Foundations of Data Science

Ralf Blöchlinger

Fall 2024

Contents

Definition of Machine Learning

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.[1]

$$\text{Learning} = \text{Representation} + \text{Evaluation} + \text{Optimisation}$$

- Representation: Hypothesis space, which models to consider
- Evaluation: Objective function
- Optimization: Search method for optimizing objective function

Perceptron

$$\text{sign}(\boldsymbol{w} \cdot \mathbf{x}) = \begin{cases} +1 & \text{if } \boldsymbol{w} \cdot \mathbf{x} \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Algorithm: Initialize $\mathbf{w} = 0$. Repeat until convergence, for $t = 1, ..., n$:

1. Compute prediction: $y' = \text{sign}\left(\mathbf{x}_t \cdot w\right)$
2. Update parameters:
   - If $y' \neq y_t$ then $\mathbf{w} := \mathbf{w} + y_t \mathbf{x}_t$[2]
   - else, leave $\mathbf{w}$ unchanged

1 Linear Regression

Linear model:

$$y = w_0 + x_1 w_1 + \cdots + x_D w_D + \epsilon$$

where y is a $N \times 1$ vector.

Estimate $\hat{\mathbf{w}}$ by optimizing *Least Squares objective function*:

$$\arg\min_{\mathbf{w}} \frac{1}{2N} \sum_{i=1}^{N} \left(\mathbf{x}_i^\top \mathbf{w} - y_i\right)^2 = \frac{1}{2N} (\mathbf{X}\mathbf{w} - \mathbf{y})^\top (\mathbf{X}\mathbf{w} - \mathbf{y})$$

where we usually include a vector of ones in the matrix $\mathbf{X}$.

[1] E.g., E is training data set consisting of faces, T is to put boxes around faces, P is number of correctly identified faces.

[2] If $y' = 1, y_t = -1$, we want to make $\mathbf{w} \cdot \mathbf{x}_t$ smaller. Note that $\mathbf{w}' \cdot \mathbf{x}_t = \mathbf{w}\mathbf{x}_t + y\mathbf{x}_t^\top \mathbf{x}_t < \mathbf{w} \cdot \mathbf{x}_t$.

- Simple linear regression with one predictor:

$$w_0 = \bar{y} - w_1 \cdot \bar{x};\ w_1 = \frac{\widehat{\text{cov}}(x, y)}{\widehat{\text{var}}(x)}$$

- General solution:

$$\mathbf{w} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}$$

  - Complexity: $O(D^2 N + D^3)$
  - $\mathbf{X}^T\mathbf{X}$ invertible if full rank, i.e., rank is $D+1$ and $N > D$

- Predictions: $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$; for a single observation $\hat{y} = \mathbf{x} \cdot \mathbf{w}$

Huber Loss

Due to square, OLS is non-robust. Alternative: Huber-Loss function which is equal to OLS for small deviations and is an absolute value loss for larger values.

Definition 1. Given arbitrary but fixed parameters $\lambda, \mu \in \mathbb{R}$ with $\lambda, \mu > 0$, the **Huber loss** is given by the function $h_{\lambda,\mu} : \mathbb{R} \mapsto \mathbb{R}$ such that

$$h_{\lambda,\mu}(z) = \begin{cases} \lambda\left(|z| - \frac{\lambda}{4\mu}\right), & \text{if } |z| \geq \frac{\lambda}{2\mu} \\ \mu z^2, & \text{otherwise} \end{cases}$$

$\diamond$

Another alternative involves using the absolute error instead of the squared error. This problem cannot be solved in closed-form as the absolute value function is not differentiable everywhere.

1.1 Basis Expansion

Idea: Capture non-linearities in linear regression by including transformations of feature space.

- Example: Quadratic model with 2 predictors

$$\psi(\mathbf{x}) = \left[1, x_1, x_2, x_1^2, x_2^2, x_1 x_2\right]^\top$$

- degree d polynomial, D dimensions $\implies$ $O(D^d)$ features
- Computational cost dominated by dot product

- Can reduce cost by using kernels. For some expansion ϕ a kernel function computes the dot product efficiently[3]

$$\kappa\left(\mathbf{x}', \mathbf{x}\right) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, \quad \kappa\left(\mathbf{x}', \mathbf{x}\right) = \phi\left(\mathbf{x}'\right) \cdot \phi(\mathbf{x})$$

- Polynomial Kernel: $\kappa_{\text{poly}}\ \left(\mathbf{x}', \mathbf{x}\right) = \left(\mathbf{x} \cdot \mathbf{x}' + \theta\right)^d$
- Radial Basis Function kernel: $\kappa_{\text{RBF}}\left(\mathbf{x}', \mathbf{x}\right) = \exp\left(-\gamma \left\|\mathbf{x} - \mathbf{x}'\right\|^2\right)$.
  - Hyperparameters: center and width. Narrow kernels (large γ) increase overfitting risk.
  - Dissimilarity measure, κ close to 1 if points are close.

We use kernels as features. For some kernel basis expansion, the linear model is

$$\boldsymbol{y} = w_0 + w_1 \kappa\left(\mu_1, \mathbf{x}\right) + \cdots + w_M \kappa\left(\mu_M, \mathbf{x}\right) + \epsilon = \mathbf{w} \cdot \boldsymbol{\psi}(\mathbf{x}) + \epsilon$$

where centers μ_i are hyperparameters.

Choosing γ:

- High γ, narrow width: overfitting, can be reduced with more data
- Low γ, wide width: underfitting
- *Curse of dimensionality*: In high-dimensional space, we easily underfit as Euclidian distances between all data points is large and similar. Recall that kernels measure similarity. Therefore, kernels lose power in large dimensions. Dataset needs to be exponentially large in dimension or underfitting occurs.

Radial basis kernels are universal but too powerful (overfitting) and computationally expensive.

[3]The idea is to directly compute the kernel function in the original space instead of first manually creating the expansion and then computing the dot product of this expansion (i.e., in the high-dimensional space induced by the expansion). For instance, for $\kappa_{\text{poly}}\ \left(\mathbf{x}', \mathbf{x}\right) = \left(\mathbf{x} \cdot \mathbf{x}'\right)^d$, we can calculate this in expansion form as:

$$\left(\mathbf{x} \cdot \mathbf{x}'\right)^d = \sum_{n_i \geq 0, \sum_i n_i = d} \underbrace{\sqrt{C\left(d; n_1, \ldots, n_D\right)} x_1^{n_1} \cdots x_D^{n_D}}_{\text{one row in } \phi_{\text{odlv}}\ (\mathbf{x})} \underbrace{\sqrt{C\left(d; n_1, \ldots, n_D\right)} \left(x_1'\right)^{n_1} \cdots \left(x_D'\right)^{n_D}}_{\text{one row in } \phi_{\text{olly}}\ (\mathbf{x}')}$$

which is $O(D^d)$. Instead directly computing $\left(\mathbf{x} \cdot \mathbf{x}'\right)^d$ is $O(D + \log d)$. For the RBF kernel, the expansion involves a projection onto an infinte dimensional space, so using an expansion is unfeasible. For some kernels, we are directly interested in using the kernels as features, as we compare our data points to some fixed vectors. For other expansions, such as polynomial expansions, whether we can use kernels depends on the way in which the polynomial expansion enters the objective function or solution (i.e., whether it occurs in form of a dot product.)

2 Learning Curves, Validation, Feature Selection

2.1 Learning Curves

Fundamental Goal of ML: Generalizing beyond the training set.

- No free lunch theorem: Impossible to beat random guessing over all possible functions one can learn
- Context knowledge can limit which functions are reasonable

Bias-Variance Tradeoff:

$$\text{Generalisation error} = \text{bias} + \text{variance}$$

- Bias: consistently learn wrong thing $\rightarrow$ Underfitting
- Variance: Learn random things irrespetive of true signal $\rightarrow$ Overfitting
- Averaging high variance models can reduce overfitting $\rightarrow$ *ensemble models*

Learning curves:

1. Split data into training and testing set.
2. Train on increasing sizes of data.
3. Plot train and test error as function of training data size
4. If test error $>$ train error, but decreasing in data size, more data would be useful.
5. If test error and train error are close, more data would not be useful. Potentially a good-working model or underfitting.

