5 Machine Learning Abstractions and Numerical Optimization

ML ABSTRACTIONS [some meta comments on machine learning]

[When you write a large computer program, you break it down into subroutines and modules. Many of you know from experience that you need to have the discipline to impose strong abstraction barriers between different modules, or your program will become so complex you can no longer manage nor maintain it.]

[When you learn a new subject, it helps to have mental abstraction barriers, too, so you know when you can replace one approach with a different approach. I want to give you four levels of abstraction that can help you think about machine learning. It's important to make mental distinctions between these four things, and the code you write should have modules that reflect these distinctions as well.]

APPLICATION/DATA

data labeled (classified) or not?
yes: labels categorical (classification) or quantitative (regression)?
no: similarity (clustering) or positioning (dimensionality reduction)?

MODEL [what kinds of hypotheses are permitted?]

e.g.:
– decision fns: linear, polynomial, logistic, neural net, . . .
– nearest neighbors, decision trees
– features
– low vs. high capacity (affects overfitting, underfitting, inference)

OPTIMIZATION PROBLEM

– variables, objective fn, constraints
e.g., unconstrained, convex program, least squares, PCA

OPTIMIZATION ALGORITHM

e.g., gradient descent, simplex, SVD

[In this course, we focus primarily on the middle two levels. As a data scientist, you might be given an application, and your challenge is to turn it into an optimization problem that we know how to solve. We will talk a bit about optimization algorithms, but usually you'll use an optimization code that's faster and more robust than what you would write yourself.]

[The second level, the model, has a huge effect on the success of your learning algorithm. Sometimes you get a big improvement by tailoring the model or its features to fit the structure of your specific data. The model also has a big effect on whether you overfit or underfit. And if you want a model that you can interpret so you can do *inference*, the model has to have a simple structure. Lastly, you have to pick a model that leads to an optimization problem that can be solved. Some optimization problems are just too hard.]

[It's important to understand that when you change something in one level of this diagram, you probably have to change all the levels underneath it. If you switch your model from a linear classifier to a neural net, your optimization problem changes, and your optimization algorithm changes too.]

[Not all machine learning methods fit this four-level decomposition. Nevertheless, for everything you learn in this class, think about where it fits in this hierarchy. If you don't distinguish which math is part of the model and which math is part of the optimization algorithm, this course will be very confusing for you.]

OPTIMIZATION PROBLEMS

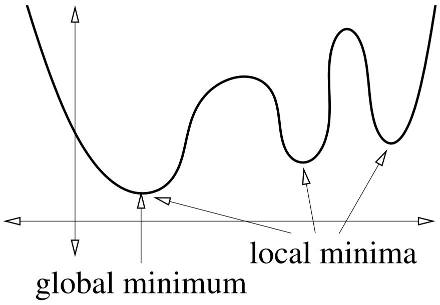
[I want to familiarize you with some types of optimization problems that can be solved reliably and efficiently, and the names of some of the optimization algorithms used to solve them. An important skill for you to develop is to be able to go from an application to a well-defined optimization problem. That skill depends on your ability to *recognize* well-studied types of optimization problems.]

Unconstrained

Goal: Find w that minimizes (or maximizes) a continuous objective fn $f(w)$.

f is smooth if its gradient is continuous too.

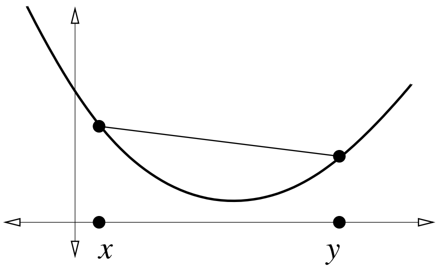
A global minimum of f is a value w such that $f(w) \leq f(v)$ for every v.
A local minimum ” ” ” ” ” ” ” ” ” ” ”
for every v in a tiny ball centered at w.
[In other words, you cannot walk downhill from w.]



[Draw this by hand. minima.pdf]

Usually, finding a local minimum is easy;
finding the global minimum is hard. [or impossible]

Exception: A function is convex if for every $x, y \in \mathbb{R}^d$,
the line segment connecting $(x, f(x))$ to $(y, f(y))$ does not go below $f(\cdot)$.



