

PCML Cheat Sheet

Contents

1	Math Prerequisites	2
1.1	Convexity	2
1.2	Linear Algebra	2
2	Cost functions	2
3	Regression	2
3.1	Linear Regression	2
3.2	Gradient Descent	3
3.2.1	Grid search	3
3.3	Batch Gradient Descent	3
3.4	Gradients for MSE	3
3.5	Least Squares	3
3.6	Maximum Likelihood	3
3.7	Ridge Regression	3
3.8	Cross-Validation	4
3.9	Bias-Variance decomposition	4
3.10	Logistic Regression	4
4	Generalized Linear Model	5
5	k-Nearest Neighbor (k-NN)	5
6	Kernel Ridge Regression	5
7	Support Vector Machine	6
8	K-means	6
9	Gaussian Mixture Models	7
10	Expectation Maximization Algorithm	7
11	Matrix factorization	7
12	Singular Value Decomposition	8
13	Principal Component Analysis	8
14	Belief Propagation	8
15	Multi-Layer Perceptron (MLP)	8
16	Gaussian Process (GP)	8
17	Decision Trees (DT)	9
18	Random Forests (RF)	9

1 Math Prerequisites

- Bayes rule

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

- 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:

$$V(XY) = E(X^2)E(Y^2) - [E(X)]^2[E(Y)]^2$$

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

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

1.1 Convexity

- A function is convex when a line joining two points never intersects anywhere else.
- 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 :

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

- Strictly convex if the inequality is strict.
- A convex function has only one global minimum.
- Sums of convex functions are also convex.
- The Hessian is related to the convexity: a twice differentiable function is convex i-o-if the Hessian is positive definite.
- The Hessian of a convex function is positive semi-definite and for a strictly-convex function it is positive definite.

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

1.2 Linear Algebra

- Column $\mathbf{x} \in R^n$, rows \mathbf{x}^T , matrix $\mathbf{A} \in R^{m \times n}$
- $\mathbf{x}^T \mathbf{x}$ is a scalar, $\mathbf{x} \mathbf{x}^T$ is a matrix
- \mathbf{A}^{-1} exist if \mathbf{A} is full rank
- $(\mathbf{A}\mathbf{B})^T = \mathbf{B}^T \mathbf{A}^T$
- **Condition number** of a function measures how much the output value can change for a small change in the input. A matrix with a high condition number is said to be **ill-conditioned**. 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 $\mathbf{a}^T \mathbf{V} \mathbf{a} > 0$
- Cost of matrix inversion: $O(n^3) \rightarrow O(n^{2.372})$
- $\det(\mathbf{A})$ using LU decomposition: $O(n^3)$

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):

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

- 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

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

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

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

3 Regression

- **Data** consists of N pairs (y_n, \mathbf{x}_n)
 1. y_n the n'th output
 2. \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.

$$y_n \equiv f(\mathbf{x}_n) := \beta_0 + \beta_1 x_{n1} + \dots = \beta_0 + \mathbf{x}_n^T \boldsymbol{\beta}$$

with $\boldsymbol{\beta}$ the parameters of the model.

- Variance grows only linearly with dimensionality

3.2 Gradient Descent

- Gradient descent uses only first-order information and takes steps in the direction of the gradient
- Given a cost function $\mathcal{L}(\beta)$ we wish to find β that minimizes the cost:

$$\min_{\beta} \mathcal{L}(\beta)$$

3.2.1 Grid search

- Compute the cost over a grid of M points to find the minimum
- Exponential Complexity $O(NDM^D)$
- Hard to find a good range of values

3.3 Batch Gradient Descent

- Take steps in the opposite direction of the gradient

$$\beta^{(k+1)} \leftarrow \beta^{(k)} - \alpha \frac{d\mathcal{L}(\beta^{(k)})}{d\beta}$$

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

3.4 Gradients for MSE

$$\tilde{\mathbf{X}} = [1 \quad \mathbf{X}]$$

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

$$\mathbf{e} = \mathbf{y} - \tilde{\mathbf{X}}\beta$$

- and MSE as follows:

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

- then the gradient is given by

$$\frac{d\mathcal{L}}{d\beta} = -\frac{1}{N} \tilde{\mathbf{X}}^T \mathbf{e}$$

- Optimality conditions:

1. *necessary*: gradient equal zero: $\frac{d\mathcal{L}(\beta^*)}{d\beta} = 0$
 2. *sufficient*: Hessian matrix is positive definite: $\mathbf{H}(\beta^*) = \frac{d^2 \mathcal{L}(\beta^*)}{d\beta d\beta^T}$
- Very sensitive to illconditioning. Therefore, always normalize your feature otherwise step-size selection is difficult since different directions might move at different speed.
 - *Complexity*: $O(NDI)$ with I the number of iterations

3.5 Least Squares

- In some cases, we can compute the minimum of the cost function analytically.
- use the first optimality conditions:

$$\frac{d\mathcal{L}}{d\beta} = 0 \Rightarrow \tilde{\mathbf{X}}^T \mathbf{e} = \tilde{\mathbf{X}}^T (\mathbf{y} - \tilde{\mathbf{X}}\beta) = 0$$

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

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

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

$$y_* = \tilde{\mathbf{x}}_*^T \beta^* = \tilde{\mathbf{x}}_*^T (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^T \mathbf{y}$$

- The **Gram matrix** $\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}$ is positive definite and is also invertible iff $\tilde{\mathbf{X}}$ has full column rank.
- *Complexity*: $O(ND^2 + D^3) \equiv O(ND^2)$
- $\tilde{\mathbf{X}}$ can be rank deficient when $D > N$ or when the columns $\tilde{\mathbf{x}}_d$ are nearly collinear. In this case, the matrix is ill-conditioned, leading to numerical issues.

3.6 Maximum Likelihood

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

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

- Another way of expressing this:

$$\begin{aligned} p(\mathbf{y}|\tilde{\mathbf{X}}, \beta) &= \prod_{n=1}^N p(y_n|\tilde{\mathbf{x}}_n, \beta) \\ &= \prod_{n=1}^N \mathcal{N}(y_n|\tilde{\mathbf{x}}_n^T \beta, \sigma^2) \end{aligned}$$

- which defines the likelihood of observing \mathbf{y} given $\tilde{\mathbf{X}}$ and β
- Define cost with log-likelihood

$$\begin{aligned} \mathcal{L}_{lik}(\beta) &= \log p(\mathbf{y}|\tilde{\mathbf{X}}, \beta) \\ &= -\frac{1}{2\sigma^2} \sum_{n=1}^N (y_n - \tilde{\mathbf{x}}_n^T \beta)^2 + cns \end{aligned}$$

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

$$\underset{\beta}{\operatorname{argmin}} \mathcal{L}_{MSE}(\beta) = \underset{\beta}{\operatorname{argmax}} \mathcal{L}_{lik}(\beta)$$

- 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

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

$$\sum_n \log p(y_n|\tilde{\mathbf{x}}_n, \beta) = \sum_n |y_n - \tilde{\mathbf{x}}_n^T \beta| + cns$$

3.7 Ridge Regression

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

$$y_n = \beta_0 + \sum_{j=1}^M \beta_j \phi_j(\mathbf{x}_n) = \tilde{\phi}(\mathbf{x}_n)^T \beta$$

- This model is linear in β but nonlinear in \mathbf{x} . Note that the dimensionality is now M , not D .
- Polynomial basis

$$\phi(x_n) = [1, x_n, x_n^2, \dots, x_n^M]$$

- The least square solution becomes

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

- Complex models overfit easily. Thus we can choose simpler models by adding a **regularization term** which penalizes complex models

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

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

- Note that β_0 is not penalized.
- By differentiating and setting to zero we get

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

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

- Ridge regression improves the condition number of the Gram matrix since the eigenvalues of $(\tilde{\Phi}^T \tilde{\Phi} + \lambda I_m)$ are at least λ
- **Maximum-a-posteriori (MAP) estimator:**
 - Maximizes the product of the likelihood *and* the **prior**.

$$\beta_{MAP} = \underset{\beta}{\operatorname{argmax}} (p(\mathbf{y}|\mathbf{X}, \Lambda) p(\beta|\Sigma))$$

- Assume $\beta_0 = 0$

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

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

$$\min_{\beta} \frac{1}{2N} \sum_{n=1}^N (y_n - \tilde{\Phi}(\mathbf{x}_n)^T \beta)^2, \quad \text{such that } \sum_{i=1}^M |\beta_i| \leq \tau$$

3.8 Cross-Validation

- We should choose λ to minimize the mistakes that will be made in the future.
- We split the data into train and validation sets and we pretend that the validation set is the future data. We fit our model on the training set and compute a prediction-error on the validation set. This gives us an *estimate* of the *generalization error*.
- **K-fold cross validation** randomly partition the data into K groups. We train on $K - 1$ groups and test on the remaining group. We repeat this until we have tested on all K sets. We then average the results.
- Cross-validation returns an unbiased estimate of the generalization error and its variance.

