognition

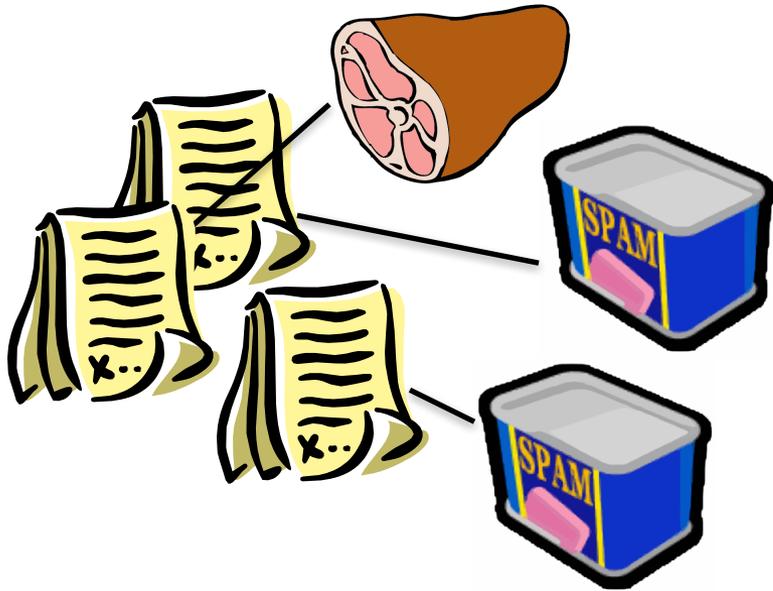
Fraud detection

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

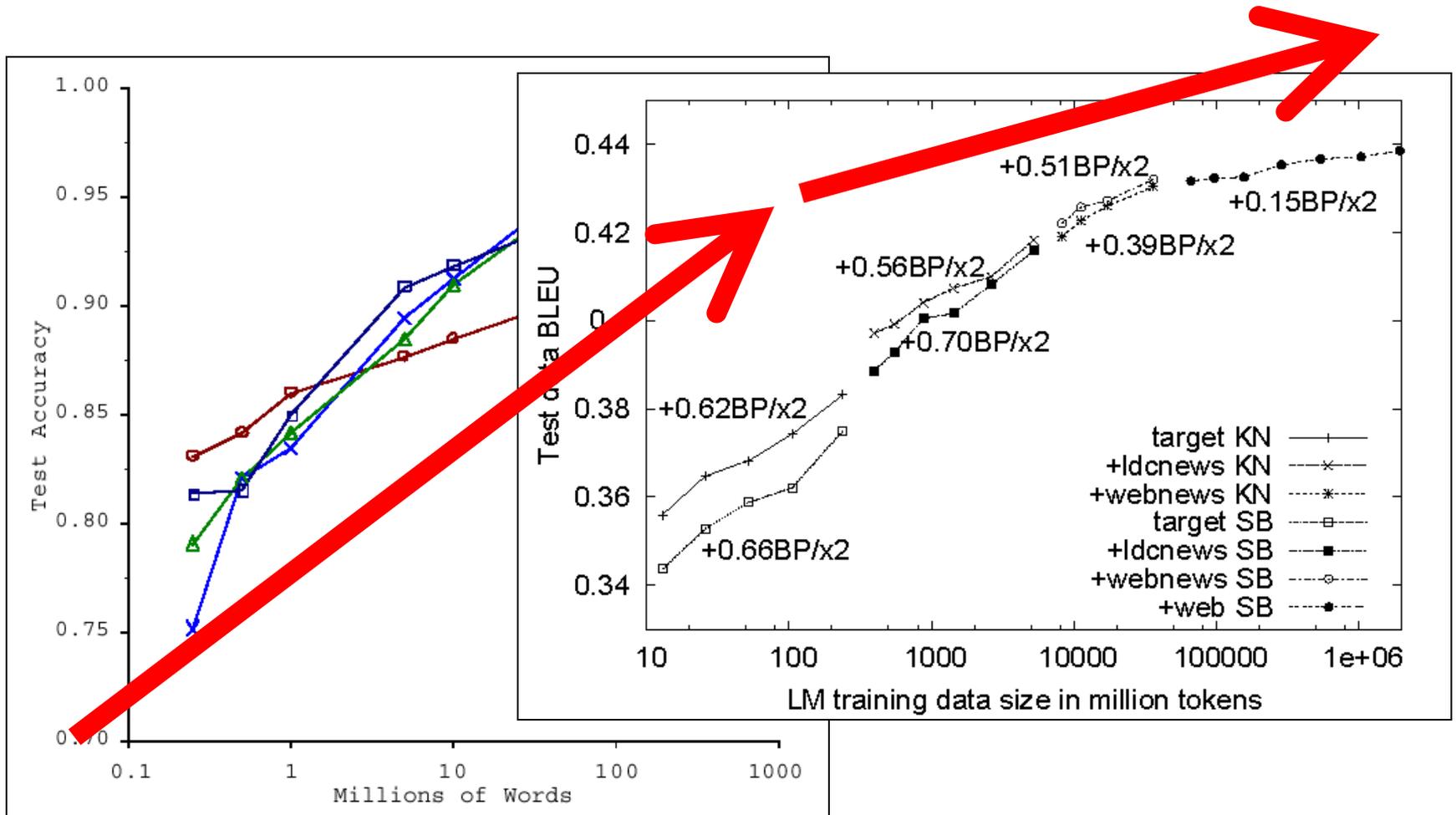
Optimization

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

logistic regression, naive
Bayes, SVM, random
forests, perceptrons,
neural networks, etc.

What “matters” the most?

No data like more data!



Limits of Supervised Classification?

Why is this a big data problem?

Isn't gathering labels a serious bottleneck?

Solutions

Crowdsourcing

Bootstrapping, semi-supervised techniques

Exploiting user behavior logs

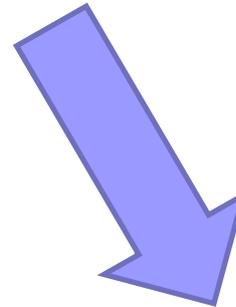