2.2 Regularisation

Three ways to avoid overfitting:

1. Regularisation
2. Cross-validation
3. Statistical signifiance tests before adding new features (e..g, χ^2-test)

Idea of regularisation: add penalty term for model parameters.

Ridge Regression

$$\mathcal{L}_{\text{ridge}}\left(\mathbf{w}\right) = \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^{\top}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) + \lambda \sum_{i=1}^{D} w_i^2$$

ℓ_2-regularisation on weights.

- Penalty on all coefficients but intercept (intercept is output translation, not model complexity)
- Standardise input to avoid dependency on scaling $x' = (x - \mu)/\sigma$
- Additionally demean $\mathbf{y}$ s.t. $w_0 = 0$. Can then be rewritten as

$$\mathcal{L}_{\text{ridge}}\left(\mathbf{w}\right) = \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^{\top}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) + \lambda \mathbf{w}^{\top}\mathbf{w}$$

- Closed-form solution:

$$\mathbf{w}_{\text{ridge}} = \left(\mathbf{X}^{\top}\mathbf{X} + \lambda \mathbf{I}_D\right)^{-1}\mathbf{X}^{\top}\mathbf{y}$$

- Note: perturbed matrix, $\mathbf{X}^T\mathbf{X}$ is psd, so $\mathbf{X}^{\top}\mathbf{X} + \lambda \mathbf{I}_D$ is pd and invertible
- Can rewrite as Lagrangian: (Inverse relation between R and λ)

$$\min \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^{\top}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) \text{ subject to } \mathbf{w}^{\top}\mathbf{w} \leq R$$

Least Absolute Shrinkage and Selection Operator

$$\mathcal{L}_{\text{lasso}}\left(\mathbf{w}\right) = \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^{\top}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) + \lambda \sum_{i=1}^{D} |w_i|$$

- ℓ_1 penalty
- No closed form solution
- Contout or objective function is mix between oval and diamond when viewing level sets. Solutions tend to be on corner of diamond restriction on $|\mathbf{w}|$ (Figure 1). We therefore have feature selection as some coefficients are often $0 \rightarrow$ *sparse models*

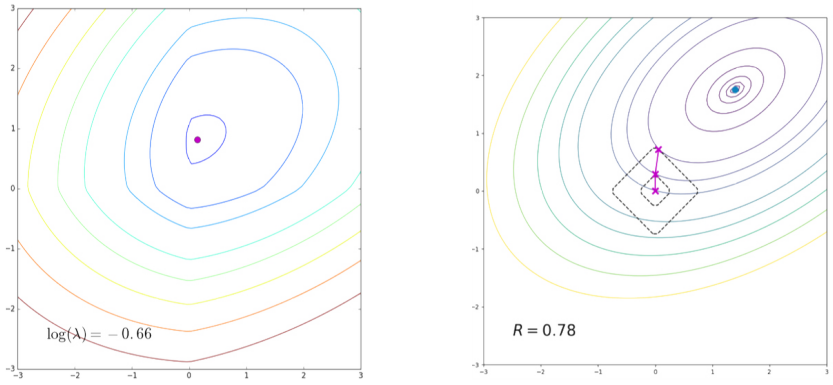

Figure 1: Left: Contour line of objective function. Right: Contour line of MSE with LASSO penalisation term in dashed lines.

2.3 Validation

Suppose we want to choose the optimal value of one hyperparemter.

1. Divide the data into parts: training, validation
    - Testing data is separate from both training and validation data
2. For each value of λ, train model using the training set and evaluate on the validation set
3. Pick the value of λ that minimises the validation error[4]

Multiple hyperparameters: **Grid search**

- Iterate over all possible combinations of hyper-parameter values $D_1 \times \cdots \times D_k$
- Perform cross-validation for each such combination

Cross Validation

When data is scarce, instead of splitting as training and validation, divide data into K folds (parts):

- Use $K-1$ folds for training and 1 fold as validation

[4]Often U-shaped curve. E.g., LASSO with low λ overfits, with high λ underfits

- Validation error for fixed hyper-parameter values: average over all runs (i.e., train with same hyper-parameter on all K possible combinations of $K-1$ folds and calculate average of error on each validation fold.)
- When $K = N$, LOOCV

2.4 Feature Selection

Goal: Given large feature set, select relevant features for model.

- Data often contains redundant or irrelevant features, e.g., due to strong correlations or not predictive power
- Selecting only relevant features reduces overfitting risk, avoids curse of dimensionality, yields simpler models, reduces training time

Each feature selection method usually consists of two components:

1. Search technique for subsets of given features
2. Evaluation measure to rank different subsets

Approaches:

1. **Wrapper methods**: train new model for each subsets.
   - Exhaustive search unfeasible as 2^n subsets of n features
   - Instead, e.g., forward stepwise selection[5]
2. **Filter methods**: proxy measure as score instead of test error
   - E.g., calculate correlation or mutual information[6] between features and target
3. **Embedded methods**: Perform feature selection while constructing model
   - E.g., LASSO or elastic net regularisation

[5] For D features, for each $1 \leq i \leq D$: What is best i feature model, given previously selected $i-1, i-2, \ldots$ features. Note: $O(D^2)$ models to train. Can be implemented as same complexity as one model for linear regression.

[6] Mutual information for two RVs X, Y is given by $I(X,Y) = \sum_x \sum_y p(X = x, Y = y) \cdot \log \frac{p(X=x, Y=y)}{p(X=x) \cdot p(Y=y)}$. Approximate by bucketizing features into discrete bins and counting fraction in training set for each bin.

3 Optimization

Two general approaches:

1. Convex optimization problem → use blackbox solvers
2. Gradient-based optimization methods[7]

3.1 MLE

Alternative view for ML: explicitly formulate the deviation (or uncertainty) from the model using probability theoretic notions.

- Given: Observations $x_1, \ldots, x_N$ iid drawn from distr p
- Likelihood of observing $x_1, \ldots, x_N \approx$ probability of making these observations assuming they are generated according to p
  - p has a parametric form, e.g., depends on the parameter vector $\boldsymbol{\theta}$
  - $p\left(x_1, \ldots, x_N \mid \boldsymbol{\theta}\right)$: joint probability distr. of observations given $\boldsymbol{\theta}$
- *Maximum Likelihood Principle*: Pick parameter $\boldsymbol{\theta}$ that maximises the likelihood
$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p\left(x_1, \ldots, x_N \mid \boldsymbol{\theta}\right)$$
- Since the observations are i.i.d.: $p\left(x_1, \ldots, x_N \mid \theta\right) = \prod_{i=1}^{N} p\left(x_i \mid \theta\right)$

MLE for Linear Regression

Assumption: Give $\mathbf{x}$ and $\mathbf{w}$, y is normal with mean $\mathbf{w}^\top \mathbf{x}$ and variance σ^2

$$p(y \mid \mathbf{w}, \mathbf{x}) = \mathcal{N}\left(\mathbf{w}^\top \mathbf{x}, \sigma^2\right) = \mathbf{w}^\top \mathbf{x} + \mathcal{N}\left(0, \sigma^2\right)$$

Equivalently, $\epsilon \sim \mathcal{N}\left(0, \sigma^2\right)$. We assume $\mathbf{X}$ to be fixed (non-random).

$$p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}, \sigma) = \left(\frac{1}{2\pi\sigma^2}\right)^{N/2} \exp\left(-\frac{1}{2\sigma^2}(\mathbf{X}\mathbf{w} - \mathbf{y})^\top (\mathbf{X}\mathbf{w} - \mathbf{y})\right)$$
$$\operatorname{NLL}(\mathbf{y} \mid \mathbf{X}, \mathbf{w}, \sigma) = \frac{1}{2\sigma^2}(\mathbf{X}\mathbf{w} - \mathbf{y})^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) + \frac{N}{2}\log\left(2\pi\sigma^2\right)$$

