PCML Cheat Sheet

1 Math Prerequisites

- Bayes rule

$$p(A,B) = \underbrace{p(A|B)}_{\text{Lik.}}\underbrace{p(B)}_{\text{Prior}} = \underbrace{p(B|A)}_{\text{Post}}\quad\underbrace{p(A)}_{\text{Marg. Lik.}}$$

- Bayes rule: $P(A|B) \propto P(B|A)P(A)$
- Gaussian distribution

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{\sqrt{2\pi|\boldsymbol{\Sigma}|}}\exp(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}))$$

- Production of independent variables:

$$\text{Var}(XY) = \mathbb{E}(X^2)\,\mathbb{E}(Y^2) - [\mathbb{E}(X)]^2[\mathbb{E}(Y)]^2$$

- Covariance matrix of a data vector $\mathbf{x}$

$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \mathbb{E}(\mathbf{x}))(\mathbf{x}_n - \mathbb{E}(\mathbf{x}))^T$$

1.1 Convexity

- The Hessian of a convex function is psd and for a strictly-convex function it's pd.

$$\mathbf{H}_{i,j} = d^2f/dx_i dx_j$$

1.2 Linear Algebra

- **Condition number** If $\mathbf{A}$ is normal ($A^TA = AA^T$) then

$$k(\mathbf{A}) = \left|\frac{\lambda_{max}(\mathbf{A})}{\lambda_{min}(\mathbf{A})}\right|$$

2 Cost functions

Mean square error (MSE):

$$MSE(\boldsymbol{w}) = \sum_{n=1}^{N}(y_n - f(\mathbf{x}_n))^2$$

Mean Absolute Error (MAE):

$$MAE = \sum_{n=1}^{N}|y_n - f(\mathbf{x}_n)|$$

Huber loss

$$Huber = \begin{cases} \frac{1}{2}z^2 & ,|z| \le \delta \\ \delta|z| - \frac{1}{2}\delta^2 & ,|z| > \delta \end{cases}$$

Hinge loss

$$Hinge = [1 - y_n f(\mathbf{x}_n)]_+ = \max(0, 1 - y_n f(\mathbf{x}_n))$$

Logistic loss

$$Logistic = \log(1 - \exp(y_n f(\mathbf{x}_n)))$$

3 Optimization

3.1 Grid search

- Compute the cost over a grid of M points to find the minimum. Exponential Complexity. Hard to find a good range of values

3.2 Gradient Descent

- GD uses only first-order information and takes steps in the opposite direction of the gradient
- Given cost function $\mathcal{L}(\boldsymbol{w})$ we want to find $\boldsymbol{w}$

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})$$

3.3 Batch Gradient Descent

- Take steps in the opposite direction of the gradient

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \gamma\boldsymbol{\nabla}\mathcal{L}(\boldsymbol{w}^{(t)})$$

  with $\gamma > 0$ the learning rate.
- With γ too big, method might diverge. With γ too small, convergence is slow.

3.4 Gradients for MSE

- We define the error vector $\mathbf{e}$:

$$\mathbf{e} := \mathbf{y} - \mathbf{X}\boldsymbol{w}$$

- and MSE as follows:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2N}\sum_{n=1}^{N}(y_n - \tilde{\mathbf{x}}_n^T\boldsymbol{w})^2 = \frac{1}{2N}\mathbf{e}^T\mathbf{e}$$

- then the gradient is given by

$$\boldsymbol{\nabla}\mathcal{L}(\boldsymbol{w}) = -\frac{1}{N}\mathbf{X}^T\mathbf{e}$$

- Optimality conditions:
  1. *necessary*: gradient equal zero: $\frac{d\mathcal{L}(\boldsymbol{w}^*)}{d\boldsymbol{w}} = 0$
  2. *sufficient*: Hessian matrix is positive definite: $\mathbf{H}(\boldsymbol{w}^*) = \frac{d^2\mathcal{L}(\boldsymbol{w}^*)}{d\boldsymbol{w}d\boldsymbol{w}^T}$
- Very sensitive to illconditioning $\Rightarrow$ always normalize features.
- *Complexity*: $O(NDI)$ with I the number of iterations

3.5 Stochastic Gradient Descent

In ML, most cost functions are formulated as a sum over the training examples:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}_n(\boldsymbol{w})$$

$\Rightarrow$ SGD algo is given by update rule:

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \gamma\boldsymbol{\nabla}\mathcal{L}_n(\boldsymbol{w}^{(t)})$$

Idea: Cheap but unbiased estimate of grad.

$$\mathbb{E}[\boldsymbol{\nabla}\mathcal{L}_n(\boldsymbol{w})] = \boldsymbol{\nabla}\mathcal{L}_n(\boldsymbol{w})$$

3.6 Mini-batch SGD

Update direction ($B \subseteq [N]$):

$$\boldsymbol{g}^{(t)} := \frac{1}{|B|}\sum_{n\in B}\boldsymbol{\nabla}\mathcal{L}_n(\boldsymbol{w}^{(t)})$$

Update rule : $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \gamma\boldsymbol{g}^{(t)}$

3.7 Subgradients (Non-Smooth optim)

A vector $\boldsymbol{g} \in \mathbb{R}^D$ s.t.

$$\mathcal{L}(\boldsymbol{u}) \ge \mathcal{L}(\boldsymbol{w}) + \boldsymbol{g}^T(\boldsymbol{u} - \boldsymbol{w}) \quad \forall\boldsymbol{u}$$

is the subgradient to $\mathcal{L}$ at $\boldsymbol{w}$. If $\mathcal{L}$ is differentiable at $\boldsymbol{w}$, we have $\boldsymbol{g} = \boldsymbol{\nabla}\mathcal{L}(\boldsymbol{w})$

