

PCML Cheat Sheet

1 Math Prerequisites

- Bayes rule

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

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

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

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

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

- A function $f(x)$ is convex, if for any $x_1, x_2 \in \mathbf{X}$ and for any $0 \leq \lambda \leq 1$, we have :

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2)$$

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

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

1.2 Linear Algebra

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

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

- A positive definite matrix is **symmetric** with all positive eigenvalues
- The real symmetric $N \times N$ matrix \mathbf{V} is said to be **positive semidefinite** if

$$\mathbf{a}^T \mathbf{V} \mathbf{a} \geq 0$$

for any real $N \times 1$ vector \mathbf{a} .

- positive definite** if $\mathbf{a}^T \mathbf{V} \mathbf{a} > 0$

2 Cost functions

- Cost functions are used to learn parameters that explain the data well.
- It is essential to make sure that a global minimum exist \rightarrow lower bounded

Mean square error (MSE):

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

- MSE is **convex** thus it has only one global minimum value.
- MSE is not good when outliers are present.

Mean Absolute Error (MAE):

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

Huber loss

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

- Huber loss is convex, differentiable, and also robust to outliers but hard to set δ .

Tukey's bisquare loss

$$L(z) = \begin{cases} (z^2 - z^2)^2 & , |z| < \delta \\ 0 & , |z| \geq \delta \end{cases}$$

Non-convex, non-diff., but robust to outliers.

Hinge loss

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

Logistic loss

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

3 Regression

- Data** consists of N pairs (y_n, \mathbf{x}_n)

- y_n the n 'th output
 - \mathbf{x}_n is a vector of D inputs
- Prediction:** predict the output for a new input vector.
 - Interpretation:** understand the effect of inputs on output.
 - Outliers** are data that are far away from most of the other examples.

3.1 Linear Regression

- Model that assume linear relationship between inputs and the output.

$$y_n \approx f(\mathbf{x}_n) := w_0 + w_1 x_{n1} + \dots = w_0 + \mathbf{x}_n^T \mathbf{w}$$

- with \mathbf{w} the parameters of the model.

- Variance grows only linearly with dimensionality

4 Optimization

4.1 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

4.2 Gradient Descent

- GD uses only first-order information and takes steps in the opposite direction of the gradient

- Given cost function $\mathcal{L}(\mathbf{w})$ we want to find \mathbf{w}
 $\mathbf{w} = \arg \min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$

4.3 Batch Gradient Descent

- Take steps in the opposite direction of the gradient

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

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

4.4 Gradients for MSE

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

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

- and MSE as follows:

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

- then the gradient is given by

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

- Optimality conditions:

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

4.5 Stochastic Gradient Descent

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

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

\Rightarrow SGD algo is given by update rule:

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

Idea: Cheap but unbiased estimate of grad.

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

4.6 Mini-batch SGD

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

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

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

4.7 Subgradients (Non-Smooth OPT)

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

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

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

5 Least Squares

- Use the first optimality conditions:
 $\nabla \mathcal{L}(\mathbf{w}^*) = 0 \Rightarrow \mathbf{X}^T \mathbf{e} = \mathbf{X}^T (\mathbf{y} - \mathbf{X} \mathbf{w}) = 0$

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

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

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

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

- 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 columns $\tilde{\mathbf{x}}_d$ are nearly collinear. \Rightarrow matrix is ill-conditioned.

6 Maximum Likelihood

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

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

- Another way of expressing this:

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

which defines the likelihood of observing \mathbf{y} given \mathbf{X} and \mathbf{w}

- Define cost with log-likelihood

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

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

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

$$\arg \min_{\mathbf{w}} \mathcal{L}_{MSE}(\mathbf{w}) = \arg \max_{\mathbf{w}} \mathcal{L}_{lik}(\mathbf{w})$$

- 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

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

$$\sum_n \log p(y_n|\mathbf{x}_n, \mathbf{w}) = \sum_n |y_n - \mathbf{x}_n^T \mathbf{w}| + c n s t$$

7 Ridge Regression

- Linear models usually overfit. One way is to use nonlinear basis functions instead.

