Machine Learning Cheat Sheet

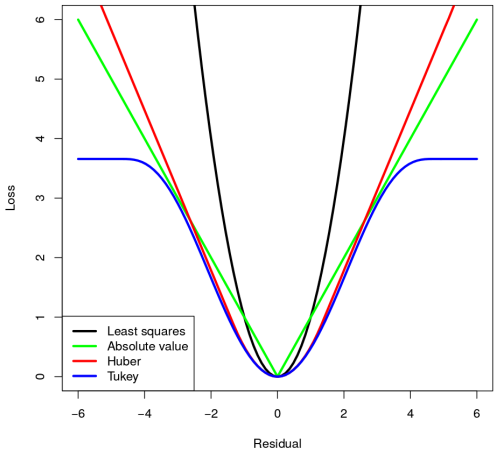
Regression

We have a set of N training examples of dimensionality D, e.g:
$x_n = \begin{bmatrix} x_{n1} & x_{n2} & \dots & x_{nD} \end{bmatrix}^T$
We often put the examples into one matrix with extra column of 1s:
$\widetilde{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1D} \\ & & \dots & & \\ 1 & x_{N1} & x_{N2} & \dots & x_{ND} \end{bmatrix}$
(note that x_n are rows, not columns)
The goal is to predict a $\hat{y}$ given x.
Simple linear regression: $y_n \approx \beta_0 + \beta_1 x_{n1}$
Multiple dimension linear regression:
$y_n \approx f(x^*) := \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + ... + \beta_D x_D^*$

Linear basis function model

$y_n = \beta_0 + \sum_{i=1}^M \beta_i \phi_i(\mathbf{x_n}) = \widetilde{\phi}^{\mathbf{T}}(\mathbf{x_n^T})\boldsymbol{\beta}$. The optimal β is given by $\beta = (\widetilde{\Phi}^T\widetilde{\Phi})^{-1}\widetilde{\Phi}^T y$ where $\widetilde{\Phi}$ is a matrix with N rows and the n-th row is $[1, \phi_1(x_n)^T, ..., \phi_M(x_n)^T]$.
Ridge regression: $\beta_{ridge} = (\widetilde{\boldsymbol{\Phi}}^T\widetilde{\boldsymbol{\Phi}} + \lambda \boldsymbol{I})^{-1}\widetilde{\boldsymbol{\Phi}}^T \boldsymbol{y}$

Cost functions



Cost function / Loss: $\mathcal{L}(\boldsymbol{\beta}) = \mathcal{L}(\mathcal{D}, \boldsymbol{\beta})$
Mean square error (MSE): $\frac{1}{2N}\sum_{n=1}^N [y_n - f(\mathbf{x_i})]^2$
Mean absolute error (MAE): $\frac{1}{2N}\sum_{n=1}^N |y_n - f(\mathbf{x_i})|$
Huber loss: $\mathcal{L}_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$
Root mean square error (RMSE): $\sqrt{2 * \text{MSE}}$
Epsilon insensitive (used for SVMs):
$\mathcal{L}_\epsilon(y, \hat{y}) = \begin{cases} 0 & \text{if } |y - \hat{y}| \le \epsilon, \\ |y - \hat{y}| - \epsilon, & \text{otherwise.} \end{cases}$

TODO: statistical/computational tradeoff

Gradient Descent

General rule: $\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} - \alpha \frac{\partial \mathcal{L}(\boldsymbol{\beta}^{(k)})}{\partial \boldsymbol{\beta}}$
How to get a good α is a hard question (too small gives slow time, too big may not converge).
The gradient for MSE comes out as:
$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = -\frac{1}{N}\widetilde{X}^T(\boldsymbol{y} - \widetilde{X}\boldsymbol{\beta})$

Newton's method

It uses second order derivatives information to converge faster (we approximate the objective function by a quadratic function rather than a line).
General rule: $\beta^{(k+1)} = \beta^{(k)} - \alpha \mathbf{H_k^{-1}} \frac{\partial \mathcal{L}(\beta^{(\mathbf{k})})}{\partial \beta}$
where $\mathbf{H_k}$ is the $D \times D$ Hessian at step k:
$\mathbf{H_k} = \frac{\partial^{\mathbf{2}} \mathcal{L}(\beta^{(\mathbf{k})})}{\partial \beta^{\mathbf{2}}}$