[7] Not blackbox, optimization hyperparameters affect performance

- Maximizing likelihood is equivalent to maximizing log-likelihood (or min NLL) as log is strongly montone.
- Objective function identical to Least Squares up to additive and multiplicative constant.
- Same estimator for MLE and OLS
- Can create confidence intervals for new predictions $y_{\text{new}} \sim \widehat{y}_{\text{new}} + \mathcal{N}\left(0, \sigma^2_{\text{ML}}\right)$.[8]

Robust Regression

Use noise distribution with fatter tails, e.g., Laplacian.

$$p\left(y_1, \ldots, y_N \mid \mathbf{x}_1, \ldots, \mathbf{x}_N, \mathbf{w}, b\right) = \prod_{i=1}^{N} \frac{1}{2b} \exp\left(-\frac{\left|y_i - \mathbf{w}^\top \mathbf{x}_i\right|}{b}\right)$$

$$\text{NLL}(\mathbf{y} \mid \mathbf{X}, \mathbf{w}, b) = \frac{1}{b} \sum_{i=1}^{N} \left|y_i - \mathbf{w}^\top \mathbf{x}_i\right| + N \log(2b)$$

- Identical to linear regression with absolute value objective function instead of squared sum of residuals $\rightarrow$ robust to outliers
- No closed form solution, not differentiable anywhere
- Can be solved as convex linear optimization problem

3.2 Convex Optimization

Definition 2. A set $C \subseteq \mathbb{R}^D$ is convex if for any $\mathbf{x}, \mathbf{y} \in C$ and $\lambda \in [0, 1]$, it holds $\lambda \mathbf{x} + (1 - \lambda)\mathbf{y} \in C$ ◇

Definition 3. A function $f : \mathbb{R}^D \rightarrow \mathbb{R}$ defined on a convex domain is convex if: for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^D$ where f is defined and $0 \leq \lambda \leq 1$,

$$f(\lambda \cdot \mathbf{x} + (1 - \lambda) \cdot \mathbf{y}) \leq \lambda \cdot f(\mathbf{x}) + (1 - \lambda) \cdot f(\mathbf{y})$$

◇

Examples of convex functions:

- Affine functions: $f(\mathbf{x}) = \mathbf{b}^\top \mathbf{x} + c$

[8] Note that this is imprecise as we estimate the coefficients and thus need to account for this additional uncertainty.

- Quadratic functions: $f(\mathbf{x}) = 1/2 \cdot \mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b}^\top \mathbf{x} + c$, where $\mathbf{A}$ is symmetric positive semidefinite
- Nonnegative weighted sums of convex functions: Given convex functions $f_1, \ldots, f_n$ and $w_1, \ldots, w_n \in \mathbb{R}_{\geq 0}$, the following is a convex function $f(\mathbf{x}) = \sum_{i=1}^n w_i \cdot f_i(\mathbf{x})$
- Norms: $\|\cdot\|_p$ except $p = 0$

Convex optimization problem

Given convex functions $f, g_1, \ldots, g_m$ and affine functions $h_1, \ldots h_n$:

$$\begin{aligned} &\text{minimise } f(\mathbf{x}) \\ &\text{subject to } g_i(\mathbf{x}) \leq 0 \quad i \in [m] \\ &h_j(\mathbf{x}) = 0 \quad j \in [n] \end{aligned}$$

Goal: When S is set of feasible functional values, want to find optimal value $v^* = \min S$ and optimal point $\mathbf{x}^* = \arg\min S$.[9]

Theorem 1. *For any convex optimisation problem, all locally optimal points are globally optimal.*

Proof. By contradiction, asumme x local optimum, y global optimum with $f(y) < f(x)$. Since local optimum $\exists B$ s.t. $f(x) < f(x')$ for any feasible x' with $||x - x'||_2 < B$. Choose $z = \lambda y + (1-\lambda)x$ and $\lambda = B/(2||x-y||_2)$. Then $||x - z||_2 < B$ and $f(z) \leq \lambda f(y) + (1-\lambda) f(x) < f(x)$ contradicting that x is a local optimum. □

Common examples:

- Linear program:[10]
$$\begin{aligned} \text{minimize } &\mathbf{c}^\top \mathbf{x} + d \\ \text{subject to } &\mathbf{A}\mathbf{x} \leq \mathbf{e} \\ &\mathbf{B}\mathbf{x} = \mathbf{f} \end{aligned}$$
- Quadratically Constrained Quadratic Programming:
$$\begin{aligned} \text{minimize } &\frac{1}{2}\mathbf{x}^\top \mathbf{B}\mathbf{x} + \mathbf{c}^\top \mathbf{x} + d \\ \text{subject to } &\frac{1}{2}\mathbf{x}^\top \mathbf{Q}_i \mathbf{x} + \mathbf{r}_i^\top \mathbf{x} + s_i \leq 0 \; i \in [m] \\ &\mathbf{A}\mathbf{x} = \mathbf{b} \end{aligned}$$

[9] For infeasible problems, we set $v^* = \infty$, for unbounded problems $v^* = -\infty$.

[10] Space of feasible solutions is polytope defined by intersection of half-spaces of constraints. Solution is always at an edge (*vertex*) of polytope. In particular, we want to find the vertex which projected onto $-\mathbf{c}$ gives the longest vector.

No closed form solutions but efficient algortihms (polynomial time) exist.

Showing that a linear program is equivalent to some maximization problem:

1. Show that there exists some solution to linear program
2. Show that the optimal solution for lin prog coincides with solution of mathematical optimization problem

Linear regression is quadratic convex program. Lasso can be rephrased as quadratic program with linear constraints.

3.3 Gradient Descent

Geometry of Gradient and Hessian

Theorem 2. *If a function f is differentiable, the gradient of f at a point is either zero or perpendicular to the contour line of f at that point.*

- Intuition: Along a contour line the functional value does not change thus $\nabla_{\mathbf{x}} f \cdot \mathbf{v} = 0$ if $\mathbf{v}$ is vector tangent to contour line.

Theorem 3. *The gradient $\nabla_{\mathbf{v}} f$ of a function f points in the direction of steepest ascent.*

- Consider the directional derivative:

$$\nabla_{\mathbf{v}} f(\mathbf{a}) = \lim_{h \to 0} \frac{f(\mathbf{a} + h\mathbf{v}) - f(\mathbf{a})}{h} = \nabla f(\mathbf{a}) \cdot \mathbf{v}$$

- We want to find the unit vector maximizing this quantitiy, from dot product definition,

$$\nabla f(\mathbf{a}) \cdot \mathbf{v} = \|\nabla f(\mathbf{a})\| \|\mathbf{v}\| \cos\theta.$$

  which is maximized for $\theta = 0$, i.e., $\mathbf{v}$ points into direction of gradient.

Hessian is symmetric if all second-order derivatives exist and are continuous. Captures curvature of surface. A saddle point is a minimum if all eigenvalues of $\mathbf{H}$ at $\mathbf{x}^*$ are positive.

Gradient Descent Algorithm

At each point t, update weights using

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \mathbf{g}_t = \mathbf{w}_t - \eta_t \nabla f(\mathbf{w}_t),$$

where $\eta > 0$ is learning rate.

Gradient descent can also be used if closed-form solutions exist. E.g., OLS closed-form is $O(ND^2 + D^3)$, each gradient step

$$\nabla_{\mathbf{w}} \mathcal{L} = 2\left(\mathbf{X}^\top \mathbf{X}\mathbf{w} - \mathbf{X}^\top \mathbf{y}\right) = 2\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y})$$

is $O(ND)$.[11] If number of iterations $K < D$, gradient descent is more efficient.

Choosing step size:

- Idea: smaller steps the closer to optimum
- Constant step size: $\eta_t = \mathrm{c}$
- Decaying step size: $\eta_t = \mathrm{c}/t$ for constant c. Another common decay rate: $\frac{1}{\sqrt{t}}$
- Backtracking line search:
  1. Start with c/t
  2. Check if the function value decreases after the weight update, i.e., if $f(\mathbf{w}_t - \eta_t \boldsymbol{\nabla} f(\mathbf{w}_t)) < f(\mathbf{w}_t)$ holds.
  3. If the decrease condition is not met, multiply η_t by a decaying factor to decrease step size
  4. Repeat the previous two steps until the decrease condition is met.

