Machine Learning Cheat Sheet

General

- Multivariate normal: $\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) \coloneqq \frac{1}{\sqrt{(2\pi)^D \det(\boldsymbol{\Sigma})}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right]$
- Bayes' Rule: $\text{pst} = \frac{\text{lkh} \cdot \text{pri}}{\text{evi}} \iff P(w \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid w)P(w)}{P(\mathcal{D})}$
  - $P(\mathcal{D}) = \int P(\mathcal{D} \mid w)P(w)dw$
- Logistic sigmoid: $\sigma(a) \coloneqq \frac{1}{1+\exp(-a)}$
- Softmax: $f_{k(\boldsymbol{a})} \coloneqq \frac{\exp(a_k)}{\sum_j \exp(a_j)}$ with $a_k = \ln p(\boldsymbol{x} \mid C_k)p(C_k)$
- One-hot coding: `C_k = ((k-1) * [0] + [1] + (n-k) * [0]).T`

Linear basis function regression

- Model $y(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{\varphi}(\boldsymbol{x})$
- Target $t = y(\boldsymbol{x}, \boldsymbol{w}) + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, \beta^{-1})$
  - $p(t \mid \boldsymbol{x}, \boldsymbol{w}, \beta) = \mathcal{N}(t \mid y(\boldsymbol{x}, \boldsymbol{w}), \beta^{-1})$
- Maximum likelihood estimation (constant prior)
  - optimize likelihood $p(\boldsymbol{t} \mid \boldsymbol{X}, \boldsymbol{w}, \beta) = \prod_n \mathcal{N}(t_n \mid \boldsymbol{w}^T \boldsymbol{\varphi}(\boldsymbol{x}_n), \beta^{-1})$
  - $\ln p(\boldsymbol{t} \mid \boldsymbol{X}, \boldsymbol{w}, \beta) = \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \frac{\beta}{2}\sum_n \left[t_n - \boldsymbol{w}^T \boldsymbol{\varphi}(\boldsymbol{x}_n)\right]^2$
  - $\hat{\boldsymbol{w}} = \arg\max_{\boldsymbol{w}} \ln p(\boldsymbol{t} \mid \boldsymbol{X}, \boldsymbol{w}, \beta) = \boldsymbol{\Phi}^\dagger \boldsymbol{t} = \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^T \boldsymbol{t}$ with $\boldsymbol{\Phi}_{nj} = \boldsymbol{\varphi}_{j(\boldsymbol{x}_n)}$
  - $\hat{\beta}^{-1} = \arg\max_{\beta^{-1}} \ln p(\boldsymbol{t} \mid \boldsymbol{X}, \boldsymbol{w}, \beta) = \frac{1}{N}\sum_n \left[t_n - \boldsymbol{w}^T \boldsymbol{\varphi}(\boldsymbol{x}_n)\right]^2$ (residual variance)
- Maximum A-Posteriori estimation (Gaussian prior)
  - optimize posterior $p(\boldsymbol{w} \mid D) \propto p(D \mid \boldsymbol{w})p(\boldsymbol{w})$
  - $\hat{\boldsymbol{w}} = \arg\max_{\boldsymbol{w}}(\ln p(D \mid \boldsymbol{w}) + \ln p(\boldsymbol{w})) = \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \alpha \boldsymbol{I}\right)^{-1} \boldsymbol{\Phi}^T \boldsymbol{t}$
- Expected loss: $E_{\mathcal{D}}[L] = \text{bias}^2 + \text{variance}$
  - $E_{\mathcal{D}}\left[(y(x, \mathcal{D}) - h(x))^2\right] = \left(E_{\mathcal{D}}[y(x, \mathcal{D})] - h(x)\right)^2 + E_{\mathcal{D}}\left[\left(y(x, \mathcal{D}) - E_{\mathcal{D}}[y(x, \mathcal{D})]\right)^2\right]$ with $\mathcal{D}$ = dataset, $h(x)$ = true data-generating function, and $y(x, \mathcal{D})$ = linear function fitted using $\mathcal{D}$
- k-fold cross-validation: split data into k partitions, use $k-1$ for training and 1 for testing and repeat k times, always using a different partition for testing