Normal equations

$\frac{\partial \mathcal{L}}{\partial \widetilde{\beta}} = -\mathbf{y^T X} + \mathbf{X^T X}\beta = \mathbf{0}$
$\beta = (\widetilde{X}^T\widetilde{X})^{-1}\widetilde{X}^T y$
Works for $\widetilde{X}$ having $rank = D$ (so that inverse is defined)

Classification

Logistic Function $\sigma = \frac{exp(x)}{1+exp(x)}$
$\sigma'(x) = \sigma(x)(1 - \sigma(x))$

Logistic Regression

$\frac{\partial \mathcal{L}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \tilde{\mathbf{X}}^T[\sigma(\tilde{\mathbf{X}}\beta) - \mathbf{y}]$
There's no closed form, because input is passed through a nonlinear function.

Cost functions

Root Mean square error (RMSE):
$\sqrt{\frac{1}{N}\sum_{n=1}^N [y_n - \hat{p_n}]^2}$
0-1 Loss: $\frac{1}{N}\sum_{n=1}^N \delta(y_n, \hat{y_n})$
logLoss:
$-\frac{1}{N}\sum_{n=1}^N y_n \log(\hat{p_n}) + (1 - y_n)\log(1 - \hat{p_n})$

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, butin the absence of certaintythe fewer assumptions that are made, the better.

Math

convexity:
$\forall_{x_1,x_2}\forall_{t\in[0,1]} f(tx_1 + (1-t)x_2) \le tf(x_1) + (1-t)f(x_2)$
Jensen's inequality (log is **concave**):
$log(\frac{\sum_{i=1}^n x_i}{n}) \ge \frac{\sum_{i=1}^n log(x_i)}{n}$ or
$\log(\sum_x^n x \cdot p(x)) \ge \sum_x^n p(x)\log(x)$
Hessian is positive semidefinite $\Rightarrow$ function is convex.
Positive semidefinite matrix $M \Leftrightarrow$ Gram matrix (inner product) $\Leftrightarrow \forall_x x^T M x \ge 0$.

Difficult words

consistent estimator converges to the true value as we increase the number of data to infinity.
unidentifiable model has many global minima due to symmetry.

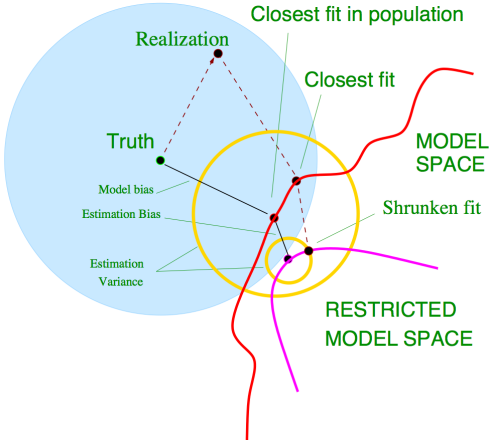
Distributions

Gaussian: $\mathcal{N}(X|\mu, \sigma^2)$
$\implies p(X = x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{1}{2}(\frac{x-\mu}{\sigma})^2)$
Poisson: $\mathcal{P}(X|\lambda)$
$\implies p(X = k) = \frac{\lambda^k}{k!}\exp(-\lambda)$

Complexities

Grid search: $\mathcal{O}(M^D N D)$, where M is the number of test points in one dimension.
Gradient descent: $\mathcal{O}(IND)$ where I is the number of iterations we make.
Least-squares (normal equations): $\mathcal{O}(ND^2 + D^3)$
Newton's method (with Hessian): $\mathcal{O}(ND^2 + D^3)$

Bias-Variance Decomposition



Bias-variance come directly out of the test error:

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

	bias	variance
regularization	+	-
reduce model complexity	+	-
more data	-	-

Maximum Likelihood

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

Gaussian p	$\mathcal{L}_{lik} \widehat{=} -\mathcal{L}_{MSE}$
Poisson p	$\mathcal{L}_{lik} \widehat{=} -\mathcal{L}_{MAE}$