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

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

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

- The least square solution becomes

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

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

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

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \left(\frac{1}{2} (\mathbf{y} - \mathbf{X} \mathbf{w})^T (\mathbf{y} - \mathbf{X} \mathbf{w}) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w} \right)$$

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

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

$$\boldsymbol{\Lambda} = \begin{bmatrix} 0 & \mathbf{0} \\ \mathbf{0} & \lambda I_m \end{bmatrix}$$

- 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
- Maximum-a-posteriori (MAP) estimator:**

- Maximizes the product of the likelihood and the **prior**.

$$\mathbf{w}_{MAP} = \arg \max_{\mathbf{w}} (p(\mathbf{y}|\mathbf{X}, \boldsymbol{\Lambda}) p(\mathbf{w}|\boldsymbol{\Sigma}))$$

- Assume $w_0 = 0$

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

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

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

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

8 Bias-Variance decomposition

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

- Squared bias**: The average *shift* of the predictions
- Variance**: measure how data points vary around their average.

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

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

9 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}_*

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

- Logistic function**

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

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

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

- The probabilistic model:

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

- The log-likelihood:

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

- We can use the fact that

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

- Gradient of the log-likelihood

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

- The negative of the log-likelihood $-\mathcal{L}_{MLE}(\mathbf{w})$ 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

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

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

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

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

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

- 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

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

$$+ (\mathbf{w} - \mathbf{w}^{(k)})^T \mathbf{H}_k (\mathbf{w} - \mathbf{w}^{(k)})$$

and it's minimum is at $\mathbf{w}^{k+1} = \mathbf{w}^{(k)} - \gamma_k \mathbf{H}_k^{-1} \nabla \mathcal{L}_k$

- Complexity: $O((ND^2 + D^3)I)$

Penalized Logistic Regression

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

10 Generalized Linear Model

Exponential family distribution

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

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

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

there is a relationship between $\boldsymbol{\eta}$ and $\boldsymbol{\mu}$ through the **link function**

$$\boldsymbol{\eta} = \log\left(\frac{\mu}{1-\mu}\right) \Leftrightarrow \boldsymbol{\mu} = \frac{e^{\boldsymbol{\eta}}}{1 + e^{\boldsymbol{\eta}}}$$

- Note that $\boldsymbol{\mu}$ 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 $A(\boldsymbol{\eta})$ are related to the mean and the variance

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

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

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

where $p(y_n|\mathbf{x}_n^T \$

11 **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 .

- **Curse of dimensionality:** Generalizing correctly becomes exponentially harder as the dimensionality grows.
- Gathering more inputs variables may be bad

12 **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 μ_k are centroids

$$\phi(\mathbf{x}) = [k(\mathbf{x}, \mu_1), \dots, k(\mathbf{x}, \mu_K)]$$

- 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

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

- Minimum doesn't change with a rescaling of \mathbf{w}
- choose the hyperplane so that the distance from it to the nearest data point on each side is maximized

- **Duality:**

- Hard to minimize $g(\mathbf{w})$ so we define

$$\mathcal{L}(\mathbf{w}) = \max_{\alpha} G(\mathbf{w}, \alpha)$$

- 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

$$\min_{\alpha} \max_{\alpha \in [0, C]^N} \sum_{n=1}^N \alpha_n (1 - y_n \phi_n^T \mathbf{w}) + \frac{1}{2} \sum_{j=1}^M w_j^2$$

- This is differentiable, convex in \mathbf{w} and concave in α

- **Minimax theorem:**

$$\min_{\mathbf{w}} \max_{\alpha} G(\mathbf{w}, \alpha) = \max_{\alpha} \min_{\mathbf{w}} G(\mathbf{w}, \alpha)$$

because G is convex in \mathbf{w} and concave in α .