Convergence tests:

- Fixed number of iterations
- Small change in the function value: Terminate if $|f(\mathbf{w}_{t+1}) - f(\mathbf{w}_t)| \leq \epsilon_1$
- Small change in the parameter values: Terminate if $\|\mathbf{w}_{t+1} - \mathbf{w}_t\|_2 \leq \epsilon_2$

[11] First compute $Xw - y$.

Stochastic Gradient Descent

Gradient is expensive since we need to take average of gradient of objective function for each observation. Instead, only take one data point (*online learning*) or a mini-batch to compute gradient.

Example: For Ridge,

$$\nabla_{\mathbf{w}}\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\nabla_{\mathbf{w}}\ell\left(\mathbf{w};\mathbf{x}_i,y_i\right) + \lambda\nabla_{\mathbf{w}}\mathcal{R}(\mathbf{w})$$

Pick a random datapoint $(\mathbf{x}_i, y_i)$ and evaluate $\mathbf{g}_i = \nabla_{\mathbf{w}}\ell\left(\mathbf{w};\mathbf{x}_i,y_i\right)$. Then,

$$\mathbb{E}\left[\mathbf{g}_i\right] = \frac{1}{N}\sum_{i=1}^{N}\nabla_{\mathbf{w}}\ell\left(\mathbf{w};\mathbf{x}_i,y_i\right)$$

In expectation $\mathbf{g}_i$ points in the same direction as the entire gradient (except for the regularisation term).[12]

- SGD will jump around a lot more, more variance
- In practice, mini-batch gradient descent performs far better

Subgradient Descent

Consider LASSO.

- Due to absolute value function, this is not differentiable everywhere $\rightarrow$ problem for gradient descent.
- Non-unique slope but can pick one possible slope value

Definition 4 (Sub-Derivatives and Sub-Gradients)**.** A sub-derivative of a univariate convex function f at a point x_0 is a scalar g such that

$$f(x) \geq f\left(x_0\right) + g\cdot\left(x - x_0\right) \qquad \text{for all } x.$$

A sub-gradient of a multivariate convex function f at a point $\mathbf{x}_0$ is a vector $\mathbf{g}$ such that

$$f(\mathbf{x}) \geq f\left(\mathbf{x}_0\right) + \mathbf{g}^{\top}\left(\mathbf{x} - \mathbf{x}_0\right) \qquad \text{for all } \mathbf{x}.$$

$\diamond$

We thus replace the gradient (in gradient descent) with a subgradient at a point where the function is not differentiable.

[12] Actually, it also holds for the regularisation term. However, if one were to use a sum instead of mean in the lossfunction, one should scale up the single function gradient by a factor N to ensure that the relative scale is correct. The regularisation term usually does not depend on the specific observation chosen.

Constrained convex optimisation and gradient descent

Suppose we have $\min f(\mathbf{x})$ subject to additional constraints $\mathbf{w}_C \in C$. We can use projected gradient descent.

$$\begin{aligned} \text{Gradient step:} \quad & \mathbf{z}_{t+1} = \mathbf{w}_t - \eta_t \nabla f(\mathbf{w}_t) \\ \text{Projection step:} \quad & \mathbf{w}_{t+1} = \underset{\mathbf{w}_C \in C}{\operatorname{argmin}} \|\mathbf{z}_{t+1} - \mathbf{w}_C\| \end{aligned}$$

I.e., we first calculate the gradient while ignoring constraints and then project our updated value back into the set of feasible values C.[13]

Newton's Method

Newton's method is a second order method which uses not only the first but also the second derivative (or Hessian in multivariate case). In some cases, this can lead to faster convergence. Instead of fitting a linear line to a point with the same slope as the original function, it fits a paraboloid that matches first and second derivatives.[14]

Consider a Taylor series expansion:

$$f(x) \approx f(x_k) + (x - x_k) f'(x_k) + \frac{1}{2}(x - x_k)^2 f''(x_k)$$

A minimzer is obtained by setting the derivative to zero:

$$f'(x_k) + (x^* - x_k) f''(x_k) = 0$$

which yields the updating rule

$$x_{k+1} = x^* = x_k - f'(x_k)\left[f''(x_k)\right]^{-1}.$$

We can extend this to higher dimensions using

$$f(\mathbf{x}) \approx f(\mathbf{x}_k) + \mathbf{g}_k^\top (\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^\top \mathbf{H}_k (\mathbf{x} - \mathbf{x}_k)$$

The gradient of f w.r.t. $\mathbf{x}$ is

$$\boldsymbol{\nabla}_{\mathbf{x}} f = \mathbf{g}_k + \mathbf{H}_k (\mathbf{x} - \mathbf{x}_k)$$

Setting $\boldsymbol{\nabla}_{\mathbf{x}} f = 0$, we obtain the minimiser

$$\mathbf{x}^* = \mathbf{x}_k - \mathbf{H}_k^{-1} \mathbf{g}_k.$$

[13] Ridge using the penalty term objective is a more efficient approach than using it in "Lagrange" form as we can compute the gradient directly without the projection step.

[14] For gradient descent we needed a step size as the minimum of the tangent line is $-\infty$. For Newton we no longer need this, as the fitted curve has a finite minimum.

- Computational costs are high.
  - $D + \binom{D}{2}$ partial derivatives at each step, $O(ND^2)$
  - Inverse $O(D^3)$
  - Can be made more efficient by using Cholesky factorisation instead of calculating inverse. I.e., factorize $\mathbf{H}_k$ to solve

$$\mathbf{H}_k \left(\mathbf{x} - \mathbf{x}_k \right) = -\mathbf{g}_k$$

- Convergence
  - For convex f, Newton's method converges to stationary points of the quadratic approximation, which are global minima (under appropriate starting points and regularity conditions on f)[15]
  - For non-convex f, stationary points may not be minima nor in the decreasing direction of f

Note: non-convex problem requires gradient descent.

4 Classification

4.1 KNN

Idea: classify point into group by most common label among k nearest neighbors. Measure closeness to data points based on some distance metric, usually Euclidian distance.

- Training time $O(1)$
- Space complexity $O(ND)$
- Prediction complexity $O(NDk)$
  - If we assume k, D constant than prediction complexity is $O(N)$
  - Note: compute N distances to all other data points in data set $O(N)$. Using a max heap of k smallest distances so far $O(N \log k)$ or quick select algorithm $O(N)$ can achieve $O(N)$ overall complexity.

[15] In particular, we require strong convexity (i.e., strict convexity and additional continuity requirements). The method may overshoot if the function domain is not all of $\mathbb{R}$ or if $f''(x)$ becomes very small which leads to large adjustment steps.

4.2 Generative models

Idea: Model the full joint distribution of input $\mathbf{x}$ and output y given the model parameters $\boldsymbol{\theta}$, $p(\mathbf{x}, y \mid \boldsymbol{\theta})$.

Conditional distribution for y for new input $\mathbf{x}_{new}$ is

$$p\left(y = c \mid \mathbf{x}_{\text{new}}, \boldsymbol{\theta}\right) = \frac{p\left(\mathbf{x}_{\text{new}}, \boldsymbol{y} = c \mid \boldsymbol{\theta}\right)}{p\left(\mathbf{x}_{\text{new}} \mid \boldsymbol{\theta}\right)} = \frac{p(y = c \mid \boldsymbol{\theta}) \cdot p\left(\mathbf{x}_{\text{new}} \mid y = c, \boldsymbol{\theta}\right)}{\sum_{c^I=1}^{c} p\left(\boldsymbol{y} = c^I \mid \boldsymbol{\theta}\right) \cdot p\left(\mathbf{x}_{\text{new}} \mid y = c^I, \boldsymbol{\theta}\right)}$$

Modelling approach:

- Marginal distribution of outputs $p(y = c|\pi) = \pi_c$
- class-conditional distributions of input given class label does not directly depend on π_c, parametrised by θ_c, $p(\mathbf{x} \mid y, \theta_c)$