3.9 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.

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

- Both model bias and estimation bias are important
- Ridge regression increases estimation bias while reducing variance
- Increasing model complexity increases test error
- Small $\lambda \rightarrow$ low bias but large variance
- Large $\lambda \rightarrow$ large bias but low variance

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

3.10 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**

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

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

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

- The probabilistic model:

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

- The log-likelihood:

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

- We can use the fact that

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

- Gradient of the log-likelihood

$$\begin{aligned} \mathbf{g} &= \frac{d\mathcal{L}}{d\beta} = \sum_{n=1}^N \left(\tilde{\mathbf{x}}_n y_n - \tilde{\mathbf{x}}_n \sigma(\tilde{\mathbf{x}}_n^T \beta) \right) \\ &= -\tilde{\mathbf{X}}^T [\sigma(\tilde{\mathbf{X}}\beta) - \mathbf{y}] \end{aligned}$$

- The negative of the log-likelihood $-\mathcal{L}_{mle}(\beta)$ 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

$$\begin{aligned}\mathbf{H}(\boldsymbol{\beta}) &= -\frac{d\mathbf{g}(\boldsymbol{\beta})}{d\boldsymbol{\beta}^T} = \sum_{n=1}^N \frac{d}{d\boldsymbol{\beta}^T} \sigma(\tilde{\mathbf{x}}_n^T \boldsymbol{\beta}) \tilde{\mathbf{x}}_n \\ &= \sum_{n=1}^N \tilde{\mathbf{x}}_n \sigma(\tilde{\mathbf{x}}_n^T \boldsymbol{\beta}) (1 - \sigma(\tilde{\mathbf{x}}_n^T \boldsymbol{\beta})) \tilde{\mathbf{x}}_n^T \\ &= \tilde{\mathbf{X}}^T \mathbf{S} \tilde{\mathbf{X}}\end{aligned}$$

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

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

- 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

$$\begin{aligned}\mathcal{L}(\boldsymbol{\beta}) &= \mathcal{L}(\boldsymbol{\beta}^{(k)}) + \mathbf{g}_k^T (\boldsymbol{\beta} - \boldsymbol{\beta}^{(k)}) \\ &\quad + (\boldsymbol{\beta} - \boldsymbol{\beta}^{(k)})^T \mathbf{H}_k (\boldsymbol{\beta} - \boldsymbol{\beta}^{(k)})\end{aligned}$$

and it's minimum is at

$$\boldsymbol{\beta}^{k+1} = \boldsymbol{\beta}^{(k)} - \alpha_k \mathbf{H}_k^{-1} \mathbf{g}_k$$

where \mathbf{g}_k is the gradient and α_k the learning rate.

- Complexity: $O((ND^2 + D^3)I)$
- **Penalized Logistic Regression**

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

4 Generalized Linear Model

- **Exponential family distribution**

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

- Bernoulli distribution

$$\begin{aligned}p(y | \mu) &= \mu^y (1 - \mu)^{1-y} \\ &= \exp(y \log(\frac{\mu}{1-\mu}) + \log(1 - \mu))\end{aligned}$$

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

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

- Note that $\boldsymbol{\mu}$ is the mean parameter of \mathbf{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

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

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

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

where $p(y_n | \tilde{\mathbf{x}}_n^T \boldsymbol{\beta})$ is an exponential family distribution

- We obtain the solution

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

5 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 in the training data

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

6 Kernel Ridge Regression

- The following is true for ridge regression

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

- Complexity of computing $\boldsymbol{\beta}$: (1) $O(D^2 N + D^3)$, (2) $O(DN^2 + N^3)$
- Thus we have

$$\boldsymbol{\beta} = \sum_{n=1}^N \alpha_n \mathbf{x}_n, \quad \mathbf{y} = \sum_{d=1}^D \beta_d \tilde{\mathbf{x}}_d$$

with \mathbf{x}_n the rows of \mathbf{X} and $\tilde{\mathbf{x}}_d$ the columns of \mathbf{X}

- The representer theorem allows us to write an equivalent optimization problem in terms of $\boldsymbol{\alpha}$.