[Draw this by hand. convex.pdf]

E.g., perceptron risk fn is convex and nonsmooth.

[When you sum together convex functions, you always get a convex function. The perceptron risk function is a sum of convex loss functions.]

A [continuous] convex function [on a closed, convex domain] has either

- no minimum (goes to $-\infty$), or
- just one local minimum, or
- a connected set of local minima that are all global minima with equal f.

[The perceptron risk function has the last of these three.]
[In the last two cases, if you walk downhill, you eventually reach a global minimum.]
[However, there are many applications where you don't have a convex objective function, and your machine learning algorithm has to settle for finding a local minimum. For example, neural nets try to optimize an objective function that has *lots* of local minima; they often don't find a global minimum.]

Algs for smooth f:

- Gradient descent
  - blind [with learning rate] repeat: $w \leftarrow w - \epsilon \nabla f(w)$
  - stochastic (blind) [trains on one point per iteration, or a small batch]
  - with line search
- Newton's method (needs Hessian matrix of f)
- Nonlinear conjugate gradient [uses the secant or Newton–Raphson line search methods]

Algs for nonsmooth f:

- Gradient descent
- BFGS [Broyden–Fletcher–Goldfarb–Shanno]

These algs find a local minimum. [They don't reliably find a global minimum, because that's very hard.]

[If you're optimizing over a d-dimensional space, the Hessian matrix is a $d \times d$ matrix and it's usually dense, so most methods that use the Hessian are computationally infeasible when d is large.]

line search: finds a local minimum along the search direction by solving an optimization problem in 1D. [...instead of using a blind step size like the perceptron algorithm does. Solving a 1D problem is much easier than solving a higher-dimensional one.]

- secant method (smooth only)
- Newton–Raphson (smooth only; may need Hessian matrix)
- direct line search (e.g., golden section search; mostly for nonsmooth)

[Neural nets are unconstrained optimization problems with many, many local minima. They sometimes benefit from line searches or second-order optimization algorithms, but when the input data set is very large, researchers often favor the dumb, blind, stochastic versions of gradient descent.]

Constrained Optimization (smooth equality constraints)

Goal: Find w that minimizes (or maximizes) $f(w)$
subject to $g(w) = 0$ [← observe that this is an isosurface]

where g is a smooth fn [g may be a vector, encoding multiple constraints]

Alg: Use Lagrange multipliers. [to transform constrained to unconstrained optimization]

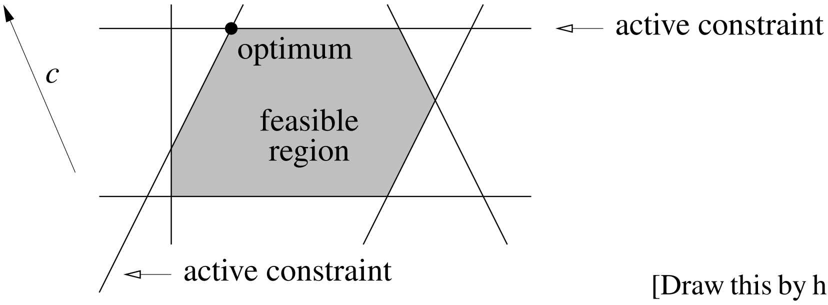
Linear Program

Linear objective fn + linear **inequality** constraints.

Goal: Find w that maximizes (or minimizes) $c \cdot w$
subject to $Aw \leq b$

where A is $n \times d$ matrix, $b \in \mathbb{R}^n$, expressing n linear constraints:
$A_i \cdot w \leq b_i, \qquad i \in [1, n]$

in w-space:



[Draw this by hand. linprog.pdf]

The set of points w that satisfy all constraints is a convex polytope called the feasible region F [shaded].
The optimum is the point in F that is furthest in the direction c. [What does convex mean?]
A point set P is convex if for every $p, q \in P$, the line segment with endpoints p, q lies entirely in P.

