ML tasks as optimization problems

For a given **loss function** $L(x, y, g)$ and **probability model** $(X, Y) \sim P$, we want to find a function g to minimize **risk**:

$$\text{minimize } R(g) = \mathbb{E}_P[L(X, Y, g)]$$

Math model $\leftrightarrow$ Statistical model

Assume an i.i.d. sample from P, focus on *empirical risk minimization* (**ERM**)

$$\text{minimize } \frac{1}{n} \sum_{i=1}^{n} L(x_i, y_i, g)$$

We also choose a **function class** (or parameter set), i.e. *type of function g*, determining the *domain of the optimization*

Example: linear regression

- **Probability model**: random errors $\epsilon \sim N(0, \sigma^2)$
- **Loss function**: squared loss $L(\mathbf{x}, y, g) = (y - g(\mathbf{x}))^2$.
- **Function class**: set of all functions of the form

$$g_\beta(\mathbf{x}) = \mathbf{x}^T \beta$$

  for some $(p+1)$-dimensional vector of parameters, i.e. the function space is $\{g_\beta : \beta \in \mathbb{R}^{p+1}\}$ (domain of optimization)
- **Algorithm**: OLS, closed-form solution (memorized yet?)

$$\underset{\beta \in \mathbb{R}^{p+1}}{\text{minimize}} \frac{1}{n} \sum_{i=1}^{n} (y_i - \mathbf{x}_i^T \beta)^2$$

More examples: GLM

- **Probability model**: `family = binomial(), poisson(), etc`
- **Loss function**: Likelihood, assume indep. $\longrightarrow$ log-lik $\ell(\cdot)$
- **Function class**: $\{g^{-1}(\mathbf{x}^T\beta) : \beta \in \mathbb{R}^{p+1}\}$, with a fixed link function g
- **Algorithm**: ERM (also MLE in this case) via iterative methods like Newton-Raphson or gradient descent

$$\underset{\beta\in\mathbb{R}^{p+1}}{\text{minimize}} \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, g^{-1}(\mathbf{x}_i^T\beta))$$

Example: SVM

(*This was not assigned for reading, it's included just as an example of a different loss function*)

- Probability model: not required, geometry instead
- Empirical loss function: **hinge** loss, $[\ ]_+$ means positive part

$$\frac{1}{n}\sum_{i=1}^{n}[1-y_i(\mathbf{x}_i^T\beta-\beta_0)]_+$$

ML design choices

Pattern repeats as we learn ML methods:

(Probability) models, loss functions, prediction function classes, optimization algorithms

We'll focus more now on optimisation questions:

- Multivariate linear case
  - **Variable selection**: which predictors to include?
- Non-linear case
  - **Smoothness**: e.g. choosing the `span` value in `loess`
- Iterative algorithms
  - Scaling, **early stopping**

Optimization strategies

Choosing predictor variables

Best subset selection

- Try all 2^p subsets of predictor variables
- Keep the best one (based on RSS or deviance or something)
- Problem: complexity exponential in p, over 10^9 models if $p = 30$

Local vs global, algorithms and optima

- Imagine a "**landscape**" of loss function values as a vertical dimension above the space of predictive functions
- Searching for the lowest point in this landscape
- If more than one "valley" then multiple candidate low points
- Global algorithm: checks all of these (e.g. best subsets)
- Local algorithm: check nearby from starting point (may not converge to global optimum, may converge to a local one near the starting point)
    - **Greedy algorithm**: check nearby and move in direction of best/fastest improvement (e.g. gradient descent)

Forward stepwise/stagewise selection

Greedy alternative to best subset

1. Start with no predictors
2. At each step, find the one predictor (or a few, in stagewise) giving the best improvement (reduction in the loss function) over the current model
3. Add the best predictor(s) and iterate

- Greedy, hence local: not guaranteed to find the best model
- Computation: only p models at step 1, $p - 1$ models at step 2, etc.
- **Problem**: when to stop adding more variables? After how many steps? (We'll come back to this)

Modeling assumption: sparsity

We might be willing to assume that a "true" (good enough) model contains only a few predictors

We call this **sparsity**, and may even refer to the number of variables as "the sparsity" of the model, or look for "the best 5-sparse model"

Motivation: Occam's razor / law of parsimony -- simpler models/theories are philosophically/scientifically preferable