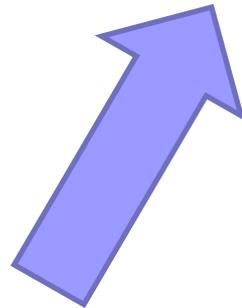
The virtuous cycle of data-driven products

Virtuous Product Cycle

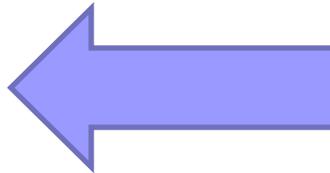
a useful service



(hopefully)



transform insights
into action



analyze user behavior
to extract insights

Google. Facebook. Twitter. Amazon. Uber.

data products

data science

What's the deal with neural networks?

Data

Features

Model

Optimization

Supervised *Binary* Classification

Restrict output label to be *binary*

Yes/No

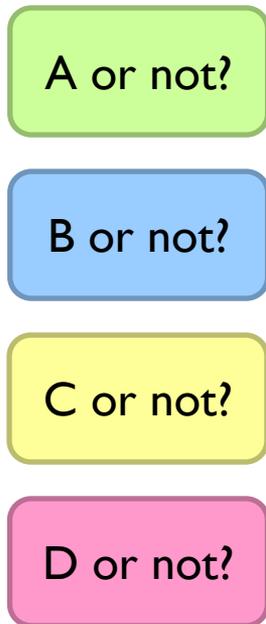
1/0

Binary classifiers form primitive
building blocks for multi-class problems...