4 Least Squares

- Use the first optimality conditions:

$$\boldsymbol{\nabla}L(\boldsymbol{w}^*) = 0 \Rightarrow \mathbf{X}^T\mathbf{e} = \mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{w}) = 0$$

- When $\mathbf{X}^T\mathbf{X}$ is invertible, we have the closed-form expression

$$\boldsymbol{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

- thus we can predict values for a new $\mathbf{x}_m$

$$y_m := \mathbf{x}_{\mathbf{m}}^{\mathbf{T}}\boldsymbol{w}^* = \mathbf{x}_{\mathbf{m}}^{\mathbf{T}}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

- The **Gram matrix** $\mathbf{X}^T\mathbf{X}$ is pd and is also invertible iff $\mathbf{X}$ has full column rank.
- *Complexity*: $O(ND^2 + D^3) \equiv O(ND^2)$
- $\mathbf{X}$ can be rank deficient when $D > N$ or when the comlumns $\bar{\mathbf{x}}_d$ are nearly collinear. $\Rightarrow$ matrix is ill-conditioned.

5 Maximum Likelihood

- Let define our mistakes $\epsilon_n \sim \mathcal{N}(0,\sigma^2)$.

$$\rightarrow y_n = \mathbf{x}_n^T\boldsymbol{w} + \epsilon_n$$

- Another way of expressing this:

$$p(\mathbf{y}|\mathbf{X},\mathbf{w}) = \prod_{n=1}^{N}p(y_n|\mathbf{x}_n,\boldsymbol{w})$$

$$= \prod_{n=1}^{N}\mathcal{N}(y_n|\mathbf{x}_n^T\boldsymbol{w},\sigma^2)$$

  which defines the likelihood of observating $\mathbf{y}$ given $\mathbf{X}$ and $\boldsymbol{w}$
- Define cost with log-likelihood

$$\mathcal{L}_{lik}(\boldsymbol{w}) = \log p(\mathbf{y}|\mathbf{X},\boldsymbol{w})$$

$$= -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(y_n - \mathbf{x}_n^T\boldsymbol{w})^2 + cnst$$

- Maximum likelihood estimator (MLE) gives another way to design cost functions

$$\underset{\boldsymbol{w}}{\text{argmin}}\,\mathcal{L}_{MSE}(\boldsymbol{w}) = \underset{\boldsymbol{w}}{\text{argmax}}\,\mathcal{L}_{lik}(\boldsymbol{w})$$

- MLE can also be interpreted as finding the model under which the observed data is most likely to have been generated from.
- With Laplace distribution

$$p(y_n|\mathbf{x}_n,\boldsymbol{w}) = \frac{1}{2b}e^{-\frac{1}{b}|y_n - \mathbf{x}_n^T\boldsymbol{w}|}$$

$$\sum_n \log p(y_n|\mathbf{x}_n,\boldsymbol{w}) = \sum_n |y_n - \mathbf{x}_n^T\boldsymbol{w}| + cnst$$

6 Ridge Regression

- Basis functions:

$$y_n = w_0 + \sum_{j=1}^{M}w_j\phi_j(\mathbf{x}_n) = \tilde{\boldsymbol{\phi}}(\mathbf{x}_n)^T\boldsymbol{w}$$

- This model is linear in $\boldsymbol{w}$ but nonlinear in $\mathbf{x}$. Dimension is now M, not N.
- Polynomial basis

$$\boldsymbol{\phi}(x_n) = [1, x_n, x_n^2, ..., x_n^M]$$

- The least square solution becomes

$$\boldsymbol{w}_{lse}^* = (\tilde{\boldsymbol{\Phi}}^T\tilde{\boldsymbol{\Phi}})^{-1}\tilde{\boldsymbol{\Phi}}^T\mathbf{y}$$

- Complex models overfit easily. Thus we can penalize them with a **regularization term**

$$\min_{\boldsymbol{w}}\left(\mathcal{L}(\boldsymbol{w}) + \frac{\lambda}{2N}\sum_{j=1}^{M}w_j^2\right)$$

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\text{argmin}}\left(\frac{1}{2}(\mathbf{y} - \mathbf{X}\boldsymbol{w})^T(\mathbf{y} - \mathbf{X}\boldsymbol{w}) + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}\right)$$

- Note that w_0 is not penalized.
- By differentiating and setting to zero we get

$$\boldsymbol{w}_{ridge} = (\tilde{\boldsymbol{\Phi}}^T\tilde{\boldsymbol{\Phi}} + \boldsymbol{\Lambda})^{-1}\tilde{\boldsymbol{\Phi}}^T\mathbf{y}$$

$$\boldsymbol{\Lambda} = \begin{bmatrix} 0 & \underline{0} \\ \underline{0} & \lambda I_M \end{bmatrix}$$

- Ridge regression improves the condition number of the Gram matrix since the eigenvalues of $(\tilde{\boldsymbol{\Phi}}^T\tilde{\boldsymbol{\Phi}} + \lambda I_m)$ are at least λ: SVD $\tilde{\Phi}^T\tilde{\Phi} = USU^T$ with $S = diag(\ge 0)$, then $\tilde{\Phi}^T\tilde{\Phi} + \Lambda = U(S+\Lambda)U^T$ has eigenvalues $\ge \lambda$.
- **Maximum-a-posteriori (MAP) estimator**:
  - Maximizes the product of the likelihood *and* the **prior**.