- Derivative w.r.t. \mathbf{w} :

$$\nabla_{\mathbf{w}} G(\mathbf{w}, \alpha) = - \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n + \lambda \mathbf{w}$$

- Equating this to 0, we get:

$$\mathbf{w}(\alpha) = \frac{1}{\lambda} \sum_{n=1}^N \alpha_n y_n \mathbf{x}_n = \frac{1}{\lambda} \mathbf{X} \mathbf{Y} \alpha$$

$$\mathbf{Y} := \text{diag}(\mathbf{y})$$

- Plugging \mathbf{w}^* back in the dual problem

$$\max_{\alpha \in [0, 1]^N} \alpha^T \mathbf{1} - \frac{1}{2\lambda} \alpha^T \mathbf{Y} \mathbf{X}^T \mathbf{X} \mathbf{Y} \alpha$$

- This is a differentiable least-squares problem. Optimization is easy using Sequential Minimal Optimization. It is also naturally kernelized with $\mathbf{K} = \mathbf{X}^T \mathbf{X}$
- The solution α is sparse and is non-zero only for the training examples that are instrumental in determining the decision boundary.

13 **Kernel Ridge Regression**

- The following is true for ridge regression

$$\begin{aligned} \mathbf{w}^* &= (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_D)^{-1} \mathbf{X}^T \mathbf{y} \\ &= \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I}_N)^{-1} \mathbf{y} = \mathbf{X}^T \alpha^* \end{aligned} \quad (1)$$

- Complexity of computing \mathbf{w} : (1) $O(D^2 N + D^3)$, (2) $O(DN^2 + N^3)$

- Thus we have

$$\mathbf{w}^* = \mathbf{X} \alpha^*, \quad \text{with } \mathbf{w}^* \in \mathbb{R}^D \text{ and } \alpha^* \in \mathbb{R}^N$$

- The representer theorem allows us to write an equivalent optimization problem in terms of α .

$$\alpha = \underset{\alpha}{\text{argmax}} \left(-\frac{1}{2} \alpha^T (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_N) \alpha + \alpha^T \mathbf{y} \right)$$

- $\mathbf{K} = \mathbf{X} \mathbf{X}^T$ 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 $\phi(\mathbf{x})$ s.t.

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

- **Kernel trick:**

- We can work directly with \mathbf{K} and never have to worry about \mathbf{X}
- Replace $(\mathbf{x}, \mathbf{x}')$ with $k(\mathbf{x}, \mathbf{x}')$.
- Kernel function can be interpreted as a measure of similarity
- The evaluation of a kernel is usually faster with k than with ϕ

- Kernelized ridge regression might be computationally more efficient in some cases.

- **Radial Basis function kernel (RBF)**

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^T(\mathbf{x} - \mathbf{x}')\right)$$

- Properties of a kernel to ensure the existence of a corresponding ϕ :

- \mathbf{K} should be symmetric:
 $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$
- \mathbf{K} should be positive semidefinite.

- Thus we get

$$\mathbf{y} = \mathbf{w}^T \mathbf{x} = \sum_{i=1}^K \alpha_i \mathbf{x}_i^T \mathbf{x} = \sum_{i=1}^K \alpha_i k(\mathbf{x}, \mathbf{x}_i)$$

14 **K-means**

Unsupervised learning: Represent particular input patterns in a way that reflects the statistical structure of the overall collections of input patterns.

- **Cluster** are groups of points whose inter-point distances are small compared to the distances outside the cluster.

$$\min_{\mathbf{z}, \mu} \mathcal{L}(\mathbf{z}, \mu) = \sum_{k=1}^K \sum_{n=1}^N z_{nk} \|\mathbf{x}_n - \mu_k\|_2^2$$

- such that $z_{nk} \in \{0, 1\}$ and $\sum_{k=1}^K z_{nk} = 1$
- K-means algorithm (Coordinate Descent): Initialize μ_k , then iterate

1. For all n , compute \mathbf{z}_n given μ
 $z_{nk} = \begin{cases} 1 & \text{if } k = \underset{j}{\text{argmin}} \|\mathbf{x}_n - \mu_j\|_2^2 \\ 0 & \text{otherwise} \end{cases}$

2. For all k , compute μ_k given \mathbf{z}
 $\mu_k = \frac{\sum_{n=1}^N z_{nk} \mathbf{x}_n}{\sum_{n=1}^N z_{nk}}$

- A good initialization procedure is to choose the prototypes to be equal to a random subset of K data points.
- Probabilistic model

$$p(\mathbf{z}, \mu) = \prod_{n=1}^N \prod_{k=1}^K [\mathcal{N}(\mathbf{x}_n | \mu_k, \mathbf{I})]^{z_{nk}}$$

- K-means as a Matrix Factorization
 $\min_{\mathbf{z}, \mu} \mathcal{L}(\mathbf{z}, \mu) = \|\mathbf{X} - \mathbf{M} \mathbf{Z}^T\|_{\text{Frob}}^2$
- Computation can be heavy, each example can belong to only on cluster and clusters have to be spherical.

