Machine Learning Cheat Sheet

General

Distributions

Multivariate gaussian distribution: $\mathcal{N}(\mathbf{X}|\mu, \mathbf{\Sigma})$
$\implies p(\mathbf{X} = \mathbf{x}) =$
$(2\pi)^{-d/2}|\mathbf{\Sigma}|^{-1/2} \exp\left[-\frac{1}{2}(\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)\right]$

Gaussian distribution: $\mathcal{N}(X|\mu, \sigma^2)$
$\implies p(X = x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}(\frac{x-\mu}{\sigma})^2\right)$

Poisson distribution: $\mathcal{P}(X|\lambda)$
$\implies p(X = k) = \frac{\lambda^k}{k!} \exp(-\lambda)$

Some complexities

Product:	$O(NMD)$
Transpose:	$O(NM)$
Inversion:	$O(n^3)$

Some derivations

In respect to a vector z:

AB	$A\frac{\partial B}{\partial z} + \frac{\partial A}{\partial z}B$
A^{-1}	$-A^{-1}\frac{\partial A}{\partial z}A^{-1}$
y^TAx	$x^TA^T\frac{\partial y}{\partial z} + y^TA\frac{\partial x}{\partial z}$

In respect to a matrix A:

$\lvert A\rvert$	$\lvert A\rvert A^{-1}$
x^TAx	XX^T

Distribution properties

Let Y be any transformation and $\Sigma = cov(Y)$, then $cov(\alpha^TY, \beta^TY) = \beta^T\Sigma\alpha$.

If $X \sim \mathcal{N}(\mu, \Omega)$ and $Y = a + BX$, then $Y \sim \mathcal{N}(a + B\mu, B\Omega B^T)$.

Properties

Bayes rule: $p(A, B) = p(A|B)p(B) = p(B|A)p(A)$

A matrix $\mathbf{M}$ is **positive semidefinite** $\iff \forall$ nonzero vector $\mathbf{a}$, we have $\mathbf{a}^T\mathbf{M}\mathbf{a} \geq 0$

If $\mathbf{V}$ symmetric positive definite, then for all $\mathbf{P} \neq 0$, $\mathbf{P^TVP}$ is positive semi-definite (and even positive definite if $\mathbf{P}$ is not singular).

Jensen's inequality applied to $\log$:
$\log(\mathbb{E}[X]) \geq \mathbb{E}[\log(X)]$
$\implies \log(\sum_x x \cdot p(x)) \geq \sum_x p(x)\log(x)$

Matrix inversion lemma:
$(\mathbf{PQ} + \mathbf{I}_N)^{-1}\mathbf{P} = \mathbf{P}(\mathbf{QP} + \mathbf{I}_M)^{-1}$

Useful derivative: $\frac{\partial \mathbf{x^TBx}}{\partial \mathbf{x}} = (\mathbf{B} + \mathbf{B^T})\mathbf{x}$

Marginal and Conditional Gaussians:

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{\Lambda}^-1)$$
$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|\mathbf{A}\mathbf{x} + \mathbf{b}, \mathbf{L}^-1)$$
$$\Downarrow$$
$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{L}^-1 + \mathbf{A}\mathbf{\Lambda}^-1\mathbf{A}^T)$$
$$p(\mathbf{x}|\mathbf{y}) = \mathcal{N}(\mathbf{x}|\mathbf{\Sigma}\{\mathbf{A^T}\mathbf{L}(\mathbf{y} - \mathbf{b}) + \mathbf{\Lambda}\mu\}, \mathbf{\Sigma})$$
where $\mathbf{\Sigma} = (\mathbf{\Lambda} + \mathbf{A^T}\mathbf{L}\mathbf{A})^{-1}$

Optimization methods

Grid Search

Simply try values for all parameters at regular intervals. Complexity: $\mathcal{O}(M^DND)$, where M is the number of values tried in each dimension.

Gradient Descent

Update rule: $\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} - \alpha\frac{\partial\mathcal{L}(\boldsymbol{\beta}^{(k)})}{\partial\boldsymbol{\beta}}$

Complexity: $\mathcal{O}(IND)$ where I is the number of iterations we take.