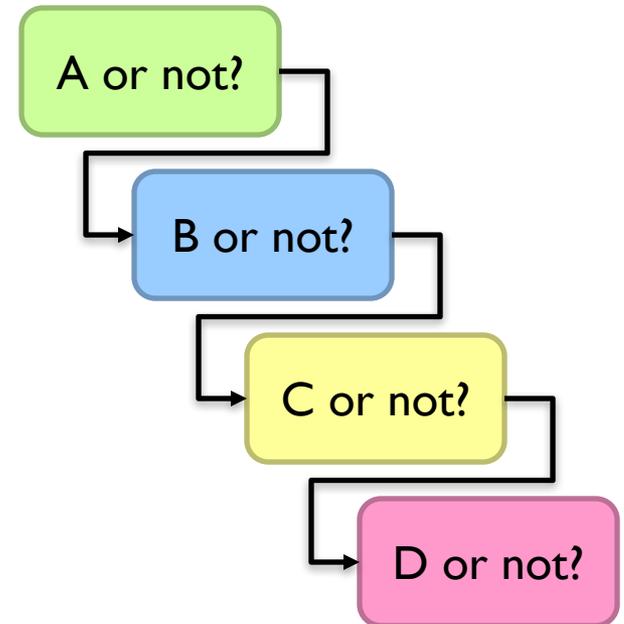
Binary Classifiers as Building Blocks

Example: four-way classification

One vs. rest classifiers

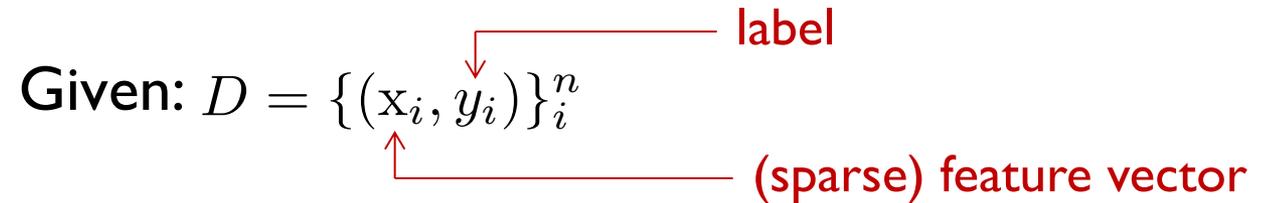


Classifier cascades



The Task

Given: $D = \{(x_i, y_i)\}_i^n$



label

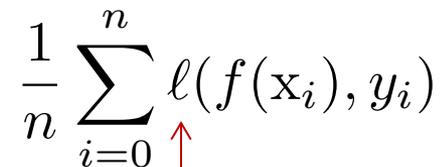
(sparse) feature vector

$$x_i = [x_1, x_2, x_3, \dots, x_d]$$

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

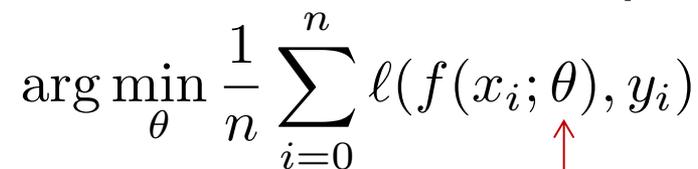
Induce: $f : X \rightarrow Y$

Such that loss is minimized

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


loss function

Typically, we 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:

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

Compute gradient:

“Points” to fastest increasing “direction”