Fisher's linear discriminant

- project $\boldsymbol{x}$ to 1D: $y = \boldsymbol{w}^T \boldsymbol{x}$
- class mean: $\boldsymbol{m}_k \coloneqq \frac{1}{N_k}\sum_{n \in C_k} \boldsymbol{x}_n$
- projected class mean: $m_k \coloneqq \boldsymbol{w}^T \boldsymbol{m}_k$
- transformed within-class variance: $s_k^2 \coloneqq \sum_{n \in C_k} (y_n - m_k)^2$
- Fisher criterion: $J(\boldsymbol{w}) \coloneqq \frac{\text{between-class variance}}{\text{within-class variance}} = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} = \frac{\boldsymbol{w}^T \boldsymbol{S}_B \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{S}_W \boldsymbol{w}}$
  - $\boldsymbol{S}_B \coloneqq (\boldsymbol{m}_2 - \boldsymbol{m}_1)(\boldsymbol{m}_2 - \boldsymbol{m}_1)^T$
  - $\boldsymbol{S}_W \coloneqq \sum_{n \in C_1}(\boldsymbol{x}_n - \boldsymbol{m}_1)(\boldsymbol{x}_n - \boldsymbol{m}_1)^T + \sum_{n \in C_2}(\boldsymbol{x}_n - \boldsymbol{m}_2)(\boldsymbol{x}_n - \boldsymbol{m}_2)^T$
- Fisher's linear discriminant: $\hat{\boldsymbol{w}} = \arg\max_{\boldsymbol{w}} J(\boldsymbol{w}) \propto \boldsymbol{S}_W^{-1}(\boldsymbol{m}_2 - \boldsymbol{m}_1)$

Probabilistic generative models

- Two classes
  - $p(C_1 \mid \boldsymbol{x}) = \sigma(a) = \sigma(\boldsymbol{w}^T \boldsymbol{x} + w_0)$ with $a = \ln \frac{p(\boldsymbol{x} \mid C_1)p(C_1)}{p(\boldsymbol{x} \mid C_2)p(C_2)}$, $\boldsymbol{w} = \Sigma^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$, and $w_0 = \frac{1}{2}(\boldsymbol{\mu}_1^T \Sigma^{-1} \boldsymbol{\mu}_1 + \boldsymbol{\mu}_2^T \Sigma^{-1} \boldsymbol{\mu}_2) + \ln \frac{p(C_1)}{p(C_2)}$
  - Maximum likelihood estimation
    - $\hat{\pi} = \frac{N_1}{N}$ with $\pi \coloneqq p(C_1), p(C_2) = 1 - \pi$
    - $\hat{\boldsymbol{\mu}}_1 = \frac{1}{N_1}\sum_n t_n \boldsymbol{x}_n$, $\hat{\boldsymbol{\mu}}_2 = \frac{1}{N_2}\sum_n (1 - t_n)\boldsymbol{x}_n$
    - $\hat{\boldsymbol{\Sigma}} = \frac{N_1}{N}\boldsymbol{S}_1 + \frac{N_2}{N}\boldsymbol{S}_2$ with $\boldsymbol{S}_k = \frac{1}{N_k}\sum_{n \in C_k}(\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T$
- k classes
  - $p(C_k \mid \boldsymbol{x}) = \text{Softmax}(\boldsymbol{a}) = \text{Softmax}(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{w}_0)$ (assuming shared $\boldsymbol{\Sigma}$)
    - $a_k = \ln p(\boldsymbol{x} \mid C_k)p(C_k)$

  - $\boldsymbol{w}_k = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k$
  - $w_{k0} = -\frac{1}{2}\boldsymbol{\mu}_k^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \ln p(C_k)$
- Decision boundary between class i and j: $0 = \left(\boldsymbol{w}_j - \boldsymbol{w}_1\right)^T\boldsymbol{x} + \left(w_{j0} - w_{i0}\right)$
  - if $\boldsymbol{\Sigma}$ is not shared, discriminant is quadratic, otherwise linear

Probabilistic discriminative models

- $y(x) = \sigma(\boldsymbol{w}^T\boldsymbol{x}), p(C_1 \mid \boldsymbol{x}) = y, p(C_2 \mid \boldsymbol{x}) = 1 - y$
- Maximum likelihood estimate (generalizes to multi-class case)
  - $p(\boldsymbol{t} \mid \boldsymbol{w}) = \prod_n y_n^{t_n}(1-y_n)^{1-t_n}$ with $\boldsymbol{t} = \{0, 1\}$
  - $E(\boldsymbol{w}) = -\ln p(\boldsymbol{t} \mid \boldsymbol{w}) = -\sum_n [t_n \ln y_n + (1-t_n)\ln(1-y_n)]$
  - $\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = \sum_n \left[\sigma(\boldsymbol{w}^T\boldsymbol{x}_n)\boldsymbol{x}_n^T - t_n\boldsymbol{x}_n^T\right] = 0$ (no direct solution)
  - Newton's method (IRLS): $\boldsymbol{w}_{k+1} = \left(\boldsymbol{X}^T\boldsymbol{R}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{R}\boldsymbol{z}$ with $R_{nn} = y_n(1-y_n)$ and $\boldsymbol{z} = \boldsymbol{X}\boldsymbol{w}_k - \boldsymbol{R}^{-1}(\boldsymbol{y} - \boldsymbol{t})$