The gradient for MSE comes out as:
$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\beta}} = -\frac{1}{N}\tilde{X}^T(\boldsymbol{y} - \tilde{X}\boldsymbol{\beta})$

Newton's method

General rule: $\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} - \alpha\mathbf{H_k}^{-1}\frac{\partial\mathcal{L}(\boldsymbol{\beta}^{(k)})}{\partial\boldsymbol{\beta}}$
where $\mathbf{H_k}$ is the $D \times D$ Hessian at step k:
$\mathbf{H_k} = \frac{\partial^2\mathcal{L}(\boldsymbol{\beta}^{(k)})}{\partial\boldsymbol{\beta}^2}$

Complexity: $\mathcal{O}(IND^2 + ID^3)$, with the D^3 cost coming from the inversion of $\mathbf{H_k}$.

For linear models, $\partial^2\mathcal{L}/\partial x^2 = N^{-1}X^TX$.

Expectation-Maximization

$\boldsymbol{\theta}_{t+1} =$
$\arg\max_\theta \sum_{n=1}^N \mathbb{E}_{p(r_n|x_n,\theta^{(i)})}[\log p(x_n, r_n|\theta)]$.

Regression

Simple linear regression: $y_n \approx \beta_0 + \beta_1 x_{n1}$

Multiple linear regression:
$y_n \approx f(\mathbf{x}_n) := \beta_0 + \beta_1 x_{n1} + \beta_2 x_{n2} + ... + \beta_D x_{nD}$

Linear basis function model

We can create more complex models while staying in the linear framework by transforming the inputs X of dimensionality D through a function $\phi : D \to M$.
$y_n = \beta_0 + \sum_{i=1}^M \beta_i\phi_i(\mathbf{x_n}) = \tilde{\boldsymbol{\phi}}^\mathbf{T}(\mathbf{x}_n^T)\boldsymbol{\beta}$. The optimal $\boldsymbol{\beta}$ can be computed in closed form by $\boldsymbol{\beta} = (\tilde{\mathbf{\Phi}}^T\tilde{\mathbf{\Phi}})^{-1}\tilde{\mathbf{\Phi}}^T y$ where $\tilde{\mathbf{\Phi}}$ is a matrix with N rows and the n-th row is $[1, \phi_1(\mathbf{x}_n), ..., \phi_M(\mathbf{x}_n)]$.

But note this requires $\tilde{\mathbf{\Phi}}^T\tilde{\mathbf{\Phi}}$ to be invertible (well-conditionned: $\tilde{\mathbf{\Phi}}$ full column-rank).

Ridge regression: $\boldsymbol{\beta}_{ridge} = (\tilde{\mathbf{\Phi}}^T\tilde{\mathbf{\Phi}} + \lambda\boldsymbol{I})^{-1}\tilde{\mathbf{\Phi}}^T\boldsymbol{y}$

Cost functions

Huber loss: $\mathcal{L}_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$

Hinge Loss: $[t]_+ = \max(0, t) = \max_{\alpha\in[0,1]} \alpha t$

Epsilon insensitive (for SVM classification):
$\mathcal{L}_\epsilon(y, \hat{y}) = \begin{cases} 0 & \text{if } |y - \hat{y}| \leq \epsilon, \\ |y - \hat{y}| - \epsilon, & \text{otherwise.} \end{cases}$

Kernel Ridge regression

$\beta^* = (X^TX + \lambda I_D)^{-1}X^Ty =$
$X^T(XX^T + \lambda I_N)^{-1}y = X^T\alpha^*$
We've

$$\beta^* = \arg\min_\beta \frac{1}{2}(y - X\beta)^T(y - X\beta) + \frac{\lambda}{2}\beta^T\beta$$
$$\alpha^* = \arg\max_\alpha -\frac{1}{2}\alpha^T(XX^T + \lambda I_M)^T\alpha + \alpha^Ty$$

Classification

Logistic Function: $\sigma(t) = \frac{exp(t)}{1+exp(t)}$

Derivative: $\frac{\partial\sigma(t)}{\partial t} = \sigma(t)[1 - \sigma(t)]$

Classification with linear regression: Use $y = 0$ as class $\mathcal{C}_1$ and $y = 1$ as class $\mathcal{C}_2$ and then decide a newly estimated y belongs to $\mathcal{C}_1$ if $y < 0.5$.

Logistic Regression

$\frac{\partial\mathcal{L}(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}} = -\tilde{\mathbf{X}}^T[\sigma(\tilde{\mathbf{X}}\boldsymbol{\beta}) - \mathbf{y}]$
$\frac{\partial^2\mathcal{L}}{\partial\beta^2} = X^TSX$, with $S = \frac{\partial\sigma(X\beta)}{\partial X\beta}$ diagonal and $S_{nn} = \sigma(\tilde{x}_n^T\beta)(1 - \sigma(\tilde{x}_n^T\beta))$
There's no closed form, we can use gradient descent.

