Data-Intensive Distributed Computing
CS 431/631 451/651 (Winter 2021)

Part 7: Data Mining (1/4)

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
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!
Applications

What is the science of classifying living things?

Racism
Supervised Machine Learning
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

Data matters the most. If we throw a lot of data at almost any algorithm it performs good.
We have seen these examples before. For example, stupid backoff outperforms other algorithms when it's trained on a lot of 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
Supervised \textit{Binary} Classification

Restrict output label to be \textit{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: \( D = \{(x_i, y_i)\}_{i=1}^{n} \)  
\( x_i = [x_1, x_2, x_3, \ldots, x_d] \)  
\( y \in \{0, 1\} \)

Induce: \( f : X \rightarrow Y \)  
Such that loss is minimized

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

Typically, we consider functions of a parametric form:

\[
\arg \min_{\theta} \frac{1}{n} \sum_{i=0}^{n} \ell(f(x_i; \theta), y_i)
\]
Key insight: machine learning as an optimization problem!
Because the closed form solutions generally not possible ...
Gradient Descent: Preliminaries

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 \quad Old weights \quad Update based on gradient
Linear Regression + Gradient Descent
Gradient Descent: Preliminaries

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

Rewrite:

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:
\[
L(\theta^{(0)}) \geq L(\theta^{(1)}) \geq L(\theta^{(2)}) \ldots
\]
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:

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

Note, sometimes formulated as ascent but entirely equivalent.
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: Preliminaries

Comparing Linear Regression and Logistic Regression:

- **Linear Regression**:
  - Output values range from -∞ to +∞.
  - Model predicts continuous outcomes.
  - Graph shows a linear relationship between the dependent variable and independent variables.

- **Logistic Regression**:
  - Output is restricted to values between 0 and 1.
  - Model predicts binary outcomes (e.g., 0 or 1, presence or absence).
  - Graph shows a curve that transforms continuous input into a probability.

The diagrams illustrate these differences, with the left side showing linear regression and the right side showing logistic regression.
Logistic Regression: Preliminaries

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

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:

\[
\begin{align*}
\Pr(y = 1|x) &= \frac{e^{w\cdot x}}{1 + e^{w\cdot x}} \\
\Pr(y = 0|x) &= \frac{1}{1 + e^{w\cdot x}}
\end{align*}
\]

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:

\[
w^{(t+1)} = w^{(t)} + \gamma^{(t)} \nabla_w L(w^{(t)})
\]

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

Final update rule:

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

Regularization
Different loss functions
...

Want more details?
Take a real machine-learning course!
\[
 w_i^{(t+1)} \leftarrow w_i^{(t)} + \gamma^{(t)} \sum_{j=0}^{n} x_{j,i} \left( y_j - Pr(y_j = 1|x_j, w^{(t)}) \right)
\]
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

mapper mapper mapper mapper

iterate until convergence

update model

32
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
}