Neural networks

- Forward pass (apply input x_n): $z_j = h\left(a_j\right)$ with $h =$ activation function, $a_j = \sum_i w_{ji}z_i$
- Evaluate $\delta_k = y_k - t_k$ at the output neurons, backpropagate $\delta_j = h'\left(a_j\right)\sum_k w_{kj}\delta_k$
- Gradient $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$

Optimization algorithms

- Newton's method: $x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$, multidimensional: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{H}^{-1}\nabla f(x_k)$
- Stochastic gradient descent: $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \lambda\nabla_{\boldsymbol{w}} E(\boldsymbol{x}, \boldsymbol{w}_k)$
- SGD with momentum: velocity $v_{k+1} = \beta v_k + \lambda\nabla_{\boldsymbol{w}} E(\boldsymbol{x}, \boldsymbol{w}_k)$ with $\beta \in [0, 1)$
  - $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - v_{k+1}$
- AdaGrad (adaptive gradient per parameter) $w_i - \frac{\eta}{\sqrt{G_i}}g_i$; $G_i = \sum_{\tau=1}^{t}\left(g_i^{(\tau)}\right)^2$
- RMSProp (root mean square propagation) $\nu^{(t)} = \gamma\nu^{t-1} + (1-\gamma)\left(g_i^{(t)}\right)^2$; $w_i - \frac{\eta}{\sqrt{\nu^{(t)}}}g_i^{(t)}$ forgets earlier gradient magnitudes over time
- Adam (adaptive moment estimation, Momentum + RMSProp) $w^{(t)} \leftarrow w^{(t-1)} - \eta\frac{\widetilde{m}_w}{\sqrt{\widetilde{v}_w} + \varepsilon}$; $m_w^{(t)} \leftarrow \beta_1 m^{(t-1)} + (1-\beta_1)\left(g^{(t)}\right)^2$; $v_w^{(t)} \leftarrow \beta_2 m^{(t-1)} + (1-\beta_2)\left(g^{(t)}\right)^2$

Convnets

- Slide a filter $\boldsymbol{w}$ over the image, computing $\boldsymbol{w}^T\boldsymbol{x} + b$ for a local chunk $\boldsymbol{x}$ of the image
- valid padding = no padding: $n \times n \Longrightarrow (n-k+1) \times (n-k+1)$ with $k \times k$ kernel
- same padding = padding with $\frac{k}{2} - 1$ zero-pixels: $n \times n \Longrightarrow n \times n$
- Receptive field size $R = 1 + \sum_l (k_l - 1)$
- s-strided convolution: output size $M = \frac{N-k}{s} + 1$
- last layer: Flatten last convolutional layer and fully-connect to output

Decision trees

- for every observation $\boldsymbol{x} \in R_i$ predict the mean of all training observations in region R_m
- Construction via greedy recursive binary splitting
  - regression trees: use exhaustive search to find predictor x_j and cut point θ that minimize MSE $\min_{j,\theta}\left[\min_{c_-}\sum_{n\in R_-}(t_n - c_-)^2 + \min_{c_+}\sum_{n\in R_+}(t_n - c_+)^2\right]$ with optimal prediction $c_\pm := \frac{1}{N_\pm}\sum_{n\in R_\pm} t_n$ and halfplanes $R_{\pm(j,\theta)} := \left\{\boldsymbol{x} \mid x_j \gtreqless \theta\right\}$
  - classification trees: use cross-entropy $H := -\sum_k p_{mk}\ln p_{mk}$ or Gini index $G := \sum_k p_{mk}(1 - p_{mk})$ as loss function instead, $p_{mk} := \frac{1}{N_m}\sum_{n\in R_m} I(t_n = k)$
  - repeat for each half plane until stopping criterion is reached
- Bagging to reduce variance (samples are correlated, variance barely decreases)