$$\boldsymbol{w}_{MAP} = \underset{\boldsymbol{w}}{\text{argmax}}\,(p(\mathbf{y}|\mathbf{X},\boldsymbol{\Lambda})p(\boldsymbol{w}|\boldsymbol{\Sigma}))$$

  - Assume $w_0 = 0$

$$\boldsymbol{w}_{ridge} = \underset{\boldsymbol{w}}{\text{argmax}}\left(\log\left[\prod_{n=1}^{N}\mathcal{N}(y_n|\mathbf{x}_n^T\boldsymbol{w},\boldsymbol{\Lambda}) \times \mathcal{N}(\boldsymbol{w}|0,\underline{\mathbf{I}})\right]\right)$$

- **Lasso regularizer** forces some w_i to be strictly 0 and therefore forces sparsity in the model.

$$\min_{\boldsymbol{w}}\frac{1}{2N}\sum_{n=1}^{N}(y_n - \tilde{\boldsymbol{\phi}}(\mathbf{x}_n)^T\boldsymbol{w})^2,$$

$$\text{such that }\sum_{i=1}^{M}|w_i| \le \tau$$

7 Bias-Variance decomposition

- The expected test error can be expressed as the sum of two terms

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

- Model bias and estimation bias are important
- RR increases estimation bias and reduces var
- Model more complex increases test error
- Small $\lambda \rightarrow$ low bias but large variance
- Large $\lambda \rightarrow$ large bias but low variance

$$err = \sigma^2 + \mathbb{E}[f_{lse} - \mathbb{E}[f_{lse}]]^2 + [f_{true} - \mathbb{E}[f_{lse}]]^2$$

8 Logistic Regression

- **Classification** relates input variables $\mathbf{x}$ to discrete output variable y
- **Binary classifier**: we use $y = 0$ for $\mathbf{C}_1$ and $y = 1$ for $\mathbf{C}_2$.
- Can use least-squares to predict $\hat{y}_*$

$$\hat{y} = \begin{cases} \mathbf{C}_1 & \hat{y}_* < 0.5 \\ \mathbf{C}_2 & \hat{y}_* \ge 0.5 \end{cases}$$

- **Logistic function**

$$\sigma(x) = \frac{\exp(x)}{1 + \exp(x)}$$

$$p(y_n = \mathbf{C}_1|\mathbf{x}_n) = \sigma(\boldsymbol{x}^T\boldsymbol{w})$$

$$p(y_n = \mathbf{C}_2|\mathbf{x}_n) = 1 - \sigma(\boldsymbol{x}^T\boldsymbol{w})$$

- The probabilistic model:

$$p(\mathbf{y}|\mathbf{X},\boldsymbol{w}) = \prod_{n=1}^{N}\sigma(\boldsymbol{x}_n^T\boldsymbol{w})^{y_n}(1 - \sigma(\boldsymbol{x}_n^T\boldsymbol{w}))^{1-y_n}$$

- The log-likelihood:

$$\mathcal{L}_{MLE}(\boldsymbol{w}) = \sum_{n=1}^{N}\left(y_n\boldsymbol{x}_n^T\boldsymbol{w} - \log(1 + \exp(\boldsymbol{x}_n^T\boldsymbol{w}))\right)$$

- We can use the fact that

$$\frac{d}{dx}\log(1 + \exp(x)) = \sigma(x)$$

- Gradient of the log-likelihood

$$\mathbf{g} = \frac{d\mathcal{L}}{d\boldsymbol{w}} = \sum_{n=1}^{N}\left(\boldsymbol{x}_n y_n - \boldsymbol{x}_n\sigma(\boldsymbol{x}_n^T\boldsymbol{w})\right)$$

$$= -\mathbf{X}^T[\sigma(\mathbf{X}\boldsymbol{w}) - \mathbf{y}]$$

- The negative of the log-likelihood $-\mathcal{L}_{mle}(\boldsymbol{w})$ is convex
- **Hessian** of the log-likelihood
  - We know that

$$\frac{d\sigma(t)}{dt} = \sigma(t)(1 - \sigma(t))$$

  - Hessian is the derivative of the gradient

$$\mathbf{H}(\boldsymbol{w}) = -\frac{d\mathbf{g}(\boldsymbol{w})}{d\boldsymbol{w}^T} = \sum_{n=1}^{N}\frac{d}{d\boldsymbol{w}^T}\sigma(\boldsymbol{x}_n^T\boldsymbol{w})\boldsymbol{x}_n$$

$$= \sum_{n=1}^{N}\boldsymbol{x}_n\sigma(\boldsymbol{x}_n^T\boldsymbol{w})(1 - \sigma(\boldsymbol{x}_n^T\boldsymbol{w}))\boldsymbol{x}_n^T$$

$$= \tilde{\mathbf{X}}^T\mathbf{S}\tilde{\mathbf{X}}$$

  where $\mathbf{S}$ is a $N \times N$ diagonal matrix with diagonals

$$S_{nn} = \sigma(\boldsymbol{x}_n^T\boldsymbol{w})(1 - \sigma(\boldsymbol{x}_n^T\boldsymbol{w}))$$