15 **Gaussian Mixture Models**

- Clusters can be elliptical using a full covariance matrix instead of isotropic

covariance.

$$p(\mathbf{X} | \mu, \Sigma, \mathbf{z}) = \prod_{n=1}^N \prod_{k=1}^K [\mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k)]^{z_{nk}}$$

- **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

$$\begin{aligned} p(\mathbf{X}, \mathbf{z} | \mu, \Sigma, \pi) &= \prod_{n=1}^N p(\mathbf{x}_n | \mathbf{r}_n, \mu, \Sigma) p(\mathbf{z}_n | \pi) \\ &= \prod_{n=1}^N \prod_{k=1}^K [(\mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k))^{z_{nk}}] \prod_{k=1}^K [\pi]^{z_{nk}} \end{aligned}$$

- z_n are called *latent* unobserved variables
- Unknown parameters are given by $\theta = \{\mu, \Sigma, \pi\}$
- We get the **marginal likelihood** by marginalizing z_n out from the likelihood

$$\begin{aligned} p(\mathbf{x}_n | \theta) &= \sum_{k=1}^K p(\mathbf{x}_n, z_n = k | \theta) \\ &= \sum_{k=1}^K p(z_n = k | \theta) p(\mathbf{x}_n | z_n = k, \theta) \\ &= \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k) \end{aligned}$$

- 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 θ , we maximize

$$\max_{\theta} \sum_{n=1}^N \log \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k)$$

16 **Expectation Maximization Algorithm**

- [ALGORITHM] Start with $\theta^{(1)}$ and iterate

1. *Expectation step:* Compute a lower bound to the cost such that it is tight at the previous $\theta^{(t)}$ with equality when,

$$q_{kn} = \frac{\pi_k \mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k)}{\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k)}$$

2. *Maximization step:* Update θ
 $\theta^{(t+1)} = \underset{\theta}{\text{argmax}} \mathcal{L}(\theta, \theta^{(t)})$

$$\mu_k^{(t+1)} = \frac{\sum_{n=1}^N q_{kn}^{(i)} (r_{nk}) \mathbf{x}_n}{\sum_{n=1}^N q_{kn}^{(t)}}$$

$$\Sigma_k^{(t+1)} = \frac{\sum_{n=1}^N q_{kn}^{(t)} (\mathbf{x}_n - \mu_k^{(t+1)}) (\mathbf{x}_n - \mu_k^{(t+1)})^T}{\sum_{n=1}^N q_{kn}^{(t)}}$$

$$\pi_k^{(t+1)} = \frac{1}{N} \sum_{n=1}^N q_{kn}^{(t)}$$

- If covariance is diagonal \rightarrow K-means.

17 **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

$$x_{dn} \approx \mathbf{w}_d^T \mathbf{z}_n$$

- We can add a regularizer and minimize the

following cost:

$$\begin{aligned} \mathcal{L}(\mathbf{W}, \mathbf{Z}) &= \frac{1}{2} \sum_{(d,n) \in \Omega} [x_{dn} - (\mathbf{W} \mathbf{Z}^T)_{dn}]^2 \\ &\quad + \frac{\lambda_w}{2} \sum_{d=1}^D \mathbf{w}_d^T \mathbf{w}_d + \frac{\lambda_z}{2} \sum_{n=1}^N \mathbf{z}_n^T \mathbf{z}_n \end{aligned}$$

- 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:

$$\frac{1}{2} \sum_{(d,n) \in \Omega} (x_{dn} - \mathbf{w}_d^T \mathbf{z}_n - w_{0d} - z_{0n} - \mu)^2$$