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

So, at any point: *

$$b = a - \gamma \nabla L(a)$$

$$L(a) \geq L(b)$$

Gradient Descent: Iterative Update

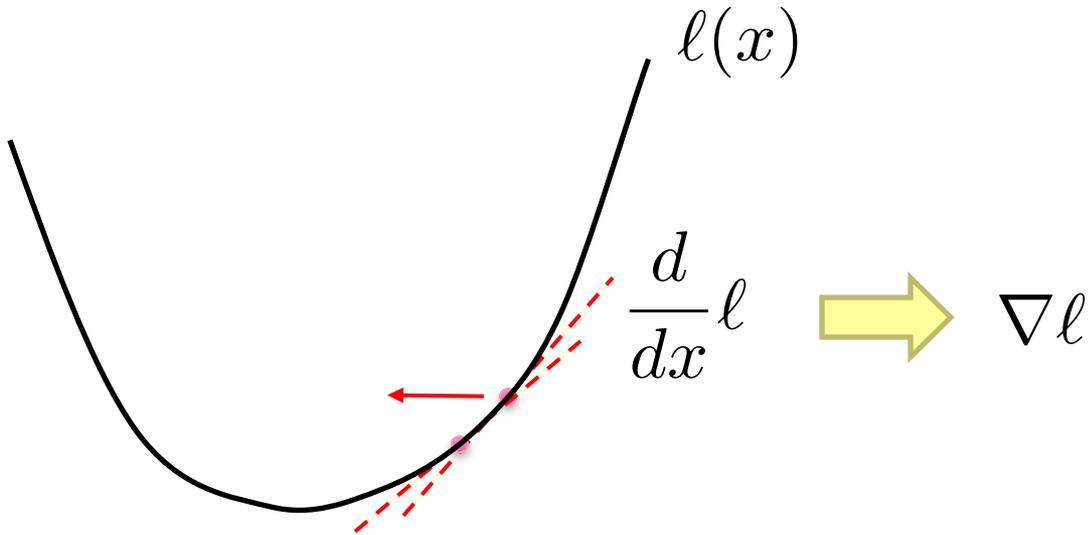
Start at an arbitrary point, iteratively update:

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

We have:

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

Intuition behind the math...



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

New weights Old weights Update based on gradient

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:

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

Lots of details:

Figuring out the step size
Getting stuck in local minima
Convergence rate

...

Gradient Descent

Repeat until convergence:

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

Note, sometimes formulated as *ascent* but entirely equivalent

Gradient Descent

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

Even More Details...

Gradient descent is a “first order” optimization technique

Often, slow 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

A claw hammer with a wooden handle and a metal head, resting on a wooden workbench. The hammer is positioned diagonally from the bottom left towards the top right. The workbench has several small metal fasteners visible. The text "Logistic Regression" is overlaid in white on the lower right portion of the hammer's handle.

Logistic Regression

Logistic Regression: Preliminaries

Given: $D = \{(x_i, y_i)\}_i^n$
 $x_i = [x_1, x_2, x_3, \dots, x_d]$
 $y \in \{0, 1\}$

Define: $f(\mathbf{x}; \mathbf{w}) : \mathbb{R}^d \rightarrow \{0, 1\}$
 $f(\mathbf{x}; \mathbf{w}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} \geq t \\ 0 & \text{if } \mathbf{w} \cdot \mathbf{x} < t \end{cases}$

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

Relation to the Logistic Function

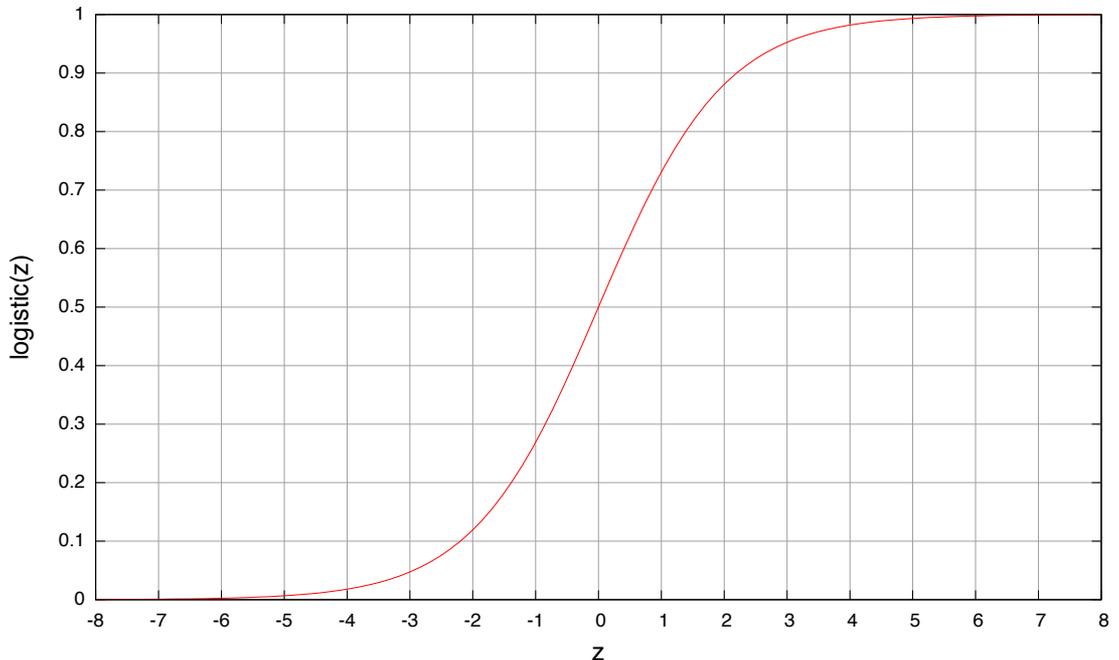
After some algebra:

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

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

The logistic function:

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



Training an LR Classifier

Maximize the conditional likelihood: $\arg \max_{\mathbf{w}} \prod_{i=1}^n \Pr(y_i | \mathbf{x}_i, \mathbf{w})$

Define the objective in terms of conditional *log* likelihood: $L(\mathbf{w}) = \sum_{i=1}^n \ln \Pr(y_i | \mathbf{x}_i, \mathbf{w})$

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

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

Substituting:

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

LR Classifier Update Rule

Take the derivative:

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

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

General form of update rule:

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

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

Final update rule:

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \gamma^{(t)} \sum_{j=0}^n x_{j,i} \left(y_j - \Pr(y_j = 1 | \mathbf{x}_j, \mathbf{w}^{(t)}) \right)$$

Lots more details...

Regularization
Different loss functions
...

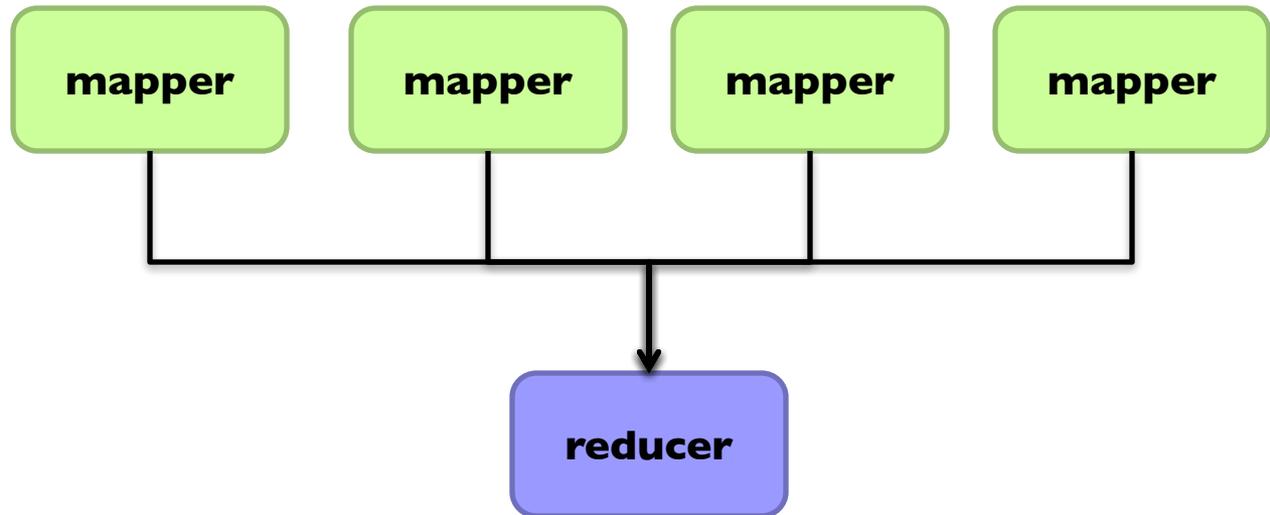
Want more details?
Take a real machine-learning course!

