ML - How to evaluate Models in terms of finding optimal parameter vector / fitted model

LOSS | Loss-function $L: Y \times \mathbb{R}^g \to \mathbb{R}$, $(y, f(x)) \mapsto L(y, f(x))$

Loss-function quantifies point-wise(!) the quality of prediction for a single feature vector x (observation) by assigning loss/error to it $\hat{=}$ deviation from model predict.

RISK | Theoretical Risk: Expected loss of model $R(f) := \mathbb{E}_{xy}[L(y, f(x))] = \int L(y, f(x)) \, d\mathbb{P}_{xy}$

Empirical Risk: $R_{emp}(f) := \sum_{i=1}^{n} L(y^{(i)}, f(x^{(i)}) | \theta)$ (replace integral with finite sum) This "quality score" encodes how well f fits training data by summing up all (point) losses

measure on i.i.d. training data (θ) parameterised $R_{emp}: \mathbb{R}^q \to \mathbb{R}$

average loss $\overline{R}_{emp}(f) := \frac{1}{n} \ldots$ leads to same optimization, average loss is easier to interpret

Empirical Risk Minimization: $\hat{\theta} = \arg\min_{\theta \in \Theta} R_{emp}(\theta)$ $\hat{\theta}$ not always unique, multiple θ-vectors can have same quality

$\hat{f} = \arg\min_{f \in \mathcal{H}} R_{emp}(f)$

Usually we find $\hat{\theta}$, the optimal parameter vector, by numerical optimization methods — Stochastic gradient descent; Combinatorial optimization, Analytical (OLS normal equation)

Geometrically] Find (global) minimum in multidimensional risk surface / loss landscape — But global minimum does not always exist!

Clearly every global minimum is also a **local minimum** which are much easier to find (sometimes those need to suffice)

$\exists \varepsilon > 0 \; \forall \theta$ with $\|\hat{\theta} - \theta\| < \varepsilon : R_{emp}(\hat{\theta}) \leq R_{emp}(\theta)$

$\hat{=}$ no local improvement possible inside very small ε-ball

Necessary condition for local minimum: $\hat{\theta}$ is stationary multidim. "Point": gradient $\nabla R_{emp}(\hat{\theta}) = 0$

For continuos differentiable R_{emp} this condition is sufficient if also Hessian Matrix $H(\hat{\theta})$ is positive definite

→ positive eigenspectrum

→ quadratic form $x^T H x > 0$

We know: The gradient points into direction of fastest local increase / steepest ascent

↔ negative gradient points into direction of fastest local decrease / steepest descent

GRADIENT DESCENT $\theta^{[t+1]} = \theta^{[t]} - \alpha \nabla R_{emp}(\theta^{[t]})$ Iterative optimization process for R_{emp} continous in θ

Gradient update focus on influential components of θ w.r.t. R_{emp} => We proceed more quickly along important dims of θ! Iterate up until we reach local minimum

Move from current candidate $\theta^{[t]}$ in direction of negative gradient $\hat{=}$ direction of steepest descent by some defined convergence criterium

(When Gradient → 0 we can observe steps becoming super tiny!)

α is the step-size / learning rate "How fast do we update parameter vector?"

Configuring the step-size α is crucial! We only follow quickest LOCAL decrease, so setting α too large can result in overshooting into unwanted risk increase

Small baby steps basically guarantee going down, but this is often too slow. We usually strive for something in the middle

We can use constant α, a fixed α-decrease, or dynamic changes of the learning rate

GD is a first order method. Second order methods (e.g. Newton-Raphson) use the Hessian to refine search direction for faster convergence (=> additional computing / memory required)

There are improvements for GD such as momentum and ADAM. Implementing these may enable us to smartly control the learning rate α, or to potentially escape saddle points to mimic second order behavior without computing the Hessian etc. Then of course the optimization becomes much more complicated.

Stochastic Gradient Descent (SGD): Gradient is not computed on complete data, but instead on small random batches.

For large-scale problems this is usually more efficient

BRUNNEN