Generalized linear model

$p(y|\eta) = \frac{h(y)}{Z}\exp\left[\eta^T\phi(y) - A(\eta)\right]$ with
$Z = \int h(y)\exp\left[\eta^T\phi(y) - A(\eta)\right]dy$.
We've

- A link function g such that $E(\phi(y)) = \mu = g^{-1}(\eta)$.
- $E(\phi(\eta)) = \frac{\partial A(\eta)}{\partial\eta}$
- $\text{var}(\phi(\eta)) = \frac{\partial^2 A(\eta)}{\partial\eta}$

With $\eta_n = \tilde{x}_n^T\beta$, we've

- $\frac{\partial\mathcal{L}}{\partial\beta} = \tilde{X}^T(g^{-1}(\eta) - \phi(y))$
- $\frac{\partial^2\mathcal{L}}{\partial^2\beta} = X^TSX$, where S diagonal and $S_{nn} = \partial^2 A(\eta_n)/\partial\eta_n^2$.

Cost functions

RMSE: $\sqrt{\frac{1}{N}\sum_{n=1}^N [y_n - \hat{p_n}]^2}$

Log-Loss:
$-\frac{1}{N}\sum_{n=1}^N y_n\log(\hat{p_n}) + (1 - y_n)\log(1 - \hat{p_n})$

Probabilistic framework

The Likelihood Function maps the model parameters to the probability distribution of $\mathbf{y}$:
$\mathcal{L}_{lik}$: parameter space $\to [0; 1] \quad \boldsymbol{\beta} \mapsto p(\mathbf{y} \mid \boldsymbol{\beta})$ An underlying p is assumed before. If the observed y are IID, $p(\mathbf{y} \mid \boldsymbol{\beta}) = \prod_n p(y_n \mid \boldsymbol{\beta})$.
$\mathcal{L}_{lik}$ can be viewed as just another cost function. Maximum likelihood then simply chooses the parameters $\boldsymbol{\beta}$ such that observed data is most likely.
$\boldsymbol{\beta} = \arg\max_{\text{all}\boldsymbol{\beta}} L(\boldsymbol{\beta})$
Assuming different p is basically what makes this so flexible. We can chose e.g.:

Gaussian p	$\mathcal{L}_{lik} \hat{=} -\mathcal{L}_{MSE}$
Laplace p	$\mathcal{L}_{lik} \hat{=} -\mathcal{L}_{MAE}$

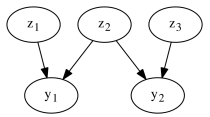
It is a sample approximation of the expected likelihood: $\mathcal{L}_{lik}(\boldsymbol{\beta}) \approx E_y[p(y \mid \boldsymbol{\beta})]$

Bayesian methods

Bayes rule: $p(A, B) = p(A|B)p(B) = p(B|A)p(A)$
The **prior** $p(\mathbf{f}|\mathbf{X})$ encodes our prior belief about the "true" model $\mathbf{f}$. The **likelihood** $p(\mathbf{y}|\mathbf{f})$ measures the probability of our (possibly noisy) observations given the prior.
Least-squares tries to find model parameters $\boldsymbol{\beta}$ which maximize the likelihood. Ridge regression maximizes the **posterior** $p(\boldsymbol{\beta}|\mathbf{y})$

Bayesian networks

We can use a Directed Acyclic Graph (DAG) to define a joint distribution of events. For example, we can express the relationship between *latent factors* (possible "causes") z_i and *observations* (results) y_i:



This example can be factorized as follows:
$p(y_1, y_2, z_1, z_2, z_3) =$
$p(y_1|z_1, z_2)p(y_2|z_2, z_3)p(z_1)p(z_2)p(z_3)$
We can then obtain the distribution over latent factors (z_i) by marginalizing over the unknown variables: $p(z_1, z_2, z_3|y_1, y_2) = \frac{\text{joint}}{p(y_1,y_2)}$
$\implies p(z_1|y_1, y_2) = \sum_{z_2,z_3}\frac{\text{joint}}{p(y_1,y_2)}$

Belief propagation

Belief propagation is a message-passing based algorithm used to compute desired marginals (e.g. $p(z_1|y_1, y_2)$) efficiently. It leverages the factorized expression of the joint. Messages passed:

$m_{z_i\to y_j} =$
$p(z_i)\Pi$(messages received except from y_j)
$m_{y_j\to z_i} =$
$\sum_{z_k\neq z_i} p(y_j|z_k)\Pi$(messages received except from z_i)

Kernel methods

$(\mathbf{K})_{i,j} = \kappa(\mathbf{x_i}, \mathbf{x_j}) = \vec{\phi}(\mathbf{x_i})^T \vec{\phi}(\mathbf{x_j})$.