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

- $\mathbf{K} = \mathbf{X} \mathbf{X}^T$ 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 $\phi(\mathbf{x})$ 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 $\langle \mathbf{x}, \mathbf{x}' \rangle$ 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)**

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

- 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

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

7 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

$$g(\beta) = \min_{\beta} \sum_{n=1}^N [1 - y_n \tilde{\phi}_n^T \beta]_+ + \frac{\lambda}{2} \sum_{j=1}^M \beta_j^2$$

- Minimum doesn't change with a rescaling of β
- choose the hyperplane so that the distance from it to the nearest data point on each side is maximized
- **Duality**:
 - Hard to minimize $g(\beta)$ so we define

$$g(\beta) = \max_{\alpha} G(\beta, \alpha)$$

- 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

$$\min_{\beta} \max_{\alpha \in [0, C]^N} \sum_{n=1}^N \alpha_n (1 - y_n \tilde{\phi}_n^T \beta) + \frac{1}{2} \sum_{j=1}^M \beta_j^2$$

- This is differentiable, convex in β and concave in α
- **Minimax theorem**:

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

because G is convex in β and concave in α .

- Derivative w.r.t. β :

$$\frac{dG}{d\beta} = - \left(\sum_{n=1}^N \alpha y_n \tilde{\phi}_n \right) + \begin{bmatrix} 0 \\ \beta_{1:M} \end{bmatrix}$$

- Equating this to 0, we get:

$$\beta_{1:M}^* = \sum_{n=1}^N \alpha_n y_n \phi_n = \Phi^T \text{diag}(\mathbf{y}) \alpha$$

$$\alpha^T \mathbf{y} = 0$$

- Plugging β^* back in the dual problem

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

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

8 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.

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

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

1. For all n , compute \mathbf{r}_n given μ

$$r_{nk} = \begin{cases} 1 & \text{if } k = \text{argmin}_j \|\mathbf{x}_n - \mu_j\|_2^2 \\ 0 & \text{otherwise} \end{cases}$$

2. For all k , compute μ_k given \mathbf{r}

$$\mu_k = \frac{\sum_{n=1}^N r_{nk} \mathbf{x}_n}{\sum_{n=1}^N r_{nk}}$$

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

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

- Computation can be heavy, each example can belong to only on cluster and clusters have to be spherical.

9 Gaussian Mixture Models

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

$$p(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \mathbf{r}) = \prod_{n=1}^N \prod_{k=1}^K [\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)]^{r_{nk}}$$

- **Soft-clustering**: Points can belong to several cluster by defining r_n to be a random variable.

$$p(r_{nk} = 1) = \pi_k \text{ where } \pi_k > 0, \forall k, \sum_{k=1}^K \pi_k = 1$$

- Joint distribution of Gaussian mixture model

$$\begin{aligned} p(\mathbf{X}, \mathbf{r}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) &= \prod_{n=1}^N [p(\mathbf{x}_n|\mathbf{r}_n, \boldsymbol{\mu}, \boldsymbol{\Sigma})p(\mathbf{r}_n|\boldsymbol{\pi})] \\ &= \left[\prod_{k=1}^K [(\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k))^{r_{nk}}] \prod_{k=1}^K [\pi]^{r_{nk}} \right] \end{aligned}$$

- r_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 r_n out from the likelihood

$$\begin{aligned} p(\mathbf{x}_n|\boldsymbol{\theta}) &= \sum_{k=1}^K p(\mathbf{x}_n, r_{nk} = 1|\boldsymbol{\theta}) \\ &= \sum_{k=1}^K p(r_{nk} = 1|\boldsymbol{\theta})p(\mathbf{x}_n|r_{nk} = 1, \boldsymbol{\theta}) \\ &= \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \end{aligned}$$

- Without a latent variable model, number of parameters grow at rate $O(N)$
- After marginalization, the growth is reduced to $O(D^2K)$
- To get maximum likelihood estimate of $\boldsymbol{\theta}$, we maximize

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

10 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 $\boldsymbol{\theta}^{(i)}$ with equality when,

$$\gamma(r_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}$$

2. *Maximization step*: Update $\boldsymbol{\theta}$

$$\boldsymbol{\theta}^{(i+1)} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^{(i)})$$

$$\boldsymbol{\mu}_k^{(i+1)} = \frac{\sum_{n=1}^N \gamma^{(i)}(r_{nk}) \mathbf{x}_n}{\sum_{n=1}^N \gamma^{(i)}(r_{nk})}$$

$$\boldsymbol{\Sigma}_k^{(i+1)} = \frac{\sum_{n=1}^N \gamma^{(i)}(r_{nk}) (\mathbf{x}_n - \boldsymbol{\mu}_k^{(i+1)}) (\mathbf{x}_n - \boldsymbol{\mu}_k^{(i+1)})^T}{\sum_{n=1}^N \gamma^{(i)}(r_{nk})}$$

$$\pi_k^{(i+1)} = \frac{1}{N} \sum_{n=1}^N \gamma^{(i)}(r_{nk})$$

- If covariance is diagonal \rightarrow K-means.

11 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 $N \times M$ matrix that gives features for the users
 - \mathbf{W} a $D \times M$ 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:

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

- 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)**:

$$\begin{aligned} \mathbf{Z}^T &\leftarrow (\mathbf{W}^T \mathbf{W} + \lambda_z \mathbf{I}_M)^{-1} \mathbf{W}^T \mathbf{X} \\ \mathbf{W}^T &\leftarrow (\mathbf{Z}^T \mathbf{Z} + \lambda_w \mathbf{I}_D)^{-1} \mathbf{Z}^T \mathbf{X}^T \end{aligned}$$

- *Complexity*: $O(DNM^2 + NM^3) \rightarrow O(DNM^2)$
- Probabilistic model

$$\begin{aligned} \prod_{n=1}^N \prod_{d \in O_n} \mathcal{N}(x_{dn}|\mathbf{w}_d^T \mathbf{z}_n, I) &\times \prod_{n=1}^N \mathcal{N}(\mathbf{z}_n|0, \frac{1}{\lambda_z} I) \\ &\times \prod_{d=1}^D \mathcal{N}(\mathbf{w}_d|0, \frac{1}{\lambda_w} I) \end{aligned}$$

- Since many ratings are missing we cannot normalize the data. A solution is to add offset terms:

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

12 Singular Value Decomposition

- Matrix factorization method

$$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

- \mathbf{U} is an $D \times D$ matrix
- \mathbf{V} is an $N \times N$ 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*.

- Let's now truncate these matrices

$$\mathbf{X} \approx \mathbf{U}_{tr} \mathbf{S}_{tr} \mathbf{V}_{tr}^*, \mathbf{T}_{tr} \approx \mathbf{U}_{tr} \mathbf{S}_{tr}, \mathbf{T}_{te} \approx \mathbf{X}_{te} \mathbf{V}_{tr}$$

with \mathbf{T}_{tr} the reduced feature set of \mathbf{X} and \mathbf{T}_{te} the reduced feature set of \mathbf{X}_{te}

13 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

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

- 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

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

14 Belief Propagation

- the goal is to learn *inference of the latent variables using belief propagation*
- Given a directed acyclic graph G and parameters θ , a **Bayesian network** defines the joint distribution as follows

$$p_\theta(\mathbf{x}) = \prod_{k=1}^K p_\theta(x_k | \text{parents}_k)$$

- Reduce complexity using bayesian rule.
- Message Passing:

1. Assign each ϕ_k to a cluster \mathbf{C}_i
2. Initial potential $\psi_i(\mathbf{C}_i) = \prod_{\phi_k \in \mathbf{C}_i} \phi_k$
3. Initial all messages to 1
4. Repeat: select and edge (i, j) and:

$$m_{i \rightarrow j}(\mathbf{S}_{i,j}) = \sum_{\mathbf{C}_i - \mathbf{S}_{i,j}} \psi_i \times \prod_{k \in (\mathcal{N}_i - j)} m_{k \rightarrow i}$$

- Belief: $\beta_i(\mathbf{C}_i) = \psi_i \times \prod_{k \in \mathcal{N}_i} \delta_{k \rightarrow i}$

15 Multi-Layer Perceptron (MLP)

- There are nonlinear function classes, whose convenient layered structure leads to efficient computation of gradients.
- Known as **feed-forward neural network**
- $\mathbf{z}_n^{(k)}$ is the k'th hidden vector
- $\mathbf{a}_n^{(k)}$ is the corresponding activation
- There are a total of K layers

$$a_{mn}^{(k)} = (\beta_m^{(k)})^T \mathbf{z}_n^{(k-1)}, \quad z_{mn}^{(k)} = h(a_{mn}^{(k)})$$