  The negative of the log-likelihood is not strictly convex.
- **Newton's Method**
  - Uses second-order information and takes steps in the direction that minimizes a quadratic approximation

$$\mathcal{L}(\boldsymbol{w}) = \mathcal{L}(\boldsymbol{w}^{(k)}) + \nabla\mathcal{L}_k^T(\boldsymbol{w} - \boldsymbol{w}^{(k)})$$

$$+\frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(k)})^T\mathbf{H}_k(\boldsymbol{w} - \boldsymbol{w}^{(k)})$$

  so

$$\nabla\mathcal{L}(w) = \nabla\mathcal{L}(w^{(k)}) + H_k(w - w^{(k)})$$

  equating to 0 gives the minimum:

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^{(k)} - \gamma_k\mathbf{H}_k^{-1}\nabla\mathcal{L}_k$$

  - γ_k added because it's all approximative.
  - Complexity: $O((ND^2 + D^3)I)$
- **Penalized Logistic Regression**

$$\min_{\boldsymbol{w}}\left(-\sum_{n=1}^{N}\log p(y_n|\mathbf{x}_n^T\boldsymbol{w}) + \lambda\sum_{d=1}^{D}w_d^2\right)$$

9 Generalized Linear Model

Exponential family distribution

$$p(\mathbf{y}|\boldsymbol{\eta}) = h(y)\exp(\boldsymbol{\eta}^T\boldsymbol{\phi}(\mathbf{y}) - A(\boldsymbol{\eta}))$$

$$A(\eta) = ln[\int_y h(y)e^{\eta^T\phi(y)}dy] < \infty$$

is convex by Holder.

- Bernoulli distribution

$$p(y|\mu) = \mu^y(1-\mu)^{1-y}$$

$$= \exp(y\log(\frac{\mu}{1-\mu} + \log(1-\mu)))$$

- there is a relationship between η and μ throught the **link function**

$$\eta = \log(\frac{\mu}{1-\mu}) \leftrightarrow \mu = \frac{e^\eta}{1 + e^\eta}$$

- Note that μ is the mean parameter of y
- Relationship between the mean $\boldsymbol{\mu}$ and $\boldsymbol{\eta}$ is defined using a link function g

$$\boldsymbol{\eta} = \mathbf{g}(\boldsymbol{\mu}) \Leftrightarrow \boldsymbol{\mu} = \mathbf{g}^{-1}(\boldsymbol{\eta})$$

- First and second derivatives of $A(\eta)$ are related to the mean and the variance

$$\frac{dA(\eta)}{d\eta} = \mathbb{E}[\boldsymbol{\phi}(\eta)], \quad \frac{d^2A(\eta)}{d\eta^2} = \text{Var}[\boldsymbol{\phi}(\eta)]$$

- $A(\eta)$ is convex
- The generalized maximum likelihood cost to minimize is

$$\min_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}) = -\sum_{n=1}^{N}\log(p(y_n|\boldsymbol{x}_n^T\boldsymbol{w}))$$

  where $p(y_n|\boldsymbol{x}_n^T\boldsymbol{w})$ is an exponential family distribution
- We obtain the solution

$$\frac{d\mathcal{L}}{d\boldsymbol{w}} = \mathbf{X}^T[\mathbf{g}^{-1}(\boldsymbol{X}w) - \boldsymbol{\phi}(\mathbf{y})]$$

  since $\frac{dA(\eta)}{d\eta} = g^{-1}(\eta)$.

10 k-Nearest Neighbor (k-NN)

- The k-NN prediction for $\mathbf{x}$ is

$$f(\mathbf{x}) = \frac{1}{k}\sum_{\mathbf{x}_n\in nbh_k(\mathbf{x})}y_n$$

  where $nbh_k(\mathbf{x})$ is the neighborhood of $\mathbf{x}$ defined by the k closest points $\mathbf{x}_n$. For classification take majority vote instead of mean.
- **Curse of dimensionality**: Let $\mathcal{X} = [0,1]^d$, $\mathcal{Y} = \{0,1\}$, $S \sim \mathcal{D}^N$, $\eta(x) = p(y = 1 \mid x)$. If $\mathcal{D}$ is known, best classifier is Bayes $f_*(x) = \mathbf{1}[\eta(x) > \frac{1}{2}]$. If

$$|\eta(x) - \eta(x')| \le c||x - x'||$$

  then, for f_S the 1-nearest neighbor classifier, $\mathbb{E}[\mathcal{L}(f_S)] \le 2\mathcal{L}(f_*) + c\mathbb{E}_{S,x\sim\mathcal{D}}[||x - nbh_1(x)||]$

$$\le 2\mathcal{L}(f_*) + 4c\sqrt{d}N^{\frac{-1}{d+1}}$$

11 Support Vector Machine

- Combination of the kernel trick plus a modified loss function (Hinge loss)
- Solution to the dual problem is sparse and non-zero entries will be our **support vectors**.
- **Kernelised feature vector** where $\boldsymbol{\mu}_k$ are centroids

$$\boldsymbol{\phi}(\mathbf{x}) = [k(\mathbf{x},\boldsymbol{\mu}_1), ..., k(\mathbf{x},\boldsymbol{\mu}_K)]$$

- In practice we'll take a subset of data points to be prototype $\rightarrow$ **sparse vector machine**.
- Assume $y_n \in \{-1, 1\}$
- SVM optimizes the following cost

$$\mathcal{L}(\boldsymbol{w}) = \min_{\boldsymbol{w}}\sum_{n=1}^{N}[1 - y_n\tilde{\boldsymbol{\phi}}_n^T\boldsymbol{w}]_+ + \frac{\lambda}{2}\sum_{j=1}^{M}w_j^2$$

- Loss is convex but not diff. Can be optim with subgradient descent.
- choose the hyperplane so that the distance from it to the nearest data point on each side is maximized