18 **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:

$$\mathbf{X} \approx \mathbf{X}_K = \mathbf{U} \mathbf{S}^{(K)} \mathbf{V}^T$$

18.1 **Principal Component 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

$$\Sigma = \frac{1}{N} \mathbf{X} \mathbf{X}^T \Rightarrow \mathbf{X} \mathbf{X}^T = \mathbf{U} \mathbf{S}^2 \mathbf{U}^T$$

$$\Rightarrow \mathbf{U}^T \mathbf{X} \mathbf{X}^T \mathbf{U} = \mathbf{U}^T \mathbf{U} \mathbf{S}^2 \mathbf{U}^T \mathbf{U} = \mathbf{S}^2$$

- 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

$$\mathbf{W} = \mathbf{U} \mathbf{S}_D^{1/2}, \mathbf{Z}^T = \mathbf{S}^{1/2} \mathbf{V}^T$$

19 **Neural Net**

- Basic structure: One *input* layer of size D , L *hidden* layers of size K , and one *output* layer. (*feedforward* network).

$$x_j^{(l)} = \phi \left(\sum_i w_{i,j}^{(l)} x_i^{(l-1)} + b_j^{(l)} \right).$$

- NN can represent the Riemann sum with only two layers \Rightarrow It's powerful!

- Cost function:

$$\begin{aligned} \mathcal{L} &= \frac{1}{N} \sum_{n=1}^N \left(y_n - f^{(L+1)} \circ \dots \circ f^{(1)}(\mathbf{x}_n^{(0)}) \right)^2 \\ \text{We can use SGD to minimize the cost function.} \\ \text{- Compact form: } \mathbf{W}_{i,j}^{(l)} &= w_{i,j}^{(l)} \\ \mathbf{x}^{(l)} &= f^{(l)}(\mathbf{x}^{(l-1)}) = \\ &\phi \left((\mathbf{W}^{(l)})^T \mathbf{x}^{(l-1)} + \mathbf{b}^{(l)} \right) \end{aligned}$$

19.1 **Backpropagation Algorithm**

- *Forward pass:* Compute

$$\mathbf{z}^{(l)} = (\mathbf{W}^{(l)})^T \mathbf{x}^{(l-1)} + \mathbf{b}^{(l)} \text{ with}$$

$$\mathbf{x}^{(0)} = \mathbf{x}_n \text{ and } \mathbf{x}^{(l)} = \phi(\mathbf{z}^{(l)}).$$

- *Backward pass:* Set $\delta^{(L+1)} = -2(y_n - \mathbf{x}^{(L+1)}) \phi'(z^{(L+1)})$ (if squared loss). Then compute

$$\begin{aligned} \delta_j^{(l)} &= \frac{\partial \mathcal{L}_n}{\partial z_j^{(l)}} = \sum_k \frac{\partial \mathcal{L}_n}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} \\ &= \sum_k \delta_k^{(l+1)} \mathbf{W}_{j,k}^{(l+1)} \phi'(z_j^{(l)}) \end{aligned}$$

- *Final Computation:*

$$\frac{\partial \mathcal{L}_n}{\partial w_{i,j}^{(l)}} = \sum_k \frac{\partial \mathcal{L}_n}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial w_{i,j}^{(l)}} = \frac{\partial \mathcal{L}_n}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{i,j}^{(l)}} = \delta_j^{(l)} \mathbf{x}_i^{(l-1)}$$

$$\frac{\partial \mathcal{L}_n}{\partial b_j^{(l)}} = \sum_k \frac{\partial \mathcal{L}_n}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial b_j^{(l)}} = \frac{\partial \mathcal{L}_n}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \cdot 1 = \delta_j^{(l)}$$

19.2 **Activation Functions**

Sigmoid $\phi(x) = \frac{1}{1+e^{-x}}$ Positive, bounded.

$\phi'(x) \simeq 0$ for large $|x| \Rightarrow$ Learning slow.

Tanh $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \phi(2x) - 1/2$.

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

$$f(x) = \max \mathbf{x}^T \mathbf{w}_1 + b_1, \dots, \mathbf{x}^T \mathbf{w}_k + b_k$$

(Generalization of ReLU)

19.3 **Convolutional NN**

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

19.4 **Reg. Data Augmentation and Dropout**

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

20 **Bayes Net**

- Graph example: $p(x, y, z) = p(y|x)p(z|x)p(x) : (y \leftarrow x \rightarrow z)$

- **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** or **tail-to-tail**
- the node is **head-to-head** 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