MLE

The probability for a single data point $(\mathbf{x}_i, y_i)$ is:

$$p\left(\mathbf{x}_i, y_i \mid \boldsymbol{\theta}, \boldsymbol{\pi}\right) = p\left(y_i \mid \boldsymbol{\pi}\right) \cdot p\left(\mathbf{x}_i \mid y_i, \boldsymbol{\theta}\right) = \prod_{c=1}^{c} \pi_c^{\mathbb{I}(y_i=c)} \cdot p\left(\mathbf{x}_i \mid y_i, \boldsymbol{\theta}\right)$$

The likelihood and the log-likelihood of the data are

$$p(\mathcal{D} \mid \boldsymbol{\theta}, \boldsymbol{\pi}) = \prod_{i=1}^{N} p\left(\mathbf{x}_i, y_i \mid \boldsymbol{\theta}, \boldsymbol{\pi}\right) = \prod_{i=1}^{N} \left(\prod_{c=1}^{c} \pi_c^{\pi(y_i=c)} \cdot p\left(\mathbf{x}_i \mid y_i, \boldsymbol{\theta}\right) \right)$$

$$\log p(\mathcal{D} \mid \boldsymbol{\theta}, \boldsymbol{\pi}) = \sum_{c=1}^{c} N_c \log \pi_c + \sum_{i=1}^{N} \log p\left(\mathbf{x}_i \mid y_i, \boldsymbol{\theta}\right)$$

Class prior

Maximise $\sum_{c=1}^{c} N_c \log \pi_c$ subject to $\sum_{c=1}^{c} \pi_c - 1 = 0 \rightarrow$ Solution: $\pi_c = \frac{N_c}{N}$.

Class conditional distributions

Change depending on different underlying assumptions.

1. **Naive Bayes**: assume class conditional distributions are independent given class label c.

   Class conditional distribution for data point:

   $$p\left(\mathbf{x}_i \mid y_i = c, \boldsymbol{\theta}\right) = p\left(\mathbf{x}_i \mid y_i = c, \boldsymbol{\theta}_c\right) = \prod_{j=1}^{D} p\left(x_{ij} \mid \boldsymbol{\theta}_{jc}\right)$$

   Joint distribution for data point:

   $$p\left(\mathbf{x}_i, y_i \mid \boldsymbol{\theta}, \boldsymbol{\pi}\right) = p\left(y_i \mid \boldsymbol{\pi}\right) \cdot p\left(\mathbf{x}_i \mid y_i, \boldsymbol{\theta}\right) = \prod_{c=1}^{C} \pi_c^{\mathbb{I}(y_i=c)} \cdot \prod_{c=1}^{C} \prod_{j=1}^{D} p\left(x_{ij} \mid \boldsymbol{\theta}_{jc}\right)^{\mathbb{I}(y_i=c)}$$

   Log likelihood:

   $$\log p(\mathcal{D} \mid \boldsymbol{\theta}, \boldsymbol{\pi}) = \sum_{c=1}^{C} N_c \log \pi_c + \sum_{c=1}^{C} \sum_{j=1}^{D} \sum_{i:y_i=c} \log p\left(x_{ij} \mid \boldsymbol{\theta}_{jc}\right)$$

   - May use different distributions for different features
   - Can skip missing features by leaving out of sum in both enumerator and denominator.[16]
   - Reduces number of parameters and thus avoids curse of dimensionality.[17]

2. **Gaussian Discriminant Analysis**: Class-conditional density is multivariate normal with class-specific mean and covariance matrix

   $$p\left(\mathbf{x} \mid \boldsymbol{y} = c, \boldsymbol{\theta}_c\right) = \mathcal{N}\left(\mathbf{x} \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\right)$$

   Log-likelihood:

   $$\log p(\mathcal{D} \mid \boldsymbol{\theta}) = \sum_{c=1}^{C} N_c \log \pi_c + \sum_{c=1}^{C} \left(\sum_{i:y_i=c} \log \mathcal{N}\left(\mathbf{x} \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\right) \right).$$

   Estimated parameters:

   $$\widehat{\boldsymbol{\mu}}_c = \frac{1}{N_c} \sum_{i:y_i=c} \mathbf{x}_i$$

   $$\widehat{\boldsymbol{\Sigma}}_c = \frac{1}{N_c} \sum_{i:y_i=c} \left(\mathbf{x}_i - \widehat{\boldsymbol{\mu}}_c\right)\left(\mathbf{x}_i - \widehat{\boldsymbol{\mu}}_c\right)^{\top}$$

[16] E.g., if x_1 is missing for a new observation, use

$$p\left(y = c \mid \mathbf{x}_{\text{new}}, \boldsymbol{\theta}\right) = \frac{\pi_c \cdot \prod_{j=2}^{D} p\left(x_j \mid y = c, \boldsymbol{\theta}_{cj}\right)}{\sum_{c'=1}^{C} p\left(y = c' \mid \boldsymbol{\theta}\right) \cdot \prod_{j=2}^{D} p\left(x_j \mid y = c', \boldsymbol{\theta}_{jc}\right)}$$

[17] Assume all features binary, then $C \cdot D$ parameters. Without independence assumption, $C \cdot 2^D$ as we assign probability of each class to each possible feature vector, of which there are 2^D.

No additional assumptions on $\boldsymbol{\Sigma}_c$, **Quadratic discriminant analysis**.

Assume two classes, decision boundary:

$$\frac{\pi_{c_1} |2\pi\boldsymbol{\Sigma}_{c_1}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_{c_1}\right)^\top \boldsymbol{\Sigma}_{c_1}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_{c_1}\right)\right)}{\pi_{c_2} |2\pi\boldsymbol{\Sigma}_{c_2}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_{c_2}\right)^\top \boldsymbol{\Sigma}_{c_2}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_{c_2}\right)\right)} = 1$$

Taking logs, quadratic curve in $\mathbf{x}$.[18]

3. **Linear Discriminant Analysis:** Special case of GDA with shared covariance matrices across classes.

$$\begin{aligned} p(y=c \mid \mathbf{x}, \boldsymbol{\theta}) &= \frac{\pi_c |2\pi\boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_c\right)^\top \boldsymbol{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_c\right)\right)}{\sum_{c'=1}^{C} \pi_{c'} |2\pi\boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_{c'}\right)^\top \boldsymbol{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_{c'}\right)\right)} \\ &= \frac{\pi_c \exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_c\right)^\top \boldsymbol{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_c\right)\right)}{\sum_{c'=1}^{C} \pi_{c'} \exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_{c'}\right)^\top \boldsymbol{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_{c'}\right)\right)} \end{aligned}$$

Can be simplified to:[19]

$$p(y=c \mid \mathbf{x}, \boldsymbol{\theta}) = \frac{\exp\left(\boldsymbol{\beta}_c^\top \mathbf{x} + \gamma_c\right)}{\sum_{c'} \exp\left(\boldsymbol{\beta}_{\beta'}^\top \mathbf{x} + \gamma_{c'}\right)} = \operatorname{softmax}(\boldsymbol{\eta})_c$$

Due to identical covariance matrices, decision boundaries become linear.

GDA has high risk of overfitting as $\approx C \cdot D^2$ parameters. Diagonal cov. matrix (Naive Bayes) or shared matrices (LDA) can reduce overfitting.

Note: Two-Class LDA:[20]

[18] With more than two classes, we will have piece-wise quadratic functions.

[19] Expand the term in the numerator as follows:

$$\begin{aligned} &\pi_c \exp\left(-\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_c\right)^\top \boldsymbol{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_c\right)\right) \\ =&\exp(\underbrace{\boldsymbol{\mu}_c^\top \boldsymbol{\Sigma}^{-1}}_{\boldsymbol{\beta}_c^\top}\mathbf{x} \underbrace{-\frac{1}{2}\boldsymbol{\mu}_c^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_c + \log \pi_c)}_{\gamma_c} \cdot \exp\left(-\frac{1}{2}\mathbf{x}^\top \boldsymbol{\Sigma}^{-1}\mathbf{x}\right) \\ =&\exp\left(\boldsymbol{\beta}_c^\top \mathbf{x} + \gamma_c\right) \cdot \exp\left(-\frac{1}{2}\mathbf{x}^\top \boldsymbol{\Sigma}^{-1}\mathbf{x}\right) \end{aligned}$$

Then apply a similar expansion the denominator and simplify.