- **Duality**:
  - Hard to minimize $g(\boldsymbol{w})$ so we define
    $$\mathcal{L}(\boldsymbol{w}) = \max_{\boldsymbol{\alpha}} G(\boldsymbol{w}, \boldsymbol{\alpha})$$
  - we use the property that
    $$C[v_n]_+ = \max(0, Cv_n) = \max_{\alpha_n \in [0,C]} \alpha_n v_n$$
  - We can rewrite the problem as
    $$\min_{\boldsymbol{w}} \max_{\boldsymbol{\alpha} \in [0,C]^N} \sum_{n=1}^{N} \alpha_n (1 - y_n \boldsymbol{\phi}_n^T \boldsymbol{w}) + \frac{1}{2} \sum_{j=1}^{M} w_j$$
  - This is differentiable, convex in $\boldsymbol{w}$ and concave in $\boldsymbol{\alpha}$
  - **Minimax theorem**:
    $$\min_{\boldsymbol{w}} \max_{\boldsymbol{\alpha}} G(\boldsymbol{w}, \boldsymbol{\alpha}) = \max_{\boldsymbol{\alpha}} \min_{\boldsymbol{w}} G(\boldsymbol{w}, \boldsymbol{\alpha})$$
    because G is convex in $\boldsymbol{w}$ and concave in $\boldsymbol{\alpha}$.
  - Derivative w.r.t. $\boldsymbol{w}$:
    $$\boldsymbol{\nabla}_{\boldsymbol{w}} G(\boldsymbol{w}, \boldsymbol{\alpha}) = -\sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n + \lambda \boldsymbol{w}$$
  - Equating this to 0, we get:
    $$\boldsymbol{w}(\boldsymbol{\alpha}) = \frac{1}{\lambda} \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n = \frac{1}{\lambda} \mathbf{X} \mathbf{Y} \boldsymbol{\alpha}$$
    $$\mathbf{Y} := \text{diag}(\boldsymbol{y})$$
  - Plugging $\boldsymbol{w}^*$ back in the dual problem
    $$\max_{\boldsymbol{\alpha} \in [0,1]^N} \boldsymbol{\alpha}^T \mathbf{1} - \frac{1}{2\lambda} \boldsymbol{\alpha}^T \mathbf{Y}^T \boldsymbol{X}^T \boldsymbol{X} \mathbf{Y} \boldsymbol{\alpha}$$
  - This is a differentiable least-squares problem. Optimization is easy using Sequential Minimal Optimization. It is also naturally kernelized with $\mathbf{K} = \boldsymbol{X}^T \boldsymbol{X}$
  - The solution $\boldsymbol{\alpha}$ is sparse and is non-zero only for the training examples that are instrumental in determining the decision boundary.
  - $\alpha_n = 0$ if x_n i far from the boundary in the right side. $\alpha_n \in (0, 1)$ if x_n is exactly on the margin and $\alpha_n = 1$ if x_n is in the margin or on the wrong side.

12 Kernel Ridge Regression

- The following is true for ridge regression
  $$\begin{aligned} \boldsymbol{w}^* &= (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_D)^{-1} \mathbf{X}^T \mathbf{y} \qquad (1) \\ &= \mathbf{X}^T (\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_N)^{-1} \mathbf{y} = \mathbf{X}^T \boldsymbol{\alpha}^* \end{aligned}$$
  since $(PQ + I_N)^{-1} P = P(QP + I_M)^{-1}$.
- Complexity of computing $\boldsymbol{w}$: (1) $O(D^2 N + D^3)$, (2) $O(DN^2 + N^3)$
- Thus we have
  $\boldsymbol{w}^* = \mathbf{X}\boldsymbol{\alpha}^*, \quad$ with $\boldsymbol{w}^* \in \mathbb{R}^D$ and $\boldsymbol{\alpha}^* \in \mathbb{R}^N$
- The representer theorem allows us to write an equivalent optimization problem in terms of $\boldsymbol{\alpha}$.
  $$\boldsymbol{\alpha} = \operatorname*{argmax}_{\boldsymbol{\alpha}} \left(-\frac{1}{2} \boldsymbol{\alpha}^T (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_N) \boldsymbol{\alpha} + \boldsymbol{\alpha}^T \mathbf{y} \right)$$
- $\mathbf{K} = \mathbf{X}\mathbf{X}^T$ is called the **kernel matrix** or **Gram matrix**.
- If $\mathbf{K}$ is positive definite, then it's called a **Mercer Kernel**.
- $\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$
- If the kernel is Mercer, then there exists a function $\boldsymbol{\phi}(\mathbf{x})$ s.t.
  $$k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x}')$$
- **Kernel trick**:
  - compute dot-product in $\mathbb{R}^m$ while remaining in $\mathbb{R}^n$
  - Replace $\langle \mathbf{x}, \mathbf{x}' \rangle$ with $k(\mathbf{x}, \mathbf{x}')$.
- **Common Kernel**
  - Polynomial Kernel: $(\gamma \langle \mathbf{x}_i, \mathbf{x}_j \rangle + r)^d$
  - Radial Basis function kernel (RBF)
    $$k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{1}{2} (\mathbf{x} - \mathbf{x}')^T (\mathbf{x} - \mathbf{x}'))$$
  - Sigmoid Kernel: $\tanh(\langle \mathbf{x}_i, \mathbf{x}_j \rangle + r)$
- **Properties of kernels** to ensure the existance of a corresponding $\boldsymbol{\phi}$:
  - symmetric: $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$
  - positive semi-definite.
- Thus we get
  $$\mathbf{y} = \boldsymbol{w}^T \mathbf{x} = \sum_{i=1}^{K} \alpha_i \mathbf{x}_i^T \mathbf{x} = \sum_{i=1}^{K} \alpha_i k(\mathbf{x}, \mathbf{x}_i)$$

