Logistics

- One TA: Owen Xie
- Office hours on course website
  www.cs.rpi.edu/~gittera/teaching/fall2020/mlandopt.htm
- We will Piazza for general discussions
- Grading breakdown:
  - HWs (50%) - 11 HWs
  - Projects (35%) - more details in a couple of weeks
- Weekly Participation (15%)
What is this class?

A survey of topics that you may see separately (in more details) in ML, optimization, randomized algorithms, and Deep Learning courses.

Why? ML = learning from data

- SOTA ML models are usually deep-learned
- Optimization
- "Big data"
- Use randomization for efficiency
Lecture breakdown

- 4 basic probability theory & how to phrase ML problems as optimization problems (ERM framework)

- 9 general techniques for solving these optimization problems (convex optimization)

- 5 randomized algorithms for some specific ML applications

- 9 deep learning (specific problem classes & architectures)

- 1 topics we didn't cover (teaser & motivation to research)
Resources

- Introduction to Applied Linear Algebra
  Boyd & Vandenberghe

- Deep Learning w/ Python. Chollet.
Examples

unsupervised:
  k-means clustering
  spectral clustering
  k-nn
  language modeling

supervised:
  SVM
  image recognition
  perception
  naive Bayes
  linear regression
  logistic regression
At a high-level ML consist of:

- formulate a task

* - collect appropriate data

- choosing an appropriate model for your problem data

- formulate an appropriate optimization problem to fit that model

*** - solving the opt prob, potentially after modifying it to make it more tractable
Ex: k-nearest neighbors (for 2-class classification problem)

Data: Given \( X = \{ (x_i, y_i) \}_{i=1}^n \) where \( x_i \in \mathbb{R}^d \)
are features and \( y_i \in \{1, -1\} \)

Prob: We want to learn \( y \) as a function of \( x \):
\[ y = f(x) \]

Think: We want to determine whether a person will default on their loan.

Model: We assume that inputs that share similar features should fall in similar classes.
let \( N_k(x) \) = k nearest neighbors of \( x \) in our training data

predict

\[
y = f(x) = \text{sign}\left( \sum_{x_j \in N_k(x)} y_j \right)
\]

--- Optimization

Given a new point \( x \), we need to find \( W_k(x) \), then sum the corresponding \( y_j \), and take the sign.

What is the cost of this operation?
\[ y = \text{sign}(1 + 1) = 1 \]

\[ x \]

\[ 1 \]

\[ 1 \]

\[ 1 \]

\[ k = 2 \]

\[ y = \text{sign}(-1 - 1) = -1 \]

\[ \text{sign}(x) = \begin{cases} 
1 & \text{if } x > 0 \\
-1 & \text{if } x < 0 \\
0 & \text{if } x = 0 
\end{cases} \]
\[ x = \text{Cost of k-m in } O(nd + n\log n + K) \]

determining \( \| x_i - x \|_2^2 \)

\[ \frac{1}{2} \sum_{j=1}^{d} (x_i)_j^2 - x_j^2 \]

is \( O(d) \)

dimensions do this for \( n \) points

\( O(nd) \) to compute distances

+ \( O(n\log n) \) to find \( k \) smallest distances

+ \( O(k) \) to sum and take the sign
When $n$ and $d$ are too large (e.g. $n \sim O(10^6)$ and $d \sim O(10^3)$) we have $nd \sim O(10^9)$, so this is too expensive to use directly.

How to reduce the cost? We use **approximate nearest neighbors**: project the points from $\mathbb{R}^d$ down to a lower dimensional space $\mathbb{R}^c$ and find the nearest neighbors there.

In practice: fix a matrix $P \in \mathbb{R}^{c \times d}$, then compare $\ell_2$ instead of $\ell_1$: for $x_i \in \mathbb{R}^d$, choose $p_i \in \mathbb{R}^c$ such that $\left\| P_{p_i} - P x_i \right\|_2$ is minimized.
This is only useful if $P$ is such that

$$
\| P(x_i - x) \|_2 \approx \| x_i - x \|_2
$$

and can be found efficiently, then $N_k(x)$ is preserved.

Algorithm

1) precompute $P$

2) precompute and store $\sum P x_i \beta_i$, $i=1$ offline

3) given a new $x$, compute $P x$ and use $k-$nn on $\sum P x_i \beta_i$, and $P x$
Costs

offline costs:

$O(d_c + n d_c)$

computing $P$

$\begin{pmatrix} x_1^T \\ \vdots \\ x_n^T \end{pmatrix}$

$= \begin{pmatrix} n \\ \vdots \\ d \end{pmatrix}$

$P r$

$P^T d x c$

online cost:

$O(d_c + n c + n \log n + k)$

computing $P x$
Takeaway:

\(k\)-nn: \(O(nd)\)

\(k\)-nearest approx neighbors: \(O(nc + dc)\)

\(n = O(10^6)\)

\(d = O(10^3)\)

\(c = O(10)\)
Ex. Binary classification via SVM

Goal: given positive & negative examples learn a linear separation boundary so that all/most positive examples fall on one side and all/most negative examples fall on the other.
model: \( y \approx \text{sign}(\langle \omega \rangle \cdot \mathbf{x} + b) \)
Optimization Problem

- we penalize the current \( \mathbf{w} \) if

\[
y_i \langle \mathbf{w}, \mathbf{x}_i \rangle < 0
\]

for any training pair \((\mathbf{x}_i, y_i)\)

- in fact, we want the model to be confident

\[
y_i \langle \mathbf{w}, \mathbf{x}_i \rangle > 1
\]

for all training pair \((\mathbf{x}_i, y_i)\)

- to avoid simply achieving confidence by letting \( \|\mathbf{w}\|_2 \to \infty \) (this is an unstable model!),

we want \( \|\mathbf{w}\|_2 \) to be small.
This leads to two possible optimization problems:

\[ \omega_* = \arg\min_{\omega \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \max \left\{ 0, 1 - y_i \langle \omega, x_i \rangle \right\} \]

-called the hinge loss

\[ \phi(t) = \max\{0, 1-t\} \]

(constrained optimization)
(penalized optimization)

\[
\omega_* = \arg\min_{\omega \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \phi(y_i \langle \omega, x_i \rangle) + \frac{\lambda}{2} \| \omega \|^2
\]

In ML we prefer penalized optimization