[20] If features are multivariate normal with same covariance matrices across classes, there are two classes we can map LDA to logistic regression. However, due to different modellig approach, coefficients are unlikely to be the same.

$$
\begin{aligned}
p(y=1 \mid \mathbf{x}, \theta) &= \frac{\exp\left(\boldsymbol{\beta}_1^\top \mathbf{x} + \gamma_1\right)}{\exp\left(\boldsymbol{\beta}_1^\top \mathbf{x} + \gamma_1\right) + \exp\left(\boldsymbol{\beta}_0^\top \mathbf{x} + \gamma_0\right)} \\
&= \frac{1}{1 + \exp\left(-\left(\left(\boldsymbol{\beta}_1 - \boldsymbol{\beta}_0\right)^\top \mathbf{x} + \left(\gamma_1 - \gamma_0\right)\right)\right)} \\
&= \text{sigmoid}\left(\left(\boldsymbol{\beta}_1 - \boldsymbol{\beta}_0\right)^\top \mathbf{x} + \left(\gamma_1 - \gamma_0\right)\right)
\end{aligned} \tag{1}
$$

4.3 Logistic Regression

Discriminative classification: we only model conditional distribution of output given $\mathbf{x}, \mathbf{w}$, not a distribution over $\mathbf{x}$.

Idea: Given input $\mathbf{x}$, models with parameters $\mathbf{w}$ produce a value $f(\mathbf{x}, \mathbf{w}) \in [0, 1]$. We model the (binary) class labels as:

$$y \sim \text{Bernoulli}(f(\mathbf{x}, \mathbf{w}))$$

We assume a linear model composed with a sigmoid function. I.e.,

$$f(\mathbf{w} \cdot \mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x}) = 1/(1 + \exp\{-\mathbf{w} \cdot \mathbf{x}\})$$

1. For classification, set threshhold (usually at $1/2$):

$$\widehat{y}_{\text{new}} = \mathbb{I}\left(\sigma\left(\mathbf{w} \cdot \mathbf{x}_{\text{new}}\right) \geq \frac{1}{2}\right) = \mathbb{I}\left(\mathbf{w} \cdot \mathbf{x}_{\text{new}} \geq 0\right)$$

2. Linear class boundary:

$$\mathbf{x} \cdot \mathbf{w} = \log \frac{p_0}{1 - p_0}$$

$$p(y = 1 \mid \mathbf{x}, \mathbf{w}) = p(y = 0 \mid \mathbf{x}, \mathbf{w}) = 1/2 \Rightarrow \mathbf{x} \cdot \mathbf{w} = 0$$

MLE

Likelihood:

$$p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = \prod_{i=1}^{N} \sigma\left(\mathbf{w}^\top \mathbf{x}_i\right)^{y_i} \cdot \left(1 - \sigma\left(\mathbf{w}^\top \mathbf{x}_i\right)\right)^{1 - y_i} = \prod_{i=1}^{N} \mu_i^{y_i} \cdot (1 - \mu_i)^{1 - y_i}$$

with $\mu_i \overset{\text{def}}{=} \sigma\left(\mathbf{w}^\top \mathbf{x}_i\right)$.

1. Minimize negative log-likelihood.[21]

$$\text{NLL}(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = -\sum_{i=1}^{N} \left(y_i \log \mu_i + (1 - y_i) \log \left(1 - \mu_i \right) \right)$$

2. Gradient with respect to w

$$\nabla_{\mathbf{w}} \text{NLL}(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = \sum_{i=1}^{N} \mathbf{x}_i \left(\mu_i - y_i \right) = \mathbf{X}^\top (\boldsymbol{\mu} - \mathbf{y})$$

3. Hessian:

$$\mathbf{H} = \mathbf{X}^\top \mathbf{S} \mathbf{X}$$

   where $\mathbf{S}$ is a diagonal matrix with $S_{ii} = \mu_i \left(1 - \mu_i \right)$

If $\mathbf{X}$ is full rank, $\mathbf{H}$ is pd, thus, convex optimisation problem.

Iteratively Re-Weighted Least Squares

Newton update rule:

$$\begin{aligned}
\mathbf{w}_{t+1} &= \mathbf{w}_t - \mathbf{H}_t^{-1} \mathbf{g}_t \\
&= \mathbf{w}_t + \left(\mathbf{X}^\top \mathbf{S}_t \mathbf{X} \right)^{-1} \mathbf{X}^\top \left(\mathbf{y} - \boldsymbol{\mu}_t \right) \\
&= \left(\mathbf{X}^\top \mathbf{S}_t \mathbf{X} \right)^{-1} \mathbf{X}^\top \mathbf{S}_t \underbrace{\left(\mathbf{X} \mathbf{w}_t + \mathbf{S}_t^{-1} \left(\mathbf{y} - \boldsymbol{\mu}_t \right) \right)}_{\mathbf{z}_t}
\end{aligned}$$

Rearrangement yields:[22]

$$\begin{aligned}
\mathbf{w} &= (\underbrace{\left(\mathbf{S}_t^{1/2} \mathbf{X} \right)^\top}_{\tilde{\mathbf{x}}^\top} \underbrace{\mathbf{S}_t^{1/2} \mathbf{X}}_{\tilde{\mathbf{x}}})^{-1} \underbrace{\left(\mathbf{S}_t^{1/2} \mathbf{X} \right)^\top}_{\tilde{\mathbf{x}}^\top} \underbrace{\mathbf{S}_t^{1/2} \mathbf{z}_t}_{\tilde{\mathbf{y}}} \\
\mathbf{w} &= \left(\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}} \right)^{-1} \tilde{\mathbf{X}}^\top \tilde{\mathbf{y}}
\end{aligned}$$

I.e., our estimates at each step are the least squares solution to

$$\mathcal{L}(\mathbf{w}) = \sum_i \left(\tilde{\mathbf{x}}_i^\top \mathbf{w} - \tilde{y}_i \right)^2 = \sum_i \mathbf{S}_{t,ii} \left(\mathbf{x}_i^\top \mathbf{w} - z_{t,i} \right)^2$$

- Each step re-weights residuals by new diagonal matrix $\mathbf{S}$
- Each step uses new vector $\mathbf{z}_t$, depending on $\mathbf{w}_t$

[21] Note, $\text{NLL}\left(y_i \mid \mathbf{x}_i, \mathbf{w} \right)$ is the cross-entropy between y_i and μ_i for $y_i \in \{0, 1\}$.

[22] Here we make use of $\mathbf{S}_t$ being diagonal, thus symmetric, and having only non-negative entries. Thus we can write $\mathbf{S}_t = \mathbf{S}_t^{1/2} \mathbf{S}_t^{1/2}$ and have $\mathbf{S}_t^{1/2^\top} = \mathbf{S}_t^{1/2}$.

Multiclass Logistic Regression

- Paramters for each class $\mathbf{w}_c$
- Multi-class logistic model:

$$p(y=c \mid \mathbf{x}, \mathbf{W}) = \frac{\exp\left(\mathbf{w}_c^\top \mathbf{x}\right)}{\sum_{c'=1}^{C} \exp\left(\mathbf{w}_{c'}^\top \mathbf{x}\right)} = \text{softmax}\left(\left[\mathbf{w}_1^\top \mathbf{x}, \ldots, \mathbf{w}_C^\top \mathbf{x}\right]^\top\right)_c$$

- Likelihood is given similar to before as

$$\mathcal{L} = \prod_i \prod_c \left(\frac{\exp\left(\mathbf{w}_c^T \mathbf{x}\right)}{\sum_{c'=1}^{C} \exp\left(\mathbf{w}_{c'}^T \mathbf{x}\right)}\right)^{\mathbb{I}(y_i=c)}$$

- NLL convex:

$$NLL = -\sum_{i=1}^{N} \left[\left(\sum_{c=1}^{C} \mathbb{I}(y_i=c) \mathbf{w}_c^T \mathbf{x}_i\right) - \log\left(\sum_{c'=1}^{C} \exp\left(\mathbf{w}_{c'}^T \mathbf{x}_i\right)\right)\right]$$

- Can use basis expansion (linear decision boundary in ψ space, nonlinear in $\mathbf{x}$.)
- Can apply regularisation[23]

4.4 Performance Measurement

- TPR, sensitivity, recall: $\frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$
- FPR, fall-out: $\frac{\text{False Positives}}{\text{True Negatives} + \text{False Positives}}$
- Precision: $\frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$
- Accuracy: $\frac{\text{True Positives} + \text{True Negatives}}{N}$
- Confusion matrix: Table with predicted value 1/0 in rows, true value 1/0 in columns
- ROC: Plot TPR v FPR
- Precision recall curve: Plot TPR v precision

[23] For linearly separable data, this is necessary as increasing all coefficients by a positive factor η will still lead to separability but will increase the steepness and likelihood. The method will not converge and will overfit.