13 K-means

- **Unsupervised learning**: Represent particular input patterns in a way that reflects the statistical structure of the overall collections of input partterns.
- **Cluster** are groups of points whose inter-point distances are small compared to the distances outside the cluster.
  $$\min_{\mathbf{z}, \boldsymbol{\mu}} \mathcal{L}(\mathbf{z}, \boldsymbol{\mu}) = \sum_{k=1}^{K} \sum_{n=1}^{N} z_{nk} ||\mathbf{x}_n - \boldsymbol{\mu}_k||_2^2$$
  such that $z_{nk} \in \{0, 1\}$ and $\sum_{k=1}^{K} z_{nk} = 1$
- K-means algorithm (Coordinate Descent): Initialize $\boldsymbol{\mu}_k$, then iterate
  1. For all n, compute $\mathbf{z}_n$ given $\boldsymbol{\mu}$
     $$z_{nk} = \begin{cases} 1 & \text{if } k = \operatorname{argmin}_j ||\mathbf{x}_n - \boldsymbol{\mu}||_2^2 \\ 0 & \text{otherwise} \end{cases}$$
  2. For all k, compute μ_k given $\mathbf{z}$
     $$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N} z_{nk} \mathbf{x}_n}{\sum_{n=1}^{N} z_{nk}}$$
- A good initialization procedure is to choose the prototypes to be equal to a random subset of K data points.
- Probabilistic model
  $$p(\mathbf{z}, \boldsymbol{\mu}) = \prod_{n=1}^{N} \prod_{k=1}^{K} [\mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \mathbf{I})]^{z_{nk}}$$
- K-means as a Matrix Factorization
  $$\min_{\mathbf{z}, \boldsymbol{\mu}} \mathcal{L}(\mathbf{z}, \boldsymbol{\mu}) = ||\mathbf{X} - \mathbf{M}\mathbf{Z}^T||_{\text{Frob}}^2$$
- Computation can be heavy, each example can belong to only on cluster and clusters have to be spherical.

14 Gaussian Mixture Models

- Clusters can be elliptical using a full covariance matrix instead of isotropic covariance.
  $$p(\mathbf{X} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \mathbf{z}) = \prod_{n=1}^{N} \prod_{k=1}^{K} [\mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \Sigma_k)]^{z_{nk}}$$
- **Soft-clustering**: Points can belong to several cluster by defining z_n to be a random variable.
  $$p(z_n = k) = \pi_k \text{ where } \pi_k > 0, \forall k, \sum_{k=1}^{K} \pi_k = 1$$
- Joint distribution of Gaussian mixture model
  $$\begin{aligned} p(\mathbf{X}, \mathbf{z} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) &= \prod_{n=1}^{N} p(\mathbf{x}_n | \mathbf{r}_n, \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(\mathbf{z}_n | \boldsymbol{\pi}) \\ &= \prod_{n=1}^{N} \prod_{k=1}^{K} [(\mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k))^{z_{nk}}] \prod_{k=1}^{K} [\pi]^{z_{nk}} \end{aligned}$$
- z_n are called *latent* unobserved variables
- Unknown parameters are given by $\boldsymbol{\theta} = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}\}$
- We get the **marginal likelihood** by marginalizing z_n out from the likelihood
  $$\begin{aligned} p(\mathbf{x}_n | \boldsymbol{\theta}) &= \sum_{k=1}^{K} p(\mathbf{x}_n, z_n = k | \boldsymbol{\theta}) \\ &= \sum_{k=1}^{K} p(z_n = k | \boldsymbol{\theta}) p(\mathbf{x}_n | z_n = k, \boldsymbol{\theta}) \\ &= \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \end{aligned}$$
- Without a latent variable model, number of parameters grow at rate $O(N)$
- After marginalization, the growth is reduced to $O(D^2 K)$
- To get maximum likelihood estimate of $\boldsymbol{\theta}$, we maximize
  $$\max_{\boldsymbol{\theta}} \sum_{n=1}^{N} \log \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

15 Expectation Maximization Algorithm

- *[ALGORITHM]* Start with $\boldsymbol{\theta}^{(1)}$ and iterate
  1. *Expectation step*: Compute a lower bound to the cost such that it is tight at the previous $\boldsymbol{\theta}^{(t)}$ with equality when,
     $$q_{kn} = \frac{\pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}$$
     this is the posterior of $z_n = k$ given x_n, θ.
  2. *Maximization step*: Update $\boldsymbol{\theta}$
     $$\boldsymbol{\theta}^{(t+1)} = \operatorname*{argmax}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)})$$
     $$\boldsymbol{\mu}_k^{(t+1)} = \frac{\sum_{n=1}^{N} \gamma^{(i)}(r_{nk}) \mathbf{x}_n}{\sum_{n=1}^{N} q_{kn}^{(t)}}$$
     $$\boldsymbol{\Sigma}_k^{(t+1)} = \frac{\sum_{n=1}^{N} q_{kn}^{(t)} (\mathbf{x}_n - \boldsymbol{\mu}_k^{(t+1)})(\mathbf{x}_n - \boldsymbol{\mu}_k^{(t+1)})^T}{\sum_{n=1}^{N} q_{kn}^{(t)})}$$
     $$\pi_k^{(t+1)} = \frac{1}{N} \sum_{n=1}^{N} q_{kn}^{(t)}$$
- If covariance is diagonal $\rightarrow$ K-means.

