Data-Intensive Distributed Computing
CS 431/631 451/651 (Winter 2019)

Part 6: Data Mining (1/4)
October 29, 2019

Ali Abedi

These slides are available at https://www.student.cs.uwaterloo.ca/~cs451

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

“Core” framework features and algorithm design

Analyzing Text
Analyzing Graphs
Analyzing Relational Data
Data Mining
Descriptive vs. Predictive Analytics
“Data Lake”

Data Warehouse

Other tools

SQL on Hadoop

“Traditional” BI tools

data scientists

external APIs

users

users

Frontend

Backend

Frontend

Backend

Frontend

Backend

OLTP database

OLTP database

OLTP database

ETL

(Extract, Transform, and Load)
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!
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

Training data

Machine Learning Algorithm

Training

Testing/deployment

Model
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.

Who comes up with the features? How?
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 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


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

A or not?
B or not?
C or not?
D or not?

Classifier cascades

A or not?
B or not?
C or not?
D or not?
The Task

Given: $D = \{(x_i, y_i)\}_{i=1}^{n}$

(sparse) feature vector

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

(label)

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

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

Compute gradient:
“Points” to fastest increasing “direction”
\[
\nabla L(\theta) = \left[ \frac{\partial L(\theta)}{\partial w_0}, \frac{\partial L(\theta)}{\partial w_1}, \ldots \frac{\partial L(\theta)}{\partial w_d} \right]
\]

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:

$$L(\theta^{(0)}) \geq L(\theta^{(1)}) \geq L(\theta^{(2)}) \ldots$$
Intuition behind the math...

\[ \ell(x) \]

\[ \frac{d}{dx} \ell \quad \rightarrow \quad \nabla \ell \]

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

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:

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

Lots of details:

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

...
Repeat until convergence:

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

Gradient Descent

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
Logistic Regression

Source: Wikipedia (Hammer)
Logistic Regression: Preliminaries

Given:

\[ D = \{(x_i, y_i)\}_{i=1}^{n} \]

\[ x_i = [x_1, x_2, x_3, \ldots, x_d] \]

\[ y \in \{0, 1\} \]

Define:

\[ f(x; w) : \mathbb{R}^d \rightarrow \{0, 1\} \]

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

Interpretation:

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

\[ \ln \left[ \frac{\Pr(y = 1|x)}{1 - \Pr(y = 1|x)} \right] = w \cdot x \]
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: 
$$\arg \max_w \prod_{i=1}^n \Pr(y_i|x_i, w)$$

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

We know: \( y \in \{0, 1\} \)

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

Substituting:
$$L(w) = \sum_{i=1}^n \left( y_i \ln \Pr(y_i = 1|x_i, w) + (1 - y_i) \ln \Pr(y_i = 0|x_i, w) \right)$$
LR Classifier Update Rule

Take the derivative:

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

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

General form of update rule:

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

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

Final update rule:

$$w_{i}^{(t+1)} \leftarrow w_{i}^{(t)} + \gamma^{(t)} \sum_{j=0}^{n} x_{j,i} \left( y_j - \Pr(y_j = 1|x_j, w^{(t)}) \right)$$
Lots more details...

Regularization
Different loss functions

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

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

- 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
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
}

Spark Implementation

What’s the difference?