Machine Learning Cheat Sheet

General

Dataset and Definitions

$\mathcal{D}$ is a set of training examples, the n-th Training Example ($n = 1, 2, ..., N$), of this set is:
$\mathbf{x_n} = \begin{bmatrix} x_{n1} & x_{n2} & ... & x_{nD} \end{bmatrix}$
We write all N training examples in a matrix:
$\mathbf{X} = \begin{bmatrix} \mathbf{x_1}; & \mathbf{x_2}; & ...; & \mathbf{x_N} \end{bmatrix}$
In supervised learning, you are also given a set of observations corresponding to the training examples:
$\mathbf{y} = \begin{bmatrix} y_1 & y_2 & ... & y_N \end{bmatrix}^\mathbf{T}$
We assume a *true* model behind the data, the observations are noisy versions of this ground truth:
$y = y_{true} + \epsilon$.
The final goal is to predict a $\hat{y} = f(\hat{\mathbf{x}})$ given any $\hat{\mathbf{x}}$.

Distributions

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

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 $\mathbf{a^T M a \geq 0}$
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:
$(\mathbf{PQ} + \mathbf{I_N})^{-\mathbf{1}}\mathbf{P} = \mathbf{P}(\mathbf{QP} + \mathbf{I_M})^{-\mathbf{1}}$
Useful derivative: $\frac{\partial \mathbf{x^T B x}}{\partial \mathbf{x}} = (\mathbf{B} + \mathbf{B^T})\mathbf{x}$
Marginal and Conditional Gaussians:

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

Optimization methods

Grid Search

Simply try values for all parameters at regular intervals. Complexity: $\mathcal{O}(M^D ND)$, 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.
α is a parameter that needs to be chosen carefully.
The gradient for MSE comes out as:
$\frac{\partial \mathcal{L}}{\partial \tilde{\boldsymbol{\beta}}} = -\frac{1}{N}\tilde{X}^T(\boldsymbol{y} - \tilde{X}\boldsymbol{\beta})$

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: $\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} - \alpha\mathbf{H_k^{-1}}\frac{\partial \mathcal{L}(\boldsymbol{\beta}^{(\mathbf{k})})}{\partial \boldsymbol{\beta}}$
where $\mathbf{H_k}$ is the $D \times D$ Hessian at step k:
$\mathbf{H_k} = \frac{\partial^\mathbf{2}\mathcal{L}(\boldsymbol{\beta}^{(\mathbf{k})})}{\partial \boldsymbol{\beta}^\mathbf{2}}$
Newton's method is equivalent to solving many least-squares problems.
Complexity: $\mathcal{O}(IND^2 + ID^3)$, with the D^3 cost coming from the inversion of $\mathbf{H_k}$.

Expectation-Maximization

EM is a family of methods to find a MLE/MAP estimator if some of the data is missing (e.g. latent variables):
Given is some data $\mathbf{x}$ and a model with some latent variables $\mathbf{z}$ and a likelihood $\mathcal{L}(\boldsymbol{\theta}; \mathbf{x}, \mathbf{z})$ (e.g. a gaussian where we need to find the mean and the covariance). Goal is to find $\beta = \arg\max_\beta \mathcal{L}_{lik}(\boldsymbol{\theta}; x)$, i.e. the MLE given only the data, without knowing the latent variables.
The problem is, that this likelihood can oftentimes not be optimized in closed form with respect to $\boldsymbol{\theta}$.
The solution is EM: Iteratively find better $\boldsymbol{\theta}$, start with $\boldsymbol{\theta}_0$ and iterate with variable t:

- E-step calculates the $\mathbb{E}$ of log likelihood, with respect to $\mathbf{Z}$ given $\mathbf{X}$ under the current estimate $\boldsymbol{\theta}_t$: $Q(\boldsymbol{\theta}, \boldsymbol{\theta}_t) = \mathbb{E}_{\mathbf{Z}|\mathbf{X},\boldsymbol{\theta_t}}[log(p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})|\mathbf{X} = \mathbf{x}]$
- M-step finds next estimate that maximizes: $\boldsymbol{\theta}_{t+1} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}_t)$
- Iterate until convergence.

Regression

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

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 β can be computed in closed form by $\beta = (\tilde{\Phi}^T\tilde{\Phi})^{-1}\tilde{\Phi}^T y$ where $\tilde{\Phi}$ is a matrix with N rows and the n-th row is $[1, \phi_1(x_n)^T, ..., \phi_M(x_n)^T]$. But note this requires $\tilde{\Phi}^T\tilde{\Phi}$ to be invertible (well-conditionned: $\tilde{\phi}$ full column-rank).
Ridge regression: $\beta_{ridge} = (\tilde{\mathbf{\Phi}}^T\tilde{\mathbf{\Phi}} + \lambda\boldsymbol{I})^{-1}\tilde{\mathbf{\Phi}}^T\boldsymbol{y}$

Cost functions

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

Least squares

Complexity: $\mathcal{O}(ND^2 + D^3)$ The gradient of the MSE set to zero is called the normal equation:

$$-\mathbf{y^T X} + \mathbf{X^T X}\beta = \mathbf{0}$$

We can solve this for β directly (by matrix manipulation) $\beta = (\tilde{X}^T\tilde{X})^{-1}\tilde{X}^T y$

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

$\frac{\partial \mathcal{L}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \tilde{\mathbf{X}}^T[\sigma(\tilde{\mathbf{X}}\beta) - \mathbf{y}]$
There's no closed form, we can use gradient descent.

Generalized linear model

The GLM consists of three elements:

- A probability distribution from the exponential family.
- A linear predictor $\hat{y} = \mathbf{X}\beta$.
- A link function g such that $E(y) = \mu = g^{-1}(\eta)$.

In a generalized linear model (GLM), each outcome of the dependent variables y is assumed to be generated from a particular distribution in the exponential family, a large range of probability distributions that includes the normal, binomial, Poisson and gamma distributions, among others.

Cost functions

RMSE: $\sqrt{\frac{1}{N}\sum_{n=1}^N [y_n - \hat{p_n}]^2}$
0-1 Loss: $\frac{1}{N}\sum_{n=1}^N \delta(y_n, \hat{y_n})$
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 $\to [0; 1] \quad \beta \mapsto \mathbf{p}(\mathbf{y} \mid \beta)$ 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	$\mathcal{L}_{lik} \hat{=} -\mathcal{L}_{MSE}$
Poisson p	$\mathcal{L}_{lik} \hat{=} -\mathcal{L}_{MAE}$

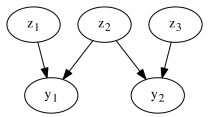
It is a sample approximation of the expected likelihood: $\mathcal{L}_{lik}(\boldsymbol{\beta}) \approx E_y[p(y \mid \boldsymbol{\beta})]$

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 β which maximize the likelihood. Ridge regression maximizes the **posterior** $p(\beta|\mathbf{y})$

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

Kernels are another way to encode our prior believe about the relation of outputs to inputs. A kernel defines a distance measure, or "similarity" of two vectors. We define:
$(\mathbf{K})_{i,j} = \kappa(\mathbf{x_i}, \mathbf{x_j}) = \tilde{\phi}(\mathbf{x_i})^\mathbf{T} \tilde{\phi}(\mathbf{x_j})$.
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.

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

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

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

Regularization

NN are not *identifiable* (existence of many local optima), therefore the maximum likelihood estimator is not *consistent*.
NN are universal density estimators, 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"):
$\mathcal{L}_{SVM}(\beta) = \sum_{\mathbf{n=1}}^{\mathbf{N}}[\mathbf{1} - \mathbf{y_n}\tilde{\boldsymbol{\phi}}_\mathbf{n}\beta]_+ + \frac{\lambda}{\mathbf{2}}\sum_{\mathbf{j=1}}^{\mathbf{M}}\beta_\mathbf{j}^\mathbf{2}$
with $[t]_+ = \max(0, t)$
This is convex but not differentiable. In the dual, the same problem can be formulated as:
$\max_{\alpha \in [0;C]^N} \alpha^T 1 - 1/2\alpha^T Y K Y \alpha, \alpha^T y = 0$

Gaussian Mixture Models

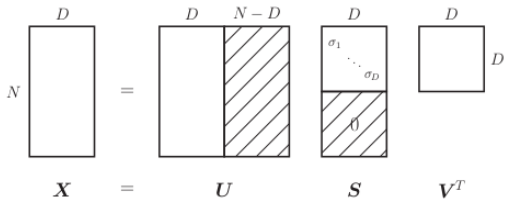
In mixture models, the data is generated by a sum (a mix) of K models. For GMM, these are gaussian:
$p(\mathbf{x_i}|\theta) = \sum_{\mathbf{k=1}}^{\mathbf{K}} \pi_\mathbf{k}\mathbf{p_k}(\mathbf{x_i}|\theta) =$
$\sum_{\mathbf{k=1}}^{\mathbf{K}} \pi_\mathbf{k}\mathcal{N}(\mathbf{x_i}|\underbrace{\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k}}_{\theta})$
To use this for clustering, we first fit this mixture and then compute the posterior $p(z_i = k|\mathbf{x_i}, \theta)$. 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: $\hat{\mathbf{x}}_\mathbf{i} = \mathbf{W}\mathbf{z_i} = \mathbf{W}\mathbf{W}^\mathbf{T}\mathbf{x_i}$, with $\mathbf{W} = [\mathbf{w_1}; ...; \mathbf{w_L}]$. This minimizes mean square error $\frac{1}{N}\sum_{i=1}^N \mathbf{x_i} - \hat{\mathbf{x}}_\mathbf{i}^\mathbf{2}$.

SVD

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



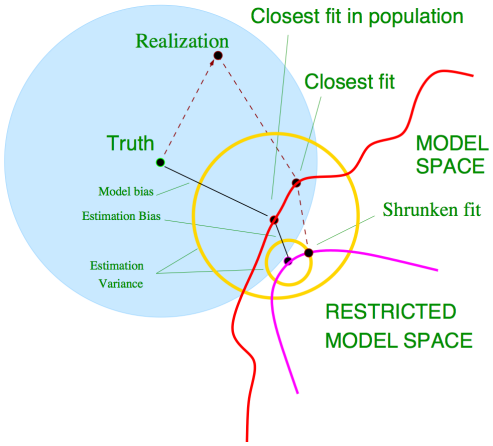
The singular vals of a $N \times D$ matrix $\mathbf{X}$ are the square roots of the eigenvalues of the $D \times D$ matrix $\mathbf{X^T X}$

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.
$\mathcal{L}(W, Z) = \mathcal{L}(aW, \frac{1}{a}Z)$

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**:

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

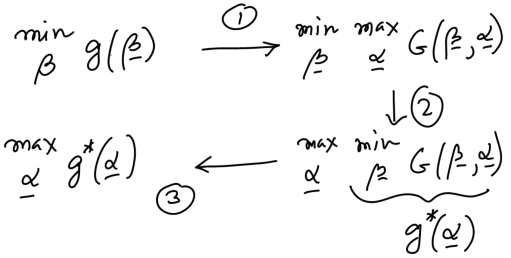
where $\beta \in \mathbb{R}^\mathbf{D}$ and $\alpha \in \mathbb{R}^\mathbf{N}$ and (like magic) $\mathbf{K}$ shows up, the Kernel Matrix.
Representer Theorem: For any β minimizing

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

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:

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

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: $\text{Var}(\beta) \geq 1/I(\beta)$, 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

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*.