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

TODO: put sample task

Mixture models

In mixture models, the data is generated by a sum (a mix) of K models. For GMM, these are gaussian:
marginal: $p(x_i|\theta) \overset{\text{marg}}{=} \sum_k p(x_i, r_i = k|\theta) = \sum_k p(x_i|r_i = k, \theta) \cdot \underbrace{p(r_i = k|\theta)}_{\pi_k} =$
$\sum_{k=1}^K \pi_k p_k(x_i|\theta) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x_i}|\underbrace{\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k}}_{\theta})$
To use this for clustering, we first fit this mixture and then compute the posterior $= \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihoood}}$
$p_{nk}^{(c)} = p(r_i = k|x_i, \theta^{(c)}) = \frac{p(x_i|r_i=k,\theta) * p(r_i=k|\theta)}{\sum_k \pi_k p_k(x_i|\theta)}$
max likelihood:
$max_\theta \log(p(x|\theta)) = max_\theta \sum_n \log(p(x_n|\theta)) = max_\theta \sum_n \log \sum_k \pi_k p_k(x_n|\theta)$
We want to maximize (log of) the marginal:
e-step:
$\log \sum_k \pi_k * p_k(x_n|\theta) = \log \sum_k \pi_k * p_k(x_n|\theta)/p_{kn}^{(c)} *$

$p_{kn}^{(c)} \geq$ (Jensen) $\sum_k p_{kn}^{(c)} \log \pi_k * p_k(x_n|\theta)/p_{kn}^{(c)} = \log a + b - c (= \text{const})$

m-step: $max_\theta \sum_n \sum_k p_{kn}^{(c)} [\log \pi_k + \log p_k(x_n|\theta)]$ - maximization in regard of all parameters (π_k, μ_k, Σ_k for GMM).

Bayesian methods

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

The **prior** $p(f)$ 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 β which maximize the likelihood.

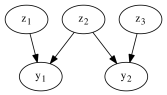
for Gaussian variable:

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

where $\mathbf{\Sigma} = (\mathbf{\Lambda} + \mathbf{A^T L A})^{-1}$

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/outputs/variables) z_i and observations/features y_i:



This example can be factorized as follows:

joint $= p(y_1, y_2, z_1, z_2, z_3) =$
$p(y_1|z_1, z_2)p(y_2|z_2, z_3)p(z_1)p(z_2)p(z_3)$

We can then obtain the z_i posterior distribution by marginalizing over the unknown variables:

$p(z_1, z_2, z_3|y_1, y_2) = \frac{\text{joint}}{p(y_1,y_2)}$

$$\implies p(z_2|y_1,y_2) = \sum_{z_1,z_3} \frac{\text{joint}}{p(y_1,y_2)} \propto \underbrace{\sum_{z_1,z_3} \text{joint}}_{\text{message}}$$

and to get marginals: $p(y) = \Pi_i p(y_i)$

$p(y_i) \overset{\text{e.g.}}{=} \sum_{z_2} p(y_i, z_2) = \sum_{z_2} p(y_i|z_2) \cdot p(z_2)$

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:

$m_{z_i \to y_j}(z_i) =$
$p(z_i)\Pi$(messages received by z_i except from y_j), e.g.
$m_{z_1 \to y_2}(z_1) = p(z_1) m_{y_3 \to z_1}(z_1)$

$F =$ set of all variables y_j depends upon

$m_{y_j \to z_i}(z_i) =$
$\sum_{F \neq z_i} p(y_j|F) \Pi_{z \in F \setminus \{z_i\}} m_{z \to y_j}(z)$, e.g.
$m_{y_2 \to z_1}(z_1) =$
$\sum_{z_2} \sum_{z_4} p(y_2|z_1, z_2, z_4) m_{z_2 \to y_2}(z_2) m_{z_4 \to y_2}(z_4)$

find a way: $p(z_i|y_1, y_2) \propto p(z_i) \cdot m_{y_1 \to z_i} \cdot m_{y_2 \to z_i}$ (if both exist). Then write messages from definition and write all paths from smallest messages to z_i.

Kernel

Basically, Kernels are a mean to measure distance, or "similarity" of two vectors. We define:

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

The ϕ are not that important in the end, because we only use the Kernel as is. Sometimes it's even impossible to write them down explicitly (RBF).