[What is a polytope? Just a polyhedron, generalized to higher dimensions.]

The optimum achieves equality for some constraints (but not most), called the active constraints of the optimum. [In the figure above, there are two active constraints. In an SVM, active constraints correspond to the sample points that touch or violate the slab, and they're also known as support vectors.]

[Sometimes, there is more than one optimal point. For example, in the figure above, if c pointed straight up, every point on the top horizontal edge would be optimal. The set of optimal points is always convex.]

Example: EVERY feasible point (w, α) gives a linear classifier:

Find w, α that maximizes 0
subject to $y_i(w \cdot X_i + \alpha) \geq 1$ for all $i \in [1, n]$

IMPORTANT: The data are linearly separable iff the feasible region is not the empty set.
→ Also true for maximum margin classifier (quadratic program)

Algs for linear programming:

- Simplex (George Dantzig, 1947)
  [Indisputably one of the most important and useful algorithms of the 20th century.]
  [Walks along edges of polytope from vertex to vertex until it finds optimum.]
- Interior point methods

[Linear programming is very different from unconstrained optimization; it has a much more combinatorial flavor. If you knew which constraints would be the active constraints once you found the solution, it would be easy; the hard part is figuring out which constraints should be the active ones. There are exponentially many possibilities, so you can't afford to try them all. So linear programming algorithms tend to have a very discrete, computer science feeling to them, like graph algorithms, whereas unconstrained optimization algorithms tend to have a continuous, numerical mathematics feeling.]

[Linear programs crop up *everywhere* in engineering and science, but they're usually in disguise. An extremely useful talent you should develop is to recognize when a problem is a linear program.]

[A linear program solver can find a linear classifier, but it can't find the maximum margin classifier. We need something more powerful.]

Quadratic Program

Quadratic, convex objective fn + linear inequality constraints.

Goal: Find w that minimizes $f(w) = w^\top Q w + c^\top w$
subject to $Aw \leq b$

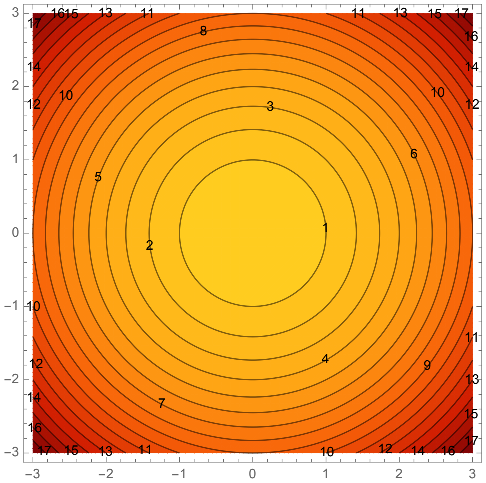
where Q is a symmetric, positive definite matrix.

A matrix is positive definite if $w^\top Q w > 0$ for all $w \neq 0$.

Only one local minimum! [Which is therefore the global minimum.]

[What if Q is not positive definite? If Q is indefinite, then f is not convex, the minimum is not always unique, and quadratic programming is NP-hard. If Q is positive semidefinite, meaning $w^\top Q w \geq 0$ for all w, then f is convex and quadratic programming is tractable, but there may be infinitely many solutions.]

Example: Find maximum margin classifier.



quadratic.pdf [Draw two polygons on these isocontours—one with one active constraint, and one with two—and show the constrained minimum for each polygon. "In an SVM, we are looking for the point in this polygon that's closest to the origin."]

Algs for quadratic programming:

- Simplex-like [commonly used for general-purpose quadratic programs, but not as good for SVMs as the following two algorithms that specifically exploit properties of SVMs]
- Sequential minimal optimization (SMO, used in LIBSVM)
- Coordinate descent (used in LIBLINEAR)

[One clever idea SMO uses is that it does a line search that uses the Hessian, but it's cheap to compute because SMO doesn't walk in the direction of steepest descent; instead it walks along just two coordinate axes at a time.]

Numerical optimization @ Berkeley: EECS 127/227AT/227BT/227C.