Sparse best subsets

Now only $\sum_{k=1}^{s} \binom{p}{k}$ models to try, if sparsity assumed $\leq s$

e.g. 174436 if $p = 30$ and $s = 5$

Coming soon: lasso

Another method to choose predictor variables

Based on sparsity assumption

Can think of it as a *less greedy* version of forward stepwise

Optimization strategies

Choosing tuning parameters

e.g. number of predictors, flexibility for non-linear methods

Modeling assumption: smoothness

Version of simplicity/parsimony for flexible function classes

Linear functions are the smoothest

Smooth function classes: set of functions with some type of bound on second derivatives, for example

Cool math fact: can be related to sparsity by considering (rate of decay of) coefficients of function's Fourier transform (smoother functions have sparser representations when written in a basis of sine functions, for example)

Discretize and fit sequentially

- Start with a grid of values for the tuning parameter
- Fit the model for each value in this grid
- Pick the best fit (visually, or based on loss function value, or...)

e.g. For the number of predictor variables, plot adjusted R-squared (or some other measure) as a function of sparsity

e.g. For the `span` or fraction s in local regression, try $s \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$ and visualize the result

- **Problem**: When to stop increasing the complexity? (i.e. decreasing the smoothness). We'll come back to this

Optimization strategies

"Scaling up" to "big data"

Computational complexity

- **Second order methods**, like Newton-Raphson, use second derivatives, i.e. *inverting the $p \times p$ Hessian matrix*
- **First order methods**, like gradient descent, only require computing the $p \times 1$ gradient vector

Many parameters $\rightarrow$ prefer first order methods

Understand this notebook on gradient descent

Coordinate descent

Update only one *coordinate* of β in each step

Cycle through coordinates until some convergence criteria is satisfied

Can combine with any strategy for univariate optimization -- e.g. one-dimensional Newton's method -- treating other parameters as constants

Optimization strategies

Scale up *more*! Bigger data!

Stochastic/random descent

- Instead of cycling through all coordinates in coordinate descent, just pick one randomly
- Instead of computing the gradient of the loss function on the entire dataset, compute it on a random sample

By identical distribution assumption, for any i', by linearity of ∇ and $\mathbb{E}$ and $\sum$,

$$\mathbb{E}[\nabla L(\mathbf{x}_{i'}, \mathbf{y}_{i'}, g_\beta)] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} \nabla L(\mathbf{x}_i, \mathbf{y}_i, g_\beta)\right]$$

Compute update using one randomly sampled observation

or a randomly sampled subset ("mini-batch SGD")

Optimization strategies

A few special topics in conclusion

Constrained optimization

Remember, some of our optimization problems have constraints on the parameters, e.g. SVM

Problem: What if the steps in these descent methods take us outside the parameter constraint region?

Solution strategy: Choose step sizes small enough to stay inside the constraint region

Solution strategy: *Project* from the updated point that is outside the constraint region to the *nearest* point inside the constraint region

Non-smooth optimization

Problem: What if the loss function is not (everywhere) differentiable?

And suppose it is *still* *convex*, e.g. hinge loss, absolute value, etc

Solution strategies: In this case there is not a well-defined gradient but there is still something called a *subgradient* which acts like a set of values that are all potential gradients--they all define tangent lines (surfaces) that *stay below the function*

Now if we're at a non-differentiable point we just need to compute any subgradient value and take a step in that direction

Early stopping

Optimization time = complexity

- For many optimization algorithms the fitted model becomes more complex the longer the optimization algorithm runs
    - e.g. the more steps of (stochastic) gradient descent used in combination with a flexible function class
    - e.g. the more steps of forward stepwise (adding more predictor variables)

Idea: control model complexity by stopping the algorithm before convergence

This is early stopping -- we'll come back to it later

Optimization theory

- If the loss function is convex many of these methods have *guaranteed convergence* to the *global minimizer*
- If the loss function is non-convex, we lose mathematical guarantees
  - Possible convergence to local minimizer
  - Local minimizers may be much worse than the best possible model...
  - Or they might not be!

Deep learning: to hell with convexity 🤠 "it just works"

Conclusion: optimization in ML is a big topic

Strategies for specific problems

e.g. stepwise inclusion of variables, constraints, etc

Strategies for general loss/function classes

e.g. gradient methods, coordinate methods

Stopping at the right amount of complexity

Maybe the most important part! Next lecture