Linear	$\kappa(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i}^\mathbf{T} \mathbf{x_j}$
Polynomial	$\kappa(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i}^\mathbf{T} \mathbf{x_j} + \mathbf{c})^\mathbf{d}$
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 $\mathbf{a}, \mathbf{a^T K a} \geq \mathbf{0}$.

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:

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

This can be marginalized on $\mathbf{y}$ to find the PDF of $\hat{\mathbf{y}}$.

Advantage: we 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 $(M^{(k-1)}+1) \times 1$ vector of weights for each of the $M^{(k)}$ activations - $\beta^{(k)}$ size is $M^{(k)} \times (M^{(k-1)}+1)$ (one for free term, as in linear regression)). Activations are then passed through a (nonlinear) function h to compute the hidden unit $z_i^{(k)}$.

$$\mathbf{x_n} \xrightarrow{\boldsymbol{\beta_i^{(1)}}} \mathbf{a_i^{(1)}} \xrightarrow{\mathbf{h}} \mathbf{z_i^{(1)}} \xrightarrow{\boldsymbol{\beta^{(2)}}} \dots \xrightarrow{\mathbf{h}} \mathbf{z^{(K)}} = \mathbf{y_n}$$

Backpropagation

Algorithm: Forward pass: compute a_i, z_i and $\mathbf{y_n}$ from $\mathbf{x_n}$. Backward pass: work out derivatives of error to the target $\boldsymbol{\beta}_i^{(k)}$ using the chain rule. Emti gives some complicated formulas for that, but regular calculations should work fine.

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

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

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

Regularization

NN are not *identifiable* (existence of many local optima), so we initialize NN randomly.

NN are universal density estimators (can fit any function), and thus prone to severe overfitting. Techniques used to reduce overfitting include early stopping (stop optimizing when test error starts increasing) and "weight decay" (i.e. L_2 regularization).

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"):

$\mathcal{L}_{SVM}(\beta) = \sum_{\mathbf{n}=\mathbf{1}}^{\mathbf{N}} [\mathbf{1} - \mathbf{y_n}\tilde{\phi}_\mathbf{n}\beta]_+ + \frac{\lambda}{\mathbf{2}} \sum_{\mathbf{j}=\mathbf{1}}^{\mathbf{M}} \beta_\mathbf{j}^\mathbf{2}$

with $[t]_+ = \max(0, t)$

This is convex but not differentiable. In the dual, the same problem can be formulated as:

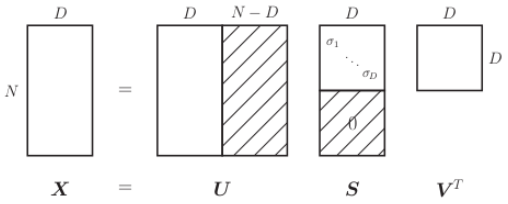
$\max_{\alpha \in [0;C]^N} \alpha^T 1 - 1/2 \alpha^T Y K Y \alpha, \alpha^T y = 0$

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: $\hat{\mathbf{x}}_\mathbf{i} = \mathbf{W z_i} = \mathbf{W W^T x_i}$, with $\mathbf{W} = [\mathbf{w_1}; ...; \mathbf{w_L}]$. This minimizes mean square error $\frac{1}{N}\sum_{i=1}^N \mathbf{x_i} - \hat{\mathbf{x}}_\mathbf{i}^\mathbf{2}$.

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 $\mathbf{X^T X}$

Curse of dimensionality

With the dimensionality increase, every data point becomes arbitrarily far from every other data point and therefore the choice of nearest neighbor becomes random. In high dimension, data only covers a tiny fraction of the input space, making generalization extremely difficult. Or in other words, the volume of the space increases so fast that the available data become sparse.

Primal vs. Dual

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

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

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

Representer Theorem: For any β minimizing

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

there exists an α such that $\beta = \mathbf{X^T}\alpha$.

When we have an explicit vector formulation of β, we can use the matrix inversion lemma to get to the dual. E.g. for ridge regression:

$$\beta = (X^T X + \lambda I_D)^{-1} X^T y = X^T \underbrace{(X X^T + \lambda I_N)^{-1} y}_{\alpha}$$

In optimization, we get to the dual like this: