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:
$\mathbf{x}_n = \begin{bmatrix} x_{n1} & x_{n2} & ... & x_{nD} \end{bmatrix}^T$
We write all N training examples in a matrix:
$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T; & \mathbf{x}_2^T; & ...; & \mathbf{x}_N^T \end{bmatrix}$
In supervised learning, you are also given a set of observations corresponding to the training examples:
$\mathbf{y} = \begin{bmatrix} y_1 & y_2 & ... & y_N \end{bmatrix}^T$
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 $\hat{\mathbf{x}}$.

Vector and matrix derivatives

First Order

$\frac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a} \qquad \frac{\partial \mathbf{a}^T \mathbf{X} \mathbf{b}}{\partial \mathbf{X}} = \mathbf{a}\mathbf{b}^T$
$\frac{\partial \mathbf{a}^T \mathbf{X}^T \mathbf{b}}{\partial \mathbf{X}} = \mathbf{b}\mathbf{a}^T \qquad \frac{\partial \mathbf{a}^T \mathbf{X} \mathbf{a}}{\partial \mathbf{X}} = \frac{\partial \mathbf{a}^T \mathbf{X}^T \mathbf{a}}{\partial \mathbf{X}} = \mathbf{a}\mathbf{a}^T$

Second Order

$\frac{\partial \mathbf{b}^T \mathbf{X}^T \mathbf{X} \mathbf{c}}{\partial \mathbf{X}} = \mathbf{X}(\mathbf{b}\mathbf{c}^T + \mathbf{c}\mathbf{b}^T)$
$\frac{\partial \mathbf{x}^T \mathbf{B} \mathbf{x}}{\partial \mathbf{x}} = (\mathbf{B} + \mathbf{B}^T)\mathbf{x}$

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

Properties

A matrix $\mathbf{M}$ is

positive semi-definite $\iff \forall$ nonzero vector $\mathbf{a}$, we have $\mathbf{a}^T \mathbf{M} \mathbf{a} \geq 0$

symmetric $\mathbf{M}^T = \mathbf{M}$

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:
$(\mathbf{P}\mathbf{Q} + \mathbf{I}_N)^{-1} \mathbf{P} = \mathbf{P}(\mathbf{Q}\mathbf{P} + \mathbf{I}_M)^{-1}$
Solution optimality :

1^{st}**-order** ***necessary*** $\frac{\partial \mathcal{L}(\boldsymbol{\beta}^*)}{\partial \boldsymbol{\beta}} = 0$

2^{nd}**-order** ***sufficient*** $\mathbf{H}(\boldsymbol{\beta}^*) := \frac{\partial^2 \mathcal{L}(\boldsymbol{\beta}^*)}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T}$ is p.s.d.

Marginal and Conditional Gaussians:

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

Optimization methods

Grid Search $\mathcal{O}(M^D ND)$

Simply try M values for all parameters at regular intervals.

Gradient Descent $\mathcal{O}(IND)$

Update rule: $\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} - \alpha \frac{\partial \mathcal{L}(\boldsymbol{\beta}^{(k)})}{\partial \boldsymbol{\beta}}$
α is a parameter that needs to be chosen carefully.
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 $\mathcal{O}(IND^2 + ID^3)$

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

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 $\boldsymbol{\theta}_t$:
  $Q(\boldsymbol{\theta}, \boldsymbol{\theta}_t) = \mathbb{E}_{\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}_t}[log(p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})|X = x]$
- 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:
$y_n \approx f(\mathbf{x}_n) := \beta_0 + \beta_1 x_{n1} + \beta_2 x_{n2} + ... + \beta_D x_{nD}$

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 $\phi : D \to M$.
$y_n = \beta_0 + \sum_{i=1}^M \beta_i \phi_i(\mathbf{x_n}) = \tilde{\phi}^\mathbf{T}(\mathbf{x}_n^T)\boldsymbol{\beta}$. 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

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: $\frac{1}{2N} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$
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 $\mathcal{O}(ND^2 + D^3)$

The gradient of the MSE set to zero is called the normal equation: $-\mathbf{y}^T \mathbf{X} + \mathbf{X}^T \mathbf{X} \boldsymbol{\beta} = 0$.
We can solve this for $\boldsymbol{\beta}$ directly (by matrix manipulation) $\boldsymbol{\beta} = (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^T \mathbf{y}$

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

$\frac{\partial \mathcal{L}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \tilde{\mathbf{X}}^T [\sigma(\tilde{\mathbf{X}}\boldsymbol{\beta}) - \mathbf{y}]$
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 $\hat{y} = \mathbf{X}\boldsymbol{\beta}$.
- A link function g s.t. $E(y) = \mu = g^{-1}(\eta)$.

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 $\to [0; 1] \quad \boldsymbol{\beta} \mapsto p(\mathbf{y} \mid \boldsymbol{\beta})$ 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 $\boldsymbol{\beta}$ s.t. 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	$\mathcal{L}_{lik} \hat{=} -\mathcal{L}_{MSE}$
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: $p(A, B) = p(A|B)p(B) = p(B|A)p(A)$
$\underbrace{p(y, z)}_{\text{joint}} = \underbrace{p(y|z)}_{\text{likelihood}} \underbrace{p(z)}_{\text{prior}} = \underbrace{p(z|y)}_{\text{posterior}} \underbrace{p(y)}_{\text{mar. li}}$
Least-squares tries to find model parameters $\boldsymbol{\beta}$ which maximize the likelihood. Ridge regression maximizes the **posterior** $p(\boldsymbol{\beta}|\mathbf{y})$.
Bayesian method are biased.

Bayesian networks

We can use a Directed Acyclic Graph (DAG) to define a joint distribution of events :
$p(\mathbf{x}) = \prod_{k=1}^K p(x_k|\mathbb{P}_k)$
We can then obtain the distribution over latent factors (z_i) by marginalizing over the unknown variables: $p(z_1, z_2, z_3|y_1, y_2) = \frac{\text{joint}}{p(y_1, y_2)}$
$\implies p(z_1|y_1, y_2) = \sum_{z_2, z_3} \frac{\text{joint}}{p(y_1, y_2)}$

Sum-product

$$\sum_{z_3}\sum_{z_2} p(y_a|z_1,z_2)p(y_b|z_2,z_3)p(z_1)p(z_2)p(z_3) =$$
$$p(z_1)\sum_{z_2} p(y_a|z_1,z_2)p(z_2)\sum_{z_3} p(y_b|z_2,z_3)p(z_3)$$

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:

vars to obs

$$m_{z_i\to a}(z_i) = p(z_i)\prod_{b\in\mathbb{N}(i)\setminus a} m_{b\to i}(z_i)$$

obs to vars $m_{a\to i}(z_i) =$
$\sum_{z_j,\forall j} p(y_a|\mathbb{P}_a)\prod_{j\in\mathbb{N}(a)\setminus i} m_{j\to a}(z_j)$

where $\mathbb{N}$ is the set of all neighbours and $\mathbb{P}$ the set of all parents.

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:
$(\mathbf{K})_{i,j} = \kappa(\mathbf{x_i},\mathbf{x_j}) = \vec{\phi}(\mathbf{x_i})^T\vec{\phi}(\mathbf{x_j})$.
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	$\kappa(\mathbf{x_i},\mathbf{x_j}) = \mathbf{x_i}^T\mathbf{x_j}$
Polynomial	$\kappa(\mathbf{x_i},\mathbf{x_j}) = (\mathbf{x_i}^T\mathbf{x_j}+c)^d$
RBF	$\kappa(\mathbf{x_i},\mathbf{x_j}) = \exp\left(-\frac{\lvert\lvert\mathbf{x_i}-\mathbf{x_j}\rvert\rvert^2}{2\sigma^2}\right)$

Properties of a Kernel:

- $\mathbf{K}$ symmetric
- $\mathbf{K}$ p.s.d.

Gaussian Process $\mathcal{O}(N^3)$

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). Decrease variance by averaging.

Neural Networks

$\mathbf{x}_n \xrightarrow{\boldsymbol{\beta}_i^{(1)}} a_i^{(1)} \xrightarrow{h} z_i^{(1)} \xrightarrow{\boldsymbol{\beta}^{(2)}} \dots \mathbf{z}^{(K)} = \mathbf{y}_n$ A feed forward Neural Network is organized in K layers, each layer with $M^{(k)}$ hidden units $z_i^{(k)}$.
$\forall M$ hidden units :

First layer : $a_{nm}^{(1)} = (\boldsymbol{\beta}_m^{(1)})^T\tilde{\mathbf{x}}_n, \quad z_{nm}^{(1)} = h(a_{nm}^{(1)})$

Layer k **:** $a_{nm}^{(k)} = (\boldsymbol{\beta}_m^{(k)})^T\tilde{\mathbf{z}}_n, \quad z_{nm}^{(k)} = h(a_{nm}^{(k)})$

Layer K **:** $a_{n1}^{(K)} = (\boldsymbol{\beta}_1^{(K)})^T\tilde{\mathbf{z}}_n, \quad z_{n1}^{(K)} = h(a_{n1}^{(K)})$

Output for ...

regression $\hat{y}_n = z_{n1}^{(K)}$

classification $\hat{y}_n = \sigma(z_{n1}^{(K)})$

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

Chain Rule

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\beta}_m^{(1)}} = \frac{\partial a_m^{(1)}}{\partial\boldsymbol{\beta}_m}\frac{\partial\mathcal{L}}{\partial a_m^{(1)}} = \mathbf{x}\frac{\partial\mathcal{L}}{\partial a_m^{(1)}}$$
$$a_m^{(1)} = \boldsymbol{\beta}_m^{(1)^T}\mathbf{x} \qquad z_m^{(1)} = h(a_m^{(1)})$$
$$\frac{\partial\mathcal{L}}{\partial a_m^{(1)}} = \frac{\partial z_m^{(1)}}{\partial a_m^{(1)}}\frac{\partial\mathcal{L}}{\partial z_m^{(1)}} = h'(a_m^{(1)})\frac{\partial\mathcal{L}}{\partial z_m^{(1)}}$$
$$\frac{\partial\mathcal{L}}{\partial z_m^{(1)}} = \frac{\partial a^{(2)}}{\partial z_m^{(1)}}\frac{\partial\mathcal{L}}{\partial a^{(2)}} \qquad a^{(2)} = \beta_1^{(2)^T}\mathbf{z}^{(1)}$$

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 s.t. the gap (margin) is biggest. It minimizes the following cost function ("hinge loss"):
$\mathcal{L}_{SVM}(\boldsymbol{\beta}) = \sum_{n=1}^N[1-y_n\tilde{\phi}_n\boldsymbol{\beta}]_+ + \frac{\lambda}{2}\sum_{j=1}^M\beta_j^2$
with $[t]_+ = \max(0,t)$
This is convex but not differentiable. In the dual, the same problem can be formulated as:
$\max_{\alpha\in[0;C]^N}\boldsymbol{\alpha}^T\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{YKY}\boldsymbol{\alpha}, \quad \boldsymbol{\alpha}^T\mathbf{y} = 0$

Gaussian Mixture Models

In mixture models, the data is generated by a sum (a mix) of K models. For GMM, these are gaussian:
$p(\mathbf{x}_i|\theta) = \sum_{k=1}^K\pi_k p_k(\mathbf{x}_i|\theta) =$
$\sum_{k=1}^K\pi_k\mathcal{N}(\mathbf{x}_i|\underbrace{\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k}_{\theta})$

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

PCA $\mathcal{O}(M^2DN_{+M^3N+M^3D})$

Lossy compression.
Find the eigenvectors of the covariance matrix $\mathbf{X^TX}$ 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: $\hat{\mathbf{x}}_i = \mathbf{Wz}_i = \mathbf{WW}^T\mathbf{x}_i$, with $\mathbf{W} = [\mathbf{w}_1;...;\mathbf{w}_L]$. This minimizes mean square error $\frac{1}{N}\sum_{i=1}^N\mathbf{x}_i - \hat{\mathbf{x}}_i^2$.
$\mathbf{Z}^T \leftarrow (\mathbf{W}^T\mathbf{W}^{-1}\mathbf{W}^T\mathbf{X}$
$\mathbf{W}^T \leftarrow (\mathbf{Z}^T\mathbf{Z}^{-1}\mathbf{Z}^T\mathbf{X}^T$

SVD $\mathcal{O}(4D^2N+8DN^2+9D^3)$

Singular value decomposition

Exact factorization. $\mathbf{X} = \mathbf{USV}^T$
The singular values of a $N\times D$ matrix $\mathbf{X}$ are the square roots of the eigenvalues of the $D\times D$ matrix $\mathbf{X^TX}$

Low rank approximation

$\tilde{\mathbf{X}} = \mathbf{WZ}^T, \quad$ where $\mathbf{W} = \mathbf{US}^{\frac{1}{2}}$, $\mathbf{Z} = \mathbf{VS}^{\frac{1}{2}}$

Concepts

Convexity

f is called convex f: $\forall x_1,x_2\in X,\forall t\in[0,1]$:
$f(tx_1+(1-t)x_2) \le tf(x_1)+(1-t)f(x_2)$.
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

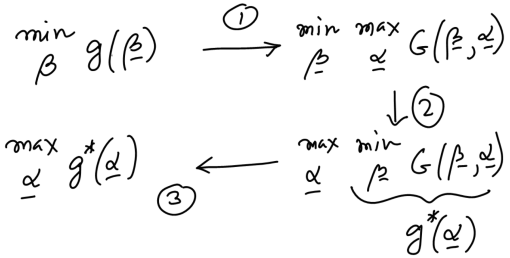
With dimensionality increase, data point become arbitrarily far from each other. Bottom line: reduce feature dimensionality whenever possible.

Primal vs. Dual

Instead of working in the **column space** of our data, we can work in the **row space**:
$\hat{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta} = \mathbf{XX}^T\boldsymbol{\alpha} = \mathbf{K}\boldsymbol{\alpha}$,
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 $\min_\beta\sum_{n=1}^N\mathcal{L}(y_n,\mathbf{x}_n^T\boldsymbol{\beta}) + \sum_{d=1}^D\lambda\beta_d^2$,
$\exists\boldsymbol{\alpha}$ s.t. $\boldsymbol{\beta} = \mathbf{X}^T\boldsymbol{\alpha}$.
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:

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

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: $\text{Var}(\boldsymbol{\beta}) \ge 1/I(\boldsymbol{\beta})$, where I is the Fisher information.

Reducing Overfitting

- penalize terms
- reduce dimensionality
- more data

Occam's Razor

It states that among competing hypotheses, the one with the fewest assumptions should be selected.

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*.