  - draw bootstrap samples $\left\{\left(x_1^{(b)}, t_1^{(b)}\right), ..., \left(x_N^{(b)}, t_N^{(b)}\right)\right\}_{b=1}^{B}$ from the training set (with replacement) and train y_b^* on bth sample, average models: $y_{\text{bag}(\boldsymbol{x})} = \frac{1}{B}\sum_b y_b^*(\boldsymbol{x})$
- Random forests to reduce variance: construct B trees on bootstrap samples using a random subset of $J \leq P$ (ex. $J = \texttt{sqrt}\ P$) features for each split, then average the trees
- Gradient boosted regression trees: initialize $Y_0(x) = 0$, for all k fit regression tree y_k to residuals $r_{kn} = t_n - Y_{k-1}(x_n)$ and update $Y_{k(x)} = Y_{k-1}(x) + y_{k(x)}$

Principal Component Analysis
- minimize squared reconstruction error, equivalent to maximizing variance of projection
- principal components are main axes of covariance ellipse
- data covariance matrix $\boldsymbol{S} = \frac{1}{N}\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$ (eigenvalue decomposition)
- Principal components = eigenvectors $\boldsymbol{U}$

Probabilistic PCA
- define multivariate Gaussian density model with reduced number of parameters: $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \mathbb{W}^T + \sigma^2\boldsymbol{I})$
- maximum likelihood: $\widehat{\boldsymbol{W}} = \boldsymbol{U}_M\frac{(\boldsymbol{\Lambda}_M - \sigma^2\boldsymbol{I})^1}{2}\boldsymbol{R}$ with $\boldsymbol{U}_M, \boldsymbol{\Lambda}_M =$ first M eigenvectors/eigenvalues and $\boldsymbol{R} =$ arbitrary orthogonal matrix

Stochastic neighbor embedding
- preserve local neighborhood relationships
- minimize $L := \sum_{i\neq j} p_{ij}\ln\frac{p_{ij}}{q_{ij}} \propto -\sum_{i\neq j} p_{ij}\ln a_{ij} + \ln\sum_{k\neq l} a_{kl}$ via gradient descent
  - $q_{ij} = \frac{a_{ij}}{\sum_{k\neq l} a_{kl}}$ and $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$ with $p_{j|i} = \frac{\exp(-d_{ij}^2)}{\sum_{k\neq i}\exp(-d_{ik}^2)}$ with $d_{ij}^2 = \frac{|\ \boldsymbol{x}_i - \boldsymbol{x}_j\ |^2}{2\sigma_i^2}$
  - SNE: $a_{ij} = \exp\left(-\ |\ \boldsymbol{y}_i - \boldsymbol{y}_j\ |^2\right)$, t-SNE: $a_{ij} = \frac{1}{1 + |\ \boldsymbol{y}_i - \boldsymbol{y}_j\ |^2}$

K-means
- minimize $J := \sum_n\sum_k r_{nk}\ |\ \boldsymbol{x}_n - \boldsymbol{\mu}_k\ |^2$ with r_n one-hot encoded
- find optimal r_{nk} given current prototypes $\boldsymbol{\mu}_k$: $r_{nk} = 1$ if $k = \arg\min_j |\ \boldsymbol{x}_n - \boldsymbol{\mu}_j\ |^2$
- find optimal $\boldsymbol{\mu}_k$ given current cluster assignments r_{nk}: $\boldsymbol{\mu}_k = \frac{\sum_n r_{nk}\boldsymbol{x}_n}{\sum_n r_{nk}}$, iterate

Gaussian mixture models
- $p(\boldsymbol{x}) = \sum_k \pi_k\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ with priors $\pi_k := p(z_k = 1)$ with $\boldsymbol{z}$ one-hot encoded
- responsibility $\gamma(z_k) := p(z_k = 1 \mid \boldsymbol{x}) = \frac{\pi_k\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$
- Maximum likelihood: $\ln p(\boldsymbol{x} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_n \ln\left[\sum_k \pi_k\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\right]$ (no direct solution)
- compute γ_{nk} given current parameters $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$
- find optimal parameters using current γ_{nk}: $\boldsymbol{\mu}_k = \frac{1}{N_k}\sum_n \gamma_{nk}\boldsymbol{x}_n$, $\boldsymbol{\Sigma}_k = \frac{1}{N_k}\sum_n \gamma_{nk}(\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T + \lambda\boldsymbol{I}_D$, $\pi_k = \frac{N_k}{N}$ with $N_k := \sum_n \gamma_{nk}$

Transformers
- self Attention $y_i = \sum_{j=1}^{N} w_{i,j}x_j; \tilde{w}_{i,j} = x_i^T x_j; w_{i,j} = \text{Softmax}(\tilde{w}_{i,j})$
- Each input is query, key, value .
- Transformers are permutation invariant -> positional encodings

No Free Lunch Theorem Averaged over all possible functions f, all algorithms are equivalent in terms of $\mathbb{E}[E|d]$.