Linear	$\kappa(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i}^T \mathbf{x_j}$
Polynomial	$\kappa(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i}^T \mathbf{x_j} + c)^d$
RBF	$\kappa(\mathbf{x_i}, \mathbf{x_j}) = \exp\left(-\frac{\|\mathbf{x_i}-\mathbf{x_j}\|^2}{2\sigma^2}\right)$

Properties of a Kernel:

- $\mathbf{K}$ should be symmetric: $\mathbf{K}^T = \mathbf{K}$
- $\mathbf{K}$ should be positive semi-definite: $\forall$ nonzero vector $\mathbf{a}, \mathbf{a}^T\mathbf{K}\mathbf{a} \geq 0$.

Gaussian Process

The predicting function f is interpreted as a random variable with jointly gaussian prior: $\mathcal{N}(f|\mathbf{0}, \mathbf{K})$. Defining the Kernel matrix $\mathbf{K} = \kappa(\mathbf{X}, \mathbf{X})$ defines the prior. The key idea is, that if $\mathbf{x}_i$ and $\mathbf{x_j}$ are deemed by the kernel to be similar, then we expect the output of f at those points to be similar, too.

We can sample functions from this random variable f and we can use prior + measurements to generate predictions.

If we have measurements $\mathbf{y}$ available, we get a joint distribution with the $\hat{\mathbf{y}}$ to be predicted:

$$\begin{bmatrix}\mathbf{y}\\ \hat{\mathbf{y}}\end{bmatrix} = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix}\kappa(\mathbf{X},\mathbf{X}) + \sigma_n^2 I & \kappa(\mathbf{X},\hat{\mathbf{X}})\\ \kappa(\hat{\mathbf{X}},\mathbf{X}), & \kappa(\hat{\mathbf{X}},\hat{\mathbf{X}})\end{bmatrix}\right)$$

This can be conditioned on $\mathbf{y}$ to find the PDF of $\hat{\mathbf{y}}$. Advantage: we output our prediction as probabilities (which represent uncertainty).

Neural Networks

A feed forward Neural Network is organized in K layers, each layer with $M^{(k)}$ hidden units $z_i^{(k)}$. Activations $a_i^{(k)}$ are computed as the linear combination of the previous layer's terms, with weights $\boldsymbol{\beta}^{(k)}$ (one $M^{(k-1)} \times 1$ vector of weights for each of the $M^{(k)}$ activations). Activations are then passed through a (possibly nonlinear) function h to compute the hidden unit $z_i^{(k)}$.

$$\mathbf{x}_n \xrightarrow{\boldsymbol{\beta}_i^{(1)}} a_i^{(1)} \xrightarrow{h} z_i^{(1)} \xrightarrow{\boldsymbol{\beta}^{(2)}} \dots \mathbf{z}^{(K)} = \mathbf{y}_n$$

Backpropagation

It's an algorithm which computes the gradient of the cost $\mathcal{L}$ w.r.t. the parameters $\boldsymbol{\beta}^{(k)}$.

Forward pass: compute a_i, z_i and $\mathbf{y}_n$ from $\mathbf{x}_n$.

Backward pass: work out derivatives from outputs to the target $\boldsymbol{\beta}_i^{(k)}$. Using the chain rule:

$\boldsymbol{\delta}^{(k-1)} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(k-1)}} = diag[h'(\mathbf{a}^{(k-1)})]\mathbf{B^{(k)T}}\boldsymbol{\delta}^{(k)}$

$\frac{\partial \mathcal{L}}{\partial \mathbf{B}^{(1)}} = \boldsymbol{\delta}^{(1)}\mathbf{x}^T$

$\frac{\partial \mathcal{L}}{\partial \mathbf{B}^{(k)}} = \boldsymbol{\delta}^{(k)}\mathbf{z}^{(k)T}$

Support Vector Machines

Search for the hyperplane separating the data such that the gap (margin) is biggest. It minimizes the following cost function ("hinge loss"):

$\mathcal{L}_{SVM}(\boldsymbol{\beta}) = \sum_{n=1}^N [1 - y_n\vec{\phi}_n\boldsymbol{\beta}]_+ + \frac{\lambda}{2}\sum_{j=1}^M \beta_j^2$.

This is convex but not differentiable. Using min-max theorem, we've:

$\max_{\alpha\in[0;C]^N} \boldsymbol{\alpha}^T\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{YKY}\boldsymbol{\alpha}$, with constaint $\boldsymbol{\alpha}^T\mathbf{y} = 0$.

Min-max theorem: If $G(\alpha, \beta)$ is convex in β and concave in α, then

$\min_\beta \max_\alpha G(\alpha,\beta) = \max_\alpha \min_\beta G(\alpha,\beta)$.

Clustering

K-Means

Minimize $\mathcal{L}(r,\mu) = \sum_{k=1}^K \sum_{n=1}^N r_{nk}\|x_n - \mu_k\|^2$.

Given μ_k, $r_n = \min_{j=1:K} \|x_n - \mu_j\|^2$ (use bizarre notation for r_{nk}).

Given r_{nk}, $\mu_k = \frac{\sum_{n=1}^N r_{nk}x_n}{\sum_{n=1}^N r_{nk}}$.

Gaussian Mixture Models

In mixture models, the data is generated by a sum (a mix) of K models. For GMM, these are gaussian:

$p(\mathbf{x}_i|\theta) = \sum_{k=1}^K \pi_k p_k(\mathbf{x}_i|\theta) =$
$\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_i|\underbrace{\mu_k, \mathbf{\Sigma}_k}_{\theta})$

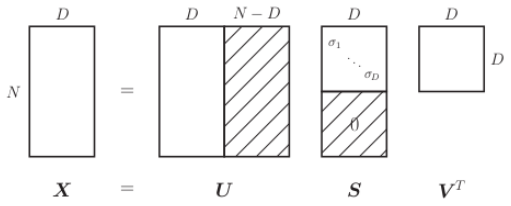
To use this for clustering, we first fit this mixture and then compute the posterior $p(z_i = k|\mathbf{x}_i, \theta)$. This yields *soft* cluster assignments.

PCA

Find the eigenvectors of the covariance matrix $\mathbf{X^T X}$ of the data. These form an orthonormal basis $\{\mathbf{w}_1, ..., \mathbf{w}_N\}$ for the data in the directions that have highest variance. One can then use the first $L < D$ vectors to rebuild the data: $\hat{\mathbf{x}}_i = \mathbf{W}\mathbf{z}_i = \mathbf{W}\mathbf{W}^T\mathbf{x}_i$, with $\mathbf{W} = [\mathbf{w}_1; ...; \mathbf{w}_L]$. This minimizes mean square error $\frac{1}{N}\sum_{i=1}^N \mathbf{x}_i - \hat{\mathbf{x}}_i^2$.

SVD

The same as with PCA, we can do with SVD:



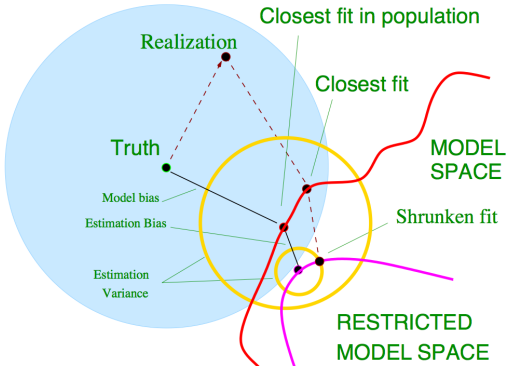
The singular vals of a $N \times D$ matrix $\mathbf{X}$ are the square roots of the eigenvalues of the $D \times D$ matrix $\mathbf{X^T X}$

Concepts

Convexity

f is called convex f: $\forall x_1, x_2 \in X, \forall t \in [0,1]$:
$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$.
Sum of two convex functions is convex. Composition of a convex function with a convex, nondecreasing function is convex. Linear, exponential and $\log(\sum \exp)$ functions are convex.

Bias-Variance Decomposition



Bias-variance comes directly out of the test error:

$$\begin{aligned}\overline{teErr} &= E[(\text{observation} - \text{prediction})^2] = E[(y - \hat{y})^2]\\ &= E[(y - y_{true} + y_{true} - \hat{y})^2]\\ &= \underbrace{E[(y - y_{true})^2]}_{\text{var of measurement}} + E[(y_{true} - \hat{y})^2]\\ &= \sigma^2 + E[(y_{true} - E[\hat{y}] + E[\hat{y}] - \hat{y})^2]\\ &= \sigma^2 + \underbrace{E[(y_{true} - E[\hat{y}])^2]}_{\text{pred bias}^2} + \underbrace{E[(E[\hat{y}] - \hat{y})^2]}_{\text{pred variance}}\end{aligned}$$

	bias	variance
regularization	+	-
choose simpler model	+	-
more data	-	

Identifiability

We say that a statistical model $\mathcal{P} = \{P_\theta : \theta \in \Theta\}$ is identifiable if the mapping $\theta \mapsto P_\theta$ is one-to-one:
$P_{\theta_1} = P_{\theta_2} \quad \Rightarrow \quad \theta_1 = \theta_2 \quad$ for all $\theta_1, \theta_2 \in \Theta$.
A non-identifiable model will typically have many local optima yielding the same cost, e.g.
$\mathcal{L}(W, Z) = \mathcal{L}(aW, \frac{1}{a}Z)$

Primal vs. Dual

Instead of working in the **column space** of our data, we can work in the **row space**:

$$\hat{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta} = \mathbf{X}\mathbf{X^T}\boldsymbol{\alpha} = \mathbf{K}\boldsymbol{\alpha}$$

where $\boldsymbol{\beta} \in \mathbb{R}^D$ and $\boldsymbol{\alpha} \in \mathbb{R}^N$ and (like magic) $\mathbf{K}$ shows up, the Kernel Matrix.

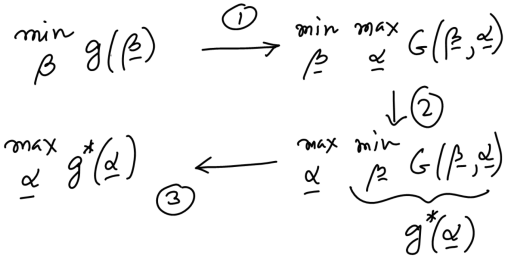
Representer Theorem: For any $\boldsymbol{\beta}$ minimizing

$$\min_\beta \sum_{n=1}^N \mathcal{L}(y_n, \mathbf{x}_n^T\boldsymbol{\beta}) + \sum_{d=1}^D \lambda\beta_d^2$$

there exists an $\boldsymbol{\alpha}$ such that $\boldsymbol{\beta} = \mathbf{X}^T\boldsymbol{\alpha}$.
When we have an explicit vector formulation of $\boldsymbol{\beta}$, we can use the matrix inversion lemma to get to the dual. E.g. for ridge regression:

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_D)^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\underbrace{(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_N)^{-1}\mathbf{y}}_{\boldsymbol{\alpha}}$$

In optimization, we get to the dual like this:



Consistency

An estimator is said to be consistent, if it eventually recovers the true parameters that generated the data as the sample size goes to infinity. Of course, this only makes sense if the data actually comes from the specified model, which is not usually the case. Nevertheless, it can be a useful theoretical property.

Efficiency

An estimator is called efficient if it achives the Cramer-Rao lower bound: $\mathrm{Var}(\boldsymbol{\beta}) \geq 1/I(\boldsymbol{\beta})$, where I is the Fisher information.

Occam's Razor

It states that among competing hypotheses, the one with the fewest assumptions should be selected. Other, more complicated solutions may ultimately prove correct, but in the absence of certainty, the fewer assumptions that are made, the better.

TODO: K-fold cross-validation, definition of Test-Error, Train-Error

TODO: statistical goodness (robust to outliers, ...) vs. computational goodness (convex, low computational complexity, ...) tradeoff. No free lunch theorem.

TODO: Decision Trees and Random Forests and Model averaging

Credits

Fork from Denwid's cheat-sheet.
Most material was taken from the lecture notes of Prof. Emtiyaz Khan.
Cost functions figure from Patrick Breheny's slides.
Biais-variance decomposition figure from Hastie, Tibshirani, Friedman, *The Elements of Statistical Learning*. The SVD figure from Kevin P. Murphy, *Machine Learning, A Probabilistic Perspective*.

Rendered January 11, 2016. Written by Dennis Meier and Merlin Nimier-David.