16 Matrix factorization

- We have D movies and N users
- $\mathbf{X}$ is a matrix $D \times N$ with x_{dn} the rating of n'th user for d'th movie.
- We project data vectors $\mathbf{x}_n$ to a smaller dimension $\mathbf{z}_n \in \mathbb{R}^M$
- We have now 2 latent variables:
  - $\mathbf{Z}$ a $N \times K$ matrix that gives features for the users
  - $\mathbf{W}$ a $D \times K$ matrix that gives features for the movies
  $$x_{dn} \approx \mathbf{w}_d^T \mathbf{z}_n$$
- Non-convex and non-identifiable $(W = \beta W, Z = \frac{1}{\beta} Z)$.
- We can add a regularizer and minimize the following cost:
  $$\begin{aligned} \mathcal{L}(\mathbf{W}, \mathbf{Z}) = \frac{1}{2} \sum_{(d,n) \in \Omega} [x_{dn} - (\mathbf{W}\mathbf{Z}^T)_{dn}]^2 \\ + \frac{\lambda_w}{2} \sum_{d=1}^{D} \mathbf{w}_d^T \mathbf{w}_d + \frac{\lambda_z}{2} \sum_{n=1}^{N} \mathbf{z}_n^T \mathbf{z}_n \end{aligned}$$
- We can use coordinate descent algorithm, by first minimizing w.r.t. $\mathbf{Z}$ given $\mathbf{W}$ and then minimizing $\mathbf{W}$ given $\mathbf{Z}$. This is called **Alternating least-squares (ALS)**:
  $$\mathbf{Z}^T \leftarrow (\mathbf{W}^T \mathbf{W} + \lambda_z \mathbf{I}_K)^{-1} \mathbf{W}^T \mathbf{X}$$
  $$\mathbf{W}^T \leftarrow (\mathbf{Z}^T \mathbf{Z} + \lambda_w \mathbf{I}_K)^{-1} \mathbf{Z}^T \mathbf{X}^T$$
- *Complexity*: $O(DNK^2 + NK^3) \rightarrow O(DNK^2)$
- Probabilistic model
  $$\prod_{(d,n) \in \Omega} \mathcal{N}(x_{dn} | \mathbf{w}_d^T \mathbf{z}_n, I) \times \prod_{n=1}^{N} \mathcal{N}(\mathbf{z}_n | 0, \frac{1}{\lambda_z} I)$$
  $$\times \prod_{d=1}^{D} \mathcal{N}(\mathbf{w}_d | 0, \frac{1}{\lambda_w} I)$$
- Since many ratings are missing we cannot normalize the data. A solution is to add offset terms:
  $$\frac{1}{2} \sum_{(d,n) \in \Omega} (x_{dn} - \mathbf{w}_d^T \mathbf{z}_n - w_{0d} - z_{0n} - \mu)^2$$

17 Singular Value Decomposition

- Matrix factorization method
  $$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$
  - $\mathbf{U}$ is a unitary $D \times D$ matrix
  - $\mathbf{V}$ is a unitary $N \times N$ matrix
  - $\mathbf{S}$ is a non-negative diagonal matrix of size $D \times N$ which are called **singular values** appearing in a descending order.
  - Columns of $\mathbf{U}$ and $\mathbf{V}$ are the left and right **singular vectors** respectively.
- Assuming $D < N$ we have
  $$\mathbf{X} = \sum_{d=1}^{D} s_d \mathbf{u}_d \mathbf{v}_d^T$$
  This tells you about the spectrum of $\mathbf{X}$ where higher singular vectors contain the *low-frequency information* and lower singular values contain the *high-frequency information.*
- Dimensionality Reduction
  Take the matrix $\mathbf{S}^{(K)}$ with the K first diagonal elements non zero. Then, rank-K approx:
  $$\mathbf{X} \approx \mathbf{X}_K = \mathbf{U}\mathbf{S}^{(K)}\mathbf{V}^T$$

17.1 Principal Componement Analysis

- PCA is a dimensionality reduction method and a method to decorrelate the data $\mathbf{X} \approx \tilde{\mathbf{X}} = \mathbf{W}\mathbf{Z}^T$ such that columns of $\mathbf{W}$ are orthogonal.
- If the data is zero mean
  $$\boldsymbol{\Sigma} = \frac{1}{N} \mathbf{X}\mathbf{X}^T \Rightarrow \mathbf{X}\mathbf{X}^T = \mathbf{U}\mathbf{S}^2\mathbf{U}^T$$
  $$\Rightarrow \mathbf{U}^T \mathbf{X}\mathbf{X}^T \mathbf{U} = \mathbf{U}^T \mathbf{U}\mathbf{S}^2\mathbf{U}^T \mathbf{U} = \mathbf{S}^2$$
- Thus the columns of matrix $\mathbf{U}$ are called the **principal components** and they decorrelate the covariance matrix.
- Using SVD, we can compute the matrices in the following way
  $$\mathbf{W} = \mathbf{U}\mathbf{S}_D^{1/2}, \mathbf{Z}^T = \mathbf{S}^{1/2}\mathbf{V}^T$$