- For the first layer we have $\mathbf{z}_n^{(0)} = \mathbf{x}_n$
- For the last layer, we use a link function to map $\mathbf{z}_n^{(K-1)}$ to the output \mathbf{y}_n
- A 1-layer MLP is simply a generalization of linear/logistic regression
- $\mathbf{B}^{(k)}$ a matrix with rows $(\beta_m^{(k)})^T$

$$\mathbf{a}_n^{(k)} = \mathbf{B}^{(k)} \mathbf{z}_n^{(k-1)}, \quad \mathbf{z}_n^{(k)} = h(\mathbf{a}_n^{(k)})$$

- thus we have the input-output relationship

$$\hat{\mathbf{y}}_n = g((\beta^{(K-1)})^T * h(\mathbf{B}^{(K-2)} * h(* \dots * h(\mathbf{B}^{(1)} * \mathbf{x}_n)))$$

- with $g()$ the link function
- We learn parameters \mathbf{B} using stochastic gradient-descent
- Frequently used transfer function

$$g(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

- **Backpropagation**: technic to compute the gradient in time linear in the number of training points and the number of weights and is nothing else than a chain rule of differential calculus.
- The key-idea is to express the derivatives in terms of activations and hidden variables.
 1. **Forward**: compute $\mathbf{a}_n^{(k)}$ and $\mathbf{z}_n^{(k)}$
 2. **Backward**: compute $\delta_n^{(k)} = d\mathcal{L}/d\mathbf{a}_n^{(k)}$ using

$$\delta_n^{(k-1)} = \text{diag}[h'(\mathbf{a}_n^{(k-1)})](\mathbf{B}^{(k)})^T \delta_n^{(k)}$$

$$\delta_n^K = \mathbf{a}_n^K - t_i$$

3. Compute $d\mathcal{L}/d\mathbf{B}^{(k)}$ using

$$\frac{d\mathcal{L}}{d\mathbf{B}^{(k)}} = \sum_n \delta_n^{(k)} (\mathbf{z}_n^{(k)})^T$$

16 Gaussian Process (GP)

- GP are a family of statistical distributions in which time plays a role and for which any finite linear combination of samples has a joint Gaussian distribution.
- **non-parametric** method (rather uses latent variables) that compute a probability dist. over predictions
- They can be completely defined by their second-order statistics which means that if they have mean zero, then defining the covariance function completely defines the process behaviour.
- Let us place a probabilistic prior shape on the approximation of a function.

- A GP process defines a prior over function f

$$p(\mathbf{f}|X) = \mathcal{N}(\mathbf{f}|\mathbf{0}, K(X))$$

- $K(X)$ defines shape and prior knowledge about our problem

$$p(\mathbf{f}|\mathbf{y}, \mathbf{X}_*, \mathbf{X}) \sim \mathcal{N}(\mu', \sigma')$$

$$\mu' = K(\mathbf{X}_*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 I]^{-1} \mathbf{y}$$

$$\sigma' = K(\mathbf{X}_*, \mathbf{X}_*) - K(\mathbf{X}_*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 I]^{-1} K(\mathbf{X}, \mathbf{X}_*)$$

with σ_n the variance of the noise

- Quadratic kernel

$$k(\mathbf{x}_n, \mathbf{x}_m) = (1 + \mathbf{x}_n^T \mathbf{x}_m)^2$$

17 Decision Trees (DT)

- A decision tree is a structure in which each internal node represents a test on an attribute and each branch represents the outcome of the test and each leaf represents a class label.
- Fast to train and fast to make predictions
- Efficient for very high dimensional feature spaces and very large amounts of training data
- Lack of smoothness and high variance (overfitting)
- Goal: find a split (k, τ) that minimizes an impurity measure at the leaves
 - Find best feature to split on
 - Find best threshold

18 Random Forests (RF)

- RF correct the overfitting bad "habit" of DTs.
- Training: Learn M trees on different subsets (random) of training data
- Prediction: Average of prediction of each tree
- Variance of the model averaging:

$$z_1 = f_1 \Rightarrow z_M = \frac{1}{M} \sum_{i=1}^M f_i$$

$$V(z_1) = \sigma^2 \Rightarrow V(z_M) = \frac{1}{M} \sigma^2 + \rho \frac{M-1}{M} \sigma^2$$

- Variance reduction ratio:

$$\frac{V(z_1)}{V(z_M)} = \frac{M}{1 + \rho(M-1)}$$

- 2 techniques for decorrelating trees:
 - **Bagging**: Randomize training data
 - Randomized feature selection