

ML Cheat Sheet

1 Math Prerequisites

1.1 Derivatives

- $\partial(\mathbf{XY}) = (\partial\mathbf{X})\mathbf{Y} + \mathbf{X}(\partial\mathbf{Y})$
- $\frac{\partial \mathbf{f}(\mathbf{g}(\mathbf{u}(\mathbf{x})))}{\partial \mathbf{x}} = \frac{\partial \mathbf{u}(\mathbf{x})}{\partial \mathbf{x}} \frac{\partial \mathbf{g}(\mathbf{u})}{\partial \mathbf{u}} \frac{\partial \mathbf{f}(\mathbf{g})}{\partial \mathbf{g}}$
- $\frac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$
- $\frac{\partial \mathbf{a}^T \mathbf{X} \mathbf{b}}{\partial \mathbf{X}} = \mathbf{ab}^T$
- $\frac{\partial \mathbf{a}^T \mathbf{X}^T \mathbf{b}}{\partial \mathbf{X}} = \mathbf{ba}^T$
- $\frac{\partial \mathbf{x}}{\partial \mathbf{x}_{ij}} = \mathbf{J}^{ij}, \mathbf{J}^{ij}$ is the single entry matrix
- $\frac{\partial \mathbf{b}^T \mathbf{X}^T \mathbf{X} \mathbf{c}}{\partial \mathbf{X}} = \mathbf{X}(\mathbf{bc}^T + \mathbf{cb}^T)$
- $\frac{\partial \mathbf{x}^T \mathbf{B} \mathbf{x}}{\partial \mathbf{x}} = (\mathbf{B} + \mathbf{B}^T) \mathbf{x}$
- $\frac{\partial}{\partial \mathbf{x}} (\mathbf{x} - \mathbf{As})^T \mathbf{W}(\mathbf{x} - \mathbf{As}) = 2\mathbf{W}(\mathbf{x} - \mathbf{As})$
- $\frac{\partial}{\partial \mathbf{X}} \|\mathbf{X}\|_F^2 = \frac{\partial}{\partial \mathbf{X}} \text{Tr}(\mathbf{XX}^H) = 2\mathbf{X}$

1.2 Linear Algebra

- **positive definite** (pd) if $\mathbf{a}^T \mathbf{V} \mathbf{a} > 0$
- $(\mathbf{x} - \mathbf{b})^T (\mathbf{x} - \mathbf{b}) = \|\mathbf{x} - \mathbf{b}\|_2^2$
- $\|\mathbf{X}\|_F = \|\mathbf{X}^T\|_F$

1.3 Distributions

Valid distribution $p(x) > 0, \forall x$ and $\sum p(x) = 1$
Model is identifiable iff $\theta_1 = \theta_2 \rightarrow P_{\theta_1} = P_{\theta_2}$

- **Gaussian** (Not convex):

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

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

- **Poisson**: $P(k \text{ events in interval}) = e^{-\lambda} \frac{\lambda^k}{k!}$
- **Bernoulli**: $p(y|\mu) = \mu^y (1-\mu)^{1-y}$

1.4 Convexity

A function $f(x)$ is convex if

- for any $\mathbf{x}_1, \mathbf{x}_2 \in \mathbf{X}$ and $0 \leq \lambda \leq 1$, we have :
 $f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda)f(\mathbf{x}_2)$
- it is a sum of convex functions
- composition of convex and linear functions
- $f(x) = g(h(x))$, g, h are convex, g increasing
- the Hessian \mathbf{H} is positive semi-definite

1.5 Others

- 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}

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

- Multi-class \mathbf{x}

$$p(\mathbf{y}|\mathbf{X}, \beta) = \prod p(\mathbf{y}_n|\mathbf{x}_n, \beta)$$

$$= \prod \prod [p(\mathbf{y}_n = k|\mathbf{x}_n, \beta)]^{y_{nk}}$$

2 Cost functions

Mean square error (MSE):

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

- MSE is **strictly convex** thus it has only one global minimum value.
- MSE is very prone to outliers.

Mean Absolute Error (MAE):

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

- MAE is more robust to outliers than MSE.

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

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

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

Hinge loss:

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

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

3 Linear Regression

- Model that assume linear relationship

$$\mathbf{y}_n \approx f(\mathbf{x}_n) := \mathbf{w}_0 + \mathbf{w}_1 \mathbf{x}_{n1} + \dots = \mathbf{w}_0 + \mathbf{x}_n^T \mathbf{w}$$

$$\approx \tilde{\mathbf{x}}_n^T \mathbf{w}, \text{ where } \tilde{\mathbf{x}} \text{ contains offset comp.}$$

- $D > N$ problem: task is underdetermined.

4 Optimization

- Local minimum:
 $L(w^*) \leq L(w) \forall w : \|w - w^*\| < \epsilon$
- Global minimum: $L(w^*) \leq L(w) \forall w$

4.1 Grid search

- Compute the cost over a grid of V points.
Exponential Complexity $\mathcal{O}(|V|^D)$, D is the dimension. Hard to find a good range of values. No guarantee to converge.

4.2 GD - Gradient Descent (Batch)

- 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}
 $\mathbf{w} = \arg \min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$
- 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 γ too big, method might diverge. With γ too small, convergence is slow.
- Very sensitive to ill-conditioning \Rightarrow always normalize features \Rightarrow allow different directions to converge at same speed.

4.3 SGD - Stochastic Gradient Descent

SGD update rule (only n-th training example):

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

Idea: Cheap but unbiased estimate of grad.

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

Robbins-Monroe condition:

- $\gamma^{(t)} : \sum_{t=1}^{\infty} \gamma^{(t)} = \infty; \sum_{t=1}^{\infty} (\gamma^{(t)})^2 < \infty$
- e.g. $\gamma^{(t)} = 1/(t+1)^r, r \in (0.5, 1)$

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

4.5 Gradients for MSE

- We define the error vector \mathbf{e} :
 $\mathbf{e} := \mathbf{y} - \mathbf{X} \mathbf{w}$
- and MSE as follows:
 $\mathcal{L}(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^N (\mathbf{y}_n - \tilde{\mathbf{x}}_n^T \mathbf{w})^2 = \frac{1}{2N} \mathbf{e}^T \mathbf{e}$
- then the gradient is given by

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

- Optimality conditions:

1. *necessary*: gradient equal zero:
 $\frac{d\mathcal{L}(\mathbf{w}^*)}{d\mathbf{w}} = 0$
2. *sufficient*: Hessian matrix is positive definite: $\mathbf{H}(\mathbf{w}^*) = \frac{d^2 \mathcal{L}(\mathbf{w}^*)}{d\mathbf{w} d\mathbf{w}^T} = \frac{1}{N} \mathbf{X}^T \mathbf{X}$

4.6 Subgradients (Non-Smooth OPT)

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

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

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

4.7 Constrained Optimization

Find solution $\min \mathcal{L}(\mathbf{w})$ s.t. $\mathbf{w} \in \mathcal{C}$

- Add proj. onto \mathcal{C} after each step:

$$P_{\mathcal{C}}(\mathbf{w}') = \arg \min_{\mathbf{v}} \|\mathbf{v} - \mathbf{w}'\|, \mathbf{v} \in \mathcal{C}$$

$$\mathbf{w}^{(t+1)} = P_{\mathcal{C}}[\mathbf{w}^{(t)} - \gamma \nabla \mathcal{L}(\mathbf{w}^{(t)})]$$

- Use penalty functions

- $\min \mathcal{L}(\mathbf{w}) + I_{\mathcal{C}}, I_{\mathcal{C}} = 0$ if $\mathbf{w} \in \mathcal{C}$, ∞ otherwise
- $\min \mathcal{L}(\mathbf{w}) + \lambda \|\mathbf{Aw} - \mathbf{b}\|$
- Stopping criteria when $\mathcal{L}(\mathbf{w})$ close to 0

4.8 Complexities for MSE/MAE per iteration

- GD= $\mathcal{O}(ND)$
- MB-GD= $\mathcal{O}(BD)$
- SGD= $\mathcal{O}(D)$

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

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

- The **Gram matrix** $\mathbf{X}^T \mathbf{X}$ is pd and is also invertible iff \mathbf{X} has full column rank.
- *Complexity*: $\mathcal{O}(ND^2 + D^3) \equiv \mathcal{O}(ND^2)$
- \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.
- Can still solve using a linear system solver using normal equations:

$$\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{y}$$

6 Maximum Likelihood (MLE)

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

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

- Another way of expressing this:

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

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

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

- Define cost with log-likelihood

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

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

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

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

- MLE can also be interpreted as finding the model under which the observed data is most likely to have been generated from.
- $\mathbf{w}_{MLE} \rightarrow \mathbf{w}_{true}$ for large amount of data

7 Ridge Regression and LASSO

- Add **regularization term**

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) + \mathcal{W}(\mathbf{w})$$

- This corresponds to MAP estimate with prior on weights.
- L_2 -Reg. (Ridge): $\Omega(\mathbf{w}) = \lambda \|\mathbf{w}\|_2^2$
- \rightarrow small values of \mathbf{w}_i , not sparse
- $\rightarrow \mathbf{w}^* = (\mathbf{X}^T \mathbf{X} + \lambda' \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$ with $\lambda' = 2N\lambda$

- $\rightarrow (\mathbf{X}^T \mathbf{X} + \lambda' \mathbf{I})^{-1}$ exists (lifted eigenvalues)
- L_1 -Reg. (Lasso): $\Omega(\mathbf{w}) = \lambda \|\mathbf{w}\|_1$
- \rightarrow sparsity of weight vector
- \rightarrow implicit model selection
- **Maximum-a-posteriori (MAP)**
- (i) Posterior prob. \propto Likelihood \times Prior prob

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

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|0, \sigma_0^2 \mathbf{I}_D)$$

$$\text{then } \rightarrow \mathbf{w}^* = \arg \max_{\mathbf{w}} p(\mathbf{y}|\mathbf{X} \mathbf{w}) \cdot p(\mathbf{w})$$

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \sum \frac{1}{2\sigma_n^2} (\mathbf{y}_n - \mathbf{x}_n^T \mathbf{w})^2 + \frac{1}{2\sigma_0^2} \|\mathbf{w}\|^2$$

8 Bias-Variance decomposition

- **Simple** (e.g. large λ) \rightarrow large bias but low variance
- **Complex** (e.g. small λ) \rightarrow low bias but large variance
- The expected squared loss between true model and learned model is a sum of three non-negative terms:

$$\mathbb{E}_S[(f(x) + \epsilon - f_S(x))^2] = \text{Var}[\epsilon] + \text{bias} + \text{variance:}$$

- **Bias** $= (f(x) - \mathbb{E}_{S'}[f_{S'}(x)])^2$: Difference between actual value and expected prediction.
- **Variance** $= \mathbb{E}_S[(\mathbb{E}_{S'}[f_{S'}(x)] - f_S(x))^2]$: variance of predictions between training sets.

- All terms are lower bounds for the error.
- Cannot do better than $\text{Var}[\epsilon]$.

9 Logistic Regression

- **Classification** relates input variables \mathbf{x} to discrete output variable \mathbf{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}_s

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

- **Logistic function**

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

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

- The probabilistic model:
 $p(\mathbf{y}_n = \mathbf{C}_2|\mathbf{x}_n) = 1 - \sigma(\mathbf{x}_n^T \mathbf{w})$

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

- The negative log-likelihood (w.r.t. MLE):

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

$$= - \sum_{n=1}^N \ln[1 + \exp(\mathbf{x}_n^T \mathbf{w})] - y_n \mathbf{x}_n^T \mathbf{w}$$

- We can use the fact that

$$\frac{d}{dz} \ln(1 + \exp(z)) = \sigma(z)$$

- Gradient of the log-likelihood

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

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

- The neg. log-likelihood $-\mathcal{L}_{MLE}(\mathbf{w})$ is convex
- **Hessian** of the neg. log-likelihood

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

- Hessian is the derivative of the gradient

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

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

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

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 neg. log-likelihood is not strictly convex. *????*

- Newton's Method

- Uses second-order information and takes steps in the direction that minimizes a quadratic approximation (Taylor)

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

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

11 k-Nearest Neighbor (k-NN)

- The k-NN prediction for \mathbf{x} is
$$f(\mathbf{x}) = \frac{1}{k} \sum_{\mathbf{x}_n \in nbh_k(\mathbf{x})} \mathbf{y}_n$$
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

$$\mathcal{L}(\mathbf{w}) = \min_{\mathbf{w}} \sum_{n=1}^N [1 - \mathbf{y}_n \tilde{\phi}_n^T \mathbf{w}]_+ + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

- 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
$$[\mathbf{v}_n]_+ = \max(0, \mathbf{v}_n) = \max_{\alpha_n \in [0, 1]} \alpha_n \mathbf{v}_n$$
- We can rewrite the problem as
$$\min_{\mathbf{w}} \max_{\alpha} \sum_{n=1}^N \alpha_n (1 - \mathbf{y}_n \phi_n^T \mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|^2$$
- 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} :
$$\nabla_{\mathbf{w}} G(\mathbf{w}, \alpha) = - \sum_{n=1}^N \alpha_n \mathbf{y}_n \mathbf{x}_n + \lambda \mathbf{w}$$

- Equating this to 0, we get:
$$\mathbf{w}(\alpha) = \frac{1}{\lambda} \sum_{n=1}^N \alpha_n \mathbf{y}_n \mathbf{x}_n = \frac{1}{\lambda} \mathbf{X}^T \mathbf{Y} \alpha$$
$$\mathbf{Y} := \text{diag}(\mathbf{y})$$
- Plugging \mathbf{w}^* back in the dual problem
$$\max_{\alpha \in [0, 1]^N} \alpha^T \mathbf{1} - \frac{1}{2\lambda} \alpha^T \mathbf{Y} \mathbf{X} \mathbf{X}^T \mathbf{Y} \alpha$$
- 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.
- α is the slope of lines that are lower bound to Hinge loss
- Non support vector:** Example that lies on the correct side, outside margin $\alpha_n = 0$
- Essen. support vector:** Example that lies on the margin $\alpha_n \in (0, 1)$
- Bound support vector:** Example that lies strictly inside the margin or wrong side

- $\alpha_n = 1$
- Use Coordinates Descent to find α . Update one coordinate (argmin) at the time and others constant.

13 Kernel Ridge Regression

- The following is true for ridge regression
$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_D)^{-1} \mathbf{X}^T \mathbf{y}, \quad (1)$$
$$= \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I}_N)^{-1} \mathbf{y} = \mathbf{X}^T \alpha^*, \quad (2)$$
- Complexity of computing \mathbf{w} : (1) $O(D^2 N + D^3)$, (2) $O(DN^2 + N^3)$
- Thus we have $\mathbf{w}^* = \mathbf{X}^T \alpha^*$, with $\mathbf{w}^* \in \mathbb{R}^D$ and $\alpha^* \in \mathbb{R}^N$
- The representer theorem allows us to write an equivalent optimization problem in terms of α .
$$\alpha = \arg\max_{\alpha} \left(-\frac{1}{2} \alpha^T (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_N) \alpha + \alpha^T \mathbf{y} \right)$$
- $\mathbf{K} = \mathbf{X} \mathbf{X}^T$ 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:**
 - 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)
$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2} (\mathbf{x} - \mathbf{x}')^T (\mathbf{x} - \mathbf{x}')\right)$$
 - Sigmoid Kernel: $\tanh(\langle \mathbf{x}_i, \mathbf{x}_j \rangle + r)$
- Properties of kernels** to ensure the existence of a corresponding ϕ :
 - symmetric: $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$
 - positive semi-definite.
- 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.
$$\min_{\mathbf{z}, \mu} \mathcal{L}(\mathbf{z}, \mu) = \sum_{k=1}^K \sum_{n=1}^N z_{nk} \|\mathbf{x}_n - \mu_k\|^2$$
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
 - For all n, compute \mathbf{z}_n given μ
$$z_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\mathbf{x}_n - \mu_j\|^2 \\ 0 & \text{otherwise} \end{cases}$$
 - 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
$$p(\mathbf{z}, \mu) = \prod_{n=1}^N \prod_{k=1}^K [\mathcal{N}(\mathbf{x}_n | \mu_k, \mathbf{I})]^{z_{nk}}$$
- $\log p(\mathbf{x}_n | \mu, \mathbf{z}) = \sum_{n=1}^N \sum_{k=1}^K \frac{1}{2} \|\mathbf{x}_n - \mu_k\|^2 z_{nk} + c'$

- 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
$$p(\mathbf{X}, \mathbf{z} | \mu, \Sigma, \pi) = \prod_{n=1}^N p(\mathbf{x}_n | \mathbf{r}_n, \mu, \Sigma) p(\mathbf{z}_n | \pi)$$
$$= \prod_{n=1}^N \prod_{k=1}^K [(\mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k))^{z_{nk}}] \prod_{k=1}^K [\pi_k]^{z_{nk}}$$
- 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
$$p(\mathbf{x}_n | \theta) = \sum_{k=1}^K p(\mathbf{x}_n, z_n = k | \theta)$$
$$= \sum_{k=1}^K p(z_n = k | \theta) p(\mathbf{x}_n | z_n = k, \theta)$$
$$= \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k)$$
- 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
$$\max_{\theta} \sum_{n=1}^N \log \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k)$$

- 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

16 Expectation Maximization Algorithm

- [ALGORITHM] Start with $\theta^{(1)}$ and iterate
 - Expectation step:* Compute a lower bound to the cost such that it is tight at the previous $\theta^{(t)}$ with equality when,
$$q_{kn} = \frac{\pi_k \mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k)}{\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k)}$$
 - Maximization step:* Update θ
$$\theta^{(t+1)} = \arg\max_{\theta} \mathcal{L}(\theta, \theta^{(t)})$$
$$\mu_k^{(t+1)} = \frac{\sum_{n=1}^N q_{kn}^{(t)} (\mathbf{x}_n) \mathbf{x}_n}{\sum_{n=1}^N q_{kn}^{(t)}}$$
$$\Sigma_k^{(t+1)} = \frac{\sum_{n=1}^N q_{kn}^{(t)} (\mathbf{x}_n - \mu_k^{(t+1)}) (\mathbf{x}_n - \mu_k^{(t+1)})^T}{\sum_{n=1}^N q_{kn}^{(t)}}$$
$$\pi_k^{(t+1)} = \frac{1}{N} \sum_{n=1}^N q_{kn}^{(t)}$$
- 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

- We can add a regularizer and minimize the following cost:

$$\mathcal{L}(\mathbf{W}, \mathbf{Z}) = \frac{1}{2} \sum_{(d,n) \in \Omega} [x_{dn} - (\mathbf{W} \mathbf{Z}^T)_{dn}]^2 + \frac{\lambda_w}{2} \|\mathbf{W}\|_{\text{Frob}}^2 + \frac{\lambda_z}{2} \|\mathbf{Z}\|_{\text{Frob}}^2$$

- SGD:** For one fixed element (d, n) we derive entry (d', k) of \mathbf{W} (if $d = d'$ oth. 0):
$$\frac{\partial}{\partial w_{d',k}} f_{d,n}(\mathbf{W}, \mathbf{Z}) = -[x_{dn} - (\mathbf{W} \mathbf{Z}^T)_{dn}] z_{nk}$$

- And of \mathbf{Z} (if $n = n'$ oth. 0):
$$\frac{\partial}{\partial z_{n',k}} f_{d,n}(\mathbf{W}, \mathbf{Z}) = -[x_{dn} - (\mathbf{W} \mathbf{Z}^T)_{dn}] w_{nk}$$

- $\mathbf{W}^{t+1} = \mathbf{W}^t - \gamma \nabla_w f_{d,n}(\mathbf{W}^t, \mathbf{Z}^t)$
$$\mathbf{Z}^{t+1} = \mathbf{W}^t - \gamma \nabla_z f_{d,n}(\mathbf{W}^t, \mathbf{Z}^t)$$
- 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:* $O(DNK^2 + NK^3) \rightarrow O(DNK^2)$

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*.
- Truncated SVD:** Take the matrix $\mathbf{S}^{(K)}$ with the K first diagonal elements non zero. Then, rank- K approx:
$$\mathbf{X} \approx \mathbf{X}_K = \mathbf{U} \mathbf{S}^{(K)} \mathbf{V}^T$$

19 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
$$\mathbf{W} = \mathbf{U} \mathbf{S}_D^{1/2}, \mathbf{Z}^T = \mathbf{S}^{1/2} \mathbf{V}^T$$

- Not invariant under scalings of the feature = arbitrariness, \rightarrow normalize \mathbf{X}

20 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:
$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^N \left(y_n - f^{(L+1)} \circ \dots \circ f^{(1)}(\mathbf{x}_n^{(0)}) \right)^2$$
We can use SGD to minimize the cost function.

20.1 Backpropagation Algorithm

- Forward pass:* Compute
$$\mathbf{z}^{(l)} = \left(\mathbf{W}^{(l)} \right)^T \mathbf{x}^{(l-1)} + \mathbf{b}^{(l)}$$
with $\mathbf{x}^{(0)} = \mathbf{x}_n$ and $\mathbf{x}^{(l)} = \phi(\mathbf{z}^{(l)})$.
- Backward pass:* Set $\delta^{(L+1)} = -2(y_n - \mathbf{x}^{(L+1)}) \phi'(\mathbf{z}^{(L+1)})$ (if squared loss). Then compute

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

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

20.2 Activation Functions

- Sigmoid** $\phi(x) = \frac{1}{1+e^{-x}}$ Positive, bounded.
$$\phi'(x) \simeq 0 \text{ for large } |x| \Rightarrow \text{Learning slow.}$$
- 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)

20.3 Convolutional NN

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

20.4 Reg. Data Augmentation and Dropout

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

21 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. (\rightarrow independent)
- 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 any of its descendants 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