18 Neural Net

- Basic structure: One *input* layer of size D, L *hidden* layers of size K, and one *output* layer. (*feedforward* network).
  $x_j^{(l)} = \phi\left(\sum_i w_{i,j}^{(l)} x_i^{(l-1)} + b_j^{(l)}\right)$.
- Lemma: For $f : \mathbb{R}^D \rightarrow \mathbb{R}$ with $\int |w||\tilde{f}(w)|dw \leq C$ with
  $\tilde{f}(w) = \int f(x) e^{-iw^T x} dx$ is the Fourier transform. Then $\forall n > 0$,
  $\exists f_n(x) = \sum_{i=1}^{n} c_i \sigma(x^t w_i + b_i) + c_0$ so that
  $\int_{|x|<r} (f - f_n)^2 \leq \frac{(2Cr)^2}{n}$, with σ sigmoid-like.
- Cost function:
  $\mathcal{L} =$
  $\frac{1}{N} \sum_{n=1}^{N} \left(y_n - f^{(L+1)} \circ \ldots \circ f^{(1)}(\boldsymbol{x}_n^{(0)})\right)^2$
  We can use SGD to minimize the cost function.
- Compact form: $\boldsymbol{W}_{i,j}^{(l)} = w_{i,j}^{(l)}$
  $\boldsymbol{x}^{(l)} = f^{(l)}(\boldsymbol{x}^{(l-1)}) =$
  $\phi\left(\left(\boldsymbol{W}^{(l)}\right)^T \boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)}\right)$

18.1 Backpropagation Algorithm

- *Forward pass*: Compute
  $\boldsymbol{z}^{(l)} = \left(\boldsymbol{W}^{(l)}\right)^T \boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)}$ with
  $\boldsymbol{x}^{(0)} = \boldsymbol{x}_n$ and $\boldsymbol{x}^{(l)} = \phi(\boldsymbol{z}^{(l)})$.
- *Backward pass*: Set
  $\delta^{(L+1)} = -2(y_n - \boldsymbol{x}^{(L+1)}) \phi'(z^{(L+1)})$ (if squared loss). Then compute
  $$\begin{aligned} \delta_j^{(l)} = \frac{\partial \mathcal{L}_n}{\partial z_j^{(l)}} &= \sum_k \frac{\partial \mathcal{L}_n}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} \\ &= \sum_k \delta_k^{(l+1)} \boldsymbol{W}_{j,k}^{(l+1)} \phi'(z_j^{(l)}) \end{aligned}$$
- *Final Computation*:
  $$\begin{aligned} \frac{\partial \mathcal{L}_n}{\partial w_{i,j}^{(l)}} = \sum_k \frac{\partial \mathcal{L}_n}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial w_{i,j}^{(l)}} &= \frac{\partial \mathcal{L}_n}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{i,j}^{(l)}} \\ &= \delta_j^{(l)} \boldsymbol{x}_i^{(l-1)} \end{aligned}$$
  $$\begin{aligned} \frac{\partial \mathcal{L}_n}{\partial b_j^{(l)}} = \sum_k \frac{\partial \mathcal{L}_n}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial b_j^{(l)}} &= \frac{\partial \mathcal{L}_n}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} \\ &= \delta_j^{(l)} \cdot 1 = \delta_j^{(l)} \end{aligned}$$

18.2 Activation Functions

Sigmoid $\phi(x) = \frac{1}{1+e^{-x}}$ Positive, bounded.
$\phi'(x) \simeq 0$ for large $|x| \Rightarrow$ Learning slow.
Tanh $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\phi(2x) - 1$.
Balanced, bounded. Learning slow too.
ReLU $(x)_+ = \max\{0, x\}$ Positive, unbounded.
Derivate = 1 if $x > 0$, 0 if $x < 0$
Leaky ReLU $f(x) = \max\{\alpha x, x\}$ Remove 0 derivative.
Maxout
$f(x) = max\{\boldsymbol{x}^T \boldsymbol{w}_1 + b_1, ..., \boldsymbol{x}^T \boldsymbol{w}_k + b_k\}$
(Generalization of ReLU)

18.3 Convolutional NN

Sparse connections and *weights sharing*: reduce complexity. (e.g. pixels in pictures only depend on neighbours)

18.4 Reg, Data Augmentation and Dropout

- Regularization term: $\frac{1}{2} \sum_{l=1}^{L+1} \mu^{(l)} ||W^{(l)}||_F^2$
- Data Augm.: e.g. shift or rotation of pics
- Dropout: avoid overfit. Drop nodes randomly. (Then average multiple drop-NN)

19 Bayes Net

- Graph example: $p(x, y, z) = p(y|x)p(z|x)p(x)$: $(y \leftarrow x \rightarrow z)$
- Lemma: X and Y are independent given Z iff X and Y are D-separated by Z.
- **D-Separation** X and Y are D-separated by Z if every path from $x \in X$ to $y \in Y$ is blocked by Z.
- **Blocked Path** contains a variable that
  - is in Z and is **head-to-tail** $(x \rightarrow z \rightarrow y)$ or **tail-to-tail** $(x \leftarrow z \rightarrow y)$.
  - the node is **head-to-head** $(x \rightarrow a \leftarrow y)$ and neither the node nor the descendant are in Z.
- **Markov Blanket** (which blocks node A from the rest of the net) contains:
  - parents of A
  - children of A
  - parents of children of A
- Lemma: Any x is independent of $y \notin MB(X)$ given $MB(x)$.
- $f(x_1, .., x_6) =$
  $f_1(x_1, x_2, x_3) f_2(x_1, x_4, x_6) f_3(x_4) f_4(x_4, x_5)$,
  then $f(x_1) =$
  $[\sum_{x_2,x_3} f_1(x_1, x_2, x_3)][\sum_{x_4} f_3(x_4)(\sum_{x_6} f_2(x_1,$