MapReduce Implementation

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

single reducer

compute partial gradient



iterate until convergence

update model

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

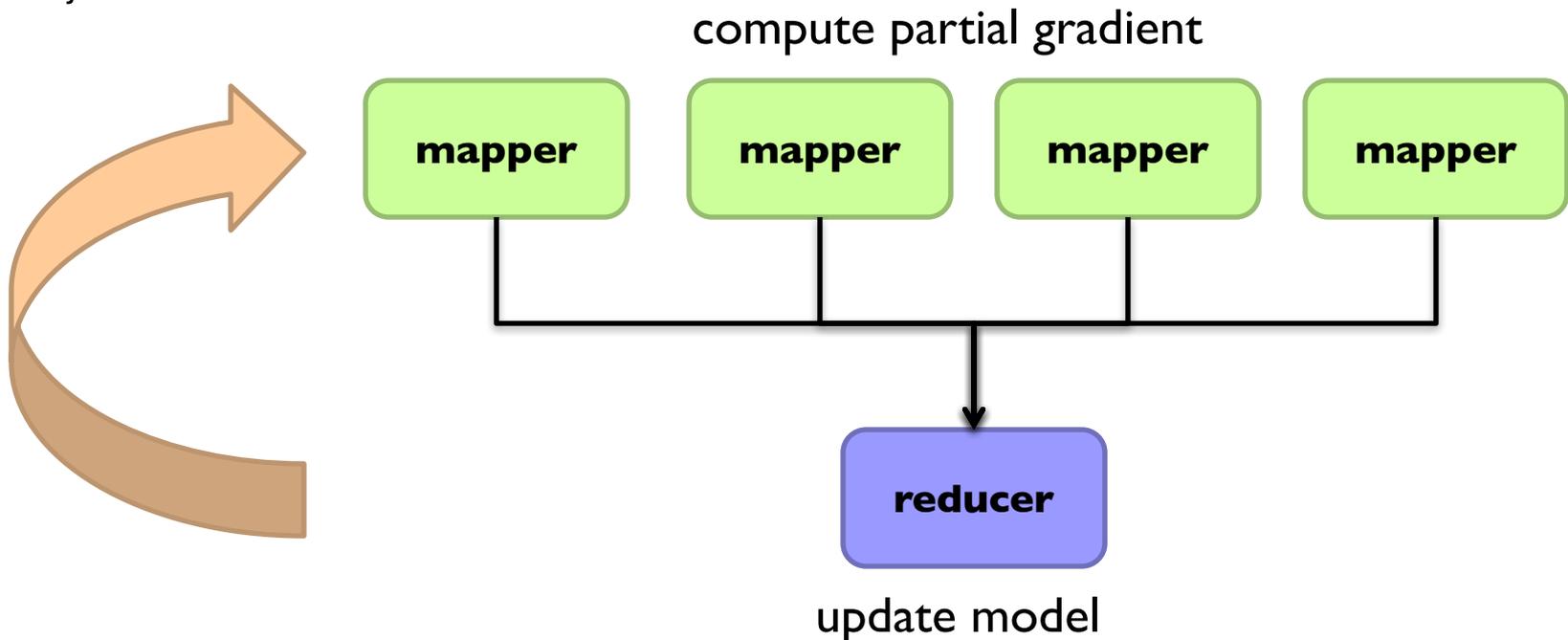
E.g., L-BFGS: faster convergence, but more to compute

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?



A traditional Japanese rock garden (karesansui) featuring a gravel path with raked patterns, several large dark rocks, and a small stream flowing through the center. The garden is surrounded by lush greenery, including moss-covered bushes and trees, with a traditional Japanese building visible in the background.

Questions?