5 SVM

Idea: Among all lines that linearly separate data, find line which maximizes the smallest margin of any data point to the line (*Maximum Margin Principle*).

- Equation of hyperplane: $H \equiv \mathbf{w} \cdot \mathbf{a} + w_0 = 0$.
- Distance of point $\mathbf{x}$ to hyperplane:

$$\frac{|\mathbf{w} \cdot \mathbf{x} + w_0|}{\|\mathbf{w}\|_2} \tag{2}$$

- Assume linear separability, then[24]

$$y_i \left(\mathbf{w} \cdot \mathbf{x}_i + w_0\right) > 0$$

- Finite data set, can find $\epsilon > 0$ such that $y_i \left(\mathbf{w} \cdot \mathbf{x}_i + w_0\right) \geq \epsilon$. Thus,

$$y_i(\underbrace{\frac{\mathbf{w}}{\epsilon}}_{\text{new w}} \cdot \mathbf{x}_i + \underbrace{\frac{w_0}{\epsilon}}_{\text{new } w_0}) \geq 1$$

- Therefore, $1/||\mathbf{w}||_2$ is lower bound of margin as $|\mathbf{w} \cdot \mathbf{x} + w_0| \geq 1$. We maximize the lower bound instead. Equivalently, we minimize $||\mathbf{w}||_2^2$:

$$\begin{aligned} &\min \frac{1}{2}\|\mathbf{w}\|_2^2 \\ &\text{subject to:} \quad y_i \left(\mathbf{w} \cdot \mathbf{x}_i + w_0\right) \geq 1, \text{ for } 1 \leq i \leq N \end{aligned}$$

- Convex-quadratic optimisation problem with linear convex constraints
- If data is not linearly separable, require constraint to hold with slack:[25]

$$\begin{aligned} &\text{minimise:} && \tfrac{1}{2}\|\mathbf{w}\|_2^2 + C \textstyle\sum_{i=1}^N \zeta_i \\ &\text{subject to:} && y_i \left(\mathbf{w} \cdot \mathbf{x}_i + w_0\right) \geq 1 - \zeta_i \text{ for } 1 \leq i \leq N \\ & && \zeta_i \geq 0 \quad \text{for } 1 \leq i \leq N \end{aligned}$$

- Can be rewritten, using $\zeta_i = \max\left(0, 1 - y_i \left(\mathbf{w} \cdot \mathbf{x}_i + w_0\right)\right) \stackrel{\text{def}}{=} \ell_{\text{hinge}}\ (\mathbf{w}, w_0; \mathbf{x}_i, y_i)$.

$$\mathcal{L}_{\text{SVM}}\left(\mathbf{w}, w_0 \mid \mathbf{x}, \mathbf{y}\right) = \underbrace{\frac{1}{2}\|\mathbf{w}\|_2^2}_{\ell_2 \text{ regulariser}} + \underbrace{C \sum_{i=1}^{N} \ell_{\text{hinge}}\ (\mathbf{w}, w_0; \mathbf{x}_i, y_i)}_{\text{hinge loss}}$$

[24] I.e., $\mathbf{w} \cdot \mathbf{x}_i + w_0 > 0$ and $y_i = +1$ or $\mathbf{w} \cdot \mathbf{x}_i + w_0 < 0$ and $y_i = -1$

[25] This is an approximation to the NP-hard problem of finding a separator which makes the fewest possible mistakes on the training set.

Solving optimization problem:

- Lagrange:

$$\Lambda(\mathbf{w}, w_0; \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|_2^2 - \sum_{i=1}^{N} \underbrace{\alpha_i}_{\text{dual variables}} \underbrace{\left(y_i\left(\mathbf{w}\cdot\mathbf{x}_i + w_0\right) - 1\right)}_{\text{constraints}}$$

- Optimality conditions:

$$\frac{\partial \Lambda}{\partial w_0} = -\sum_{i=1}^{N} \alpha_i y_i = 0; \quad \nabla_{\mathbf{w}}\Lambda = \mathbf{w} - \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i = 0$$

- Optimal $\mathbf{w}$: $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$
- At optimality for support vectors: $y_i\left(\mathbf{w}\cdot\mathbf{x}_i + w_0\right) = 1$. Thus, using above optimality of $\mathbf{w}$ and $y_i^2 = 1$,

$$w_0 = \frac{1}{|S|}\sum_{\mathbf{x}_i \in S}\left(y_i - \sum_{j=1}^{N} \alpha_j y_j \left(\mathbf{x}_j \cdot \mathbf{x}_i\right)\right)$$

- Plug the optimal $\mathbf{w}$ and $\sum_{i=1}^{N} \alpha_i y_i = 0$ into Lagrangian:[26]

$$g(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i y_i \alpha_j y_j \left(\mathbf{x}_i \cdot \mathbf{x}_j\right)$$

$\implies$ find critical points of g such that $\alpha_i \geq 0$ and $\sum_{i=1}^{N} \alpha_i y_i = 0$

Extension to allow for slack, additional N choice variables ζ_i:

Primal Form	**Dual Form**
min: $\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\zeta_i$ subject to: $y_i(\mathbf{w}\cdot\mathbf{x}_i + w_0) \geq 1 - \zeta_i$ and $\zeta_i \geq 0$	max: $\sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j(\mathbf{x}_i\cdot\mathbf{x}_j)$ subject to: $\sum_{i=1}^{N}\alpha_i y_i = 0$ and $0 \leq \alpha_i \leq C$
- Quadratic convex problem - $D + N + 1$ variables - Constraints define a complex polytope	- Quadratic concave problem - N variables - Simple box constraints + 1 zero-sum constraint

Table 1: Primal and Dual Forms of the Optimization Problem

Reasons to prefer dual form:

[26] Term involving w_0 drops out due to $\sum_{i=1}^{N}\alpha_i y_i = 0$ constraint.

1. If $D \gg N$[27]
2. Natural kernelisation[28]
3. Number of non-zero $\alpha_i \ll N$ in practice. Only non-zero for **support vectors.**[29]

5.1 Kernel Calculation Rules

A function κ is a kernel that computes the dot product for some expansion ϕ. Gram matrix is the matrix consisting of kernels

$$\mathbf{K} = \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \kappa(\mathbf{x}_1, \mathbf{x}_2) & \cdots & \kappa(\mathbf{x}_1, \mathbf{x}_N) \\ \kappa(\mathbf{x}_2, \mathbf{x}_1) & \kappa(\mathbf{x}_2, \mathbf{x}_2) & \cdots & \kappa(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_N, \mathbf{x}_1) & \kappa(\mathbf{x}_N, \mathbf{x}_2) & \cdots & \kappa(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}$$

Definition 5. A function $\kappa(\mathbf{x}', \mathbf{x}) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, \kappa(\mathbf{x}', \mathbf{x}) = \phi(\mathbf{x}') \cdot \phi(\mathbf{x})$ is a **Mercer Kernel** if the gram matrix is positive semi-definite. ◇

Given kernels κ_1, κ_2, the following are kernels:

1. $\kappa(\mathbf{x}, \mathbf{x}') = c\kappa_1(\mathbf{x}, \mathbf{x}')$ with $c > 0$
2. $\kappa(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})\kappa_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}')$ f is a function
3. $\kappa(\mathbf{x}, \mathbf{x}') = q(\kappa_1(\mathbf{x}, \mathbf{x}'))$ q is a polynomial with non-negative coefficients
4. $\kappa(\mathbf{x}, \mathbf{x}') = \exp(\kappa_1(\mathbf{x}, \mathbf{x}'))$
5. $\kappa(\mathbf{x}, \mathbf{x}') = \kappa_1(\mathbf{x}, \mathbf{x}') \kappa_2(\mathbf{x}, \mathbf{x}')$
6. $\kappa(\mathbf{x}, \mathbf{x}') = \kappa_a(\mathbf{x}_a, \mathbf{x}'_a) \kappa_b(\mathbf{x}_b, \mathbf{x}'_b)$ $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b), \mathbf{x}' = (\mathbf{x}'_a, \mathbf{x}'_b), \kappa_a, \kappa_b$ are kernels
7. $\kappa(\mathbf{x}, \mathbf{x}') = \kappa_a(\mathbf{x}_a, \mathbf{x}'_a) + \kappa_b(\mathbf{x}_b, \mathbf{x}'_b)$
8. $\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{A} \mathbf{x}'$ where A is a symmetric positive semi-definite matrix

[27] Especially for case without slack, then primal form contains D variables

[28] Dot product in optimization problem can be replaced by Kernel matrix, $\sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \mathbf{K}_{i,j}$. Prediction on new point involves dot product, $\mathbf{w} \cdot \mathbf{x}_{\text{new}} = \sum_{i=1}^N \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}_{\text{new}})$.

[29] Only the points $\mathbf{x}_i$ with $\alpha_i > 0$ contribute to the solution $\mathbf{w}$. Such vectors $\mathbf{x}_i$ form the support of the solution, are points with least margin

To show that the dot product is a kernel, note that we can write the induced Gram matrix as a product of two matrices $\mathbf{K} = \psi^\top \psi$. From this it easily follows that $(\psi \mathbf{z})^\top \psi \mathbf{z} \geq 0$ for any $\mathbf{z}$ of appropriate dimensions.

We can also apply Kernels on discrete data, we are not restricted to Euclidian distances.

6 Neural Networks

Artificial Neuron:

- Generalisation of perceptron
- Output: $f(b + \mathbf{w} \cdot \mathbf{x})$
- Activation function $f : \mathbb{R} \to \mathbb{R}$

Neural Networks are compositions of artificial neurons into networks.

- Non-linear activation functions
- Loss functions usually non-convex $\to$ Gradient Descent

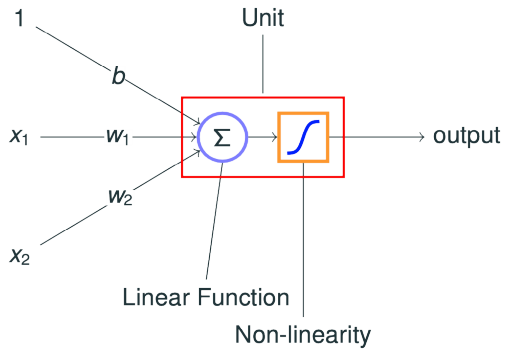

Figure 2: Simple Neural Network with one Artificial Neuron. If $f = \sigma(.)$, this is a logistic regression.

Multi-layer neural networks consist of input and output layers and multiple hidden layers. The idea behind using hidden layers is that layers should perform feature extraction.

We consider multi-layer perceptrons which consist of fully connected, feedforward layers. Notation:

- w_{ij}^{ℓ} : weight for the connection to unit i in layer ℓ from unit j in layer $\ell - 1$
- b_i^{ℓ} : bias term for unit i in layer ℓ
- z_i^{ℓ} : pre-activation for unit i in layer ℓ, sum of weighted outputs from $\ell - 1$
- a_i^{ℓ} : activation for unit i in layer ℓ, non-linear function of pre-activation.[30]
- Weights, etc. in matrix form: each *row* corresponds to one unit. E.g.,

$$\mathbf{W}^k = \begin{pmatrix} w_{11}^k & w_{12}^k \\ w_{21}^k & w_{22}^k \end{pmatrix}$$

  the first row corresponds to all weights for unit 1 in layer k.
- View derivatives as row vectors,

$$\frac{\partial f}{\partial \mathbf{z}} = \begin{bmatrix} \frac{\partial f}{\partial z_1} & \cdots & \frac{\partial f}{\partial z_n} \end{bmatrix}$$

- Derivative of multi-variate function with respect to vector, example:

$$\frac{\partial \mathbf{a}^2}{\partial \mathbf{z}^2} = \begin{bmatrix} \frac{\partial a_1^2}{\partial z_1^2} & \frac{\partial a_1^2}{\partial z_2^2} \\ \frac{\partial a_2^2}{\partial z_1^2} & \frac{\partial a_2^2}{\partial z_2^2} \end{bmatrix}$$

6.1 Model Learning

Idea:

- Compute derivatives for all parameters for each data point
- Average derivatives
- Perform gradient descent step to update parameters
  - Error propagated back to every layer
  - Weights in matrix adjust proportional to error and weights

Backpropagation excessively uses the chain rule to compute derivatives in modular and efficient fashion, by reusing deriatives to ease computation.

[30] Note: one activation function per unit, thus $\mathbf{a}^{\ell}$ is a column vector.

Chain Rule:

Functions $f : \mathbb{R}^n \to \mathbb{R}^k, g : \mathbb{R}^k \to \mathbb{R}^m, h = g \circ f : \mathbb{R}^n \to \mathbb{R}^m$.

For $\mathbf{x} \in \mathbb{R}^n$, let $\mathbf{z} = f(\mathbf{x}), h = g(f(x))$

$$\frac{\partial h}{\partial \mathbf{x}} = \frac{\partial h}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}}$$

- $\frac{\partial h}{\partial \mathbf{x}}$ is an $m \times n$ Jacobian matrix
- $\frac{\partial h}{\partial z}$ is an $m \times k$ Jacobian matrix
- $\frac{\partial \mathbf{z}}{\partial \mathbf{x}}$ is an $k \times n$ Jacobian matrix

Backpropagation Algorithm

Assume all layers are fully connected. Layer I consists of the following:

$$\mathbf{z}' = \mathbf{W}'\mathbf{a}'^{-1} + \mathbf{b}'$$
$$\mathbf{a}' = f_l\left(\mathbf{z}'\right)$$

where f_l is the non-linear activation in layer I.

If there are n_l units in layer I, then $\mathbf{W}^I \in \mathbb{R}^{n_l \times n_{l-1}}$.

Forward pass:

1. $\mathbf{a}^1 = \mathbf{x}$ (input)
2. $\mathbf{z}' = \mathbf{W}'\mathbf{a}^{I-1} + \mathbf{b}'$
3. $\mathbf{a}' = f_l\left(\mathbf{z}'\right)$ (May map vector- or elementwise)
4. $\ell\left(\mathbf{a}^L, y\right)$

Backward pass

Consider the last layer L:

$$\mathbf{z}^L = \mathbf{W}^L\mathbf{a}^{L-1} + \mathbf{b}^L$$
$$\mathbf{a}^L = f_L\left(\mathbf{z}^L\right)$$
$$\text{Loss:} \quad \ell\left(y, \mathbf{a}^L\right)$$
$$\frac{\partial \ell}{\partial \mathbf{z}^L} = \frac{\partial \ell}{\partial \mathbf{a}^L} \frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L}$$

Arbitrary layer I:

We have:

- Inputs into layer $I+1$, $\mathbf{a}^I$
- $\mathbf{z}'^{+1} = \mathbf{W}'^{+1}\mathbf{a}' + \mathbf{b}'^{+1}$ where $w_{j,k}^{I+1}$ weight on connectrion of k-th unit in layer I to j-th unit in layer $I+1$
- Activation function, $\mathbf{a}' = f(\mathbf{z}')$
- Derivative passed from previous layer, $\frac{\partial \ell}{\partial z^{I+1}}$

We can then compute:

$$\begin{aligned}\frac{\partial \ell}{\partial \mathbf{z}^I} &= \frac{\partial \ell}{\partial \mathbf{z}^{I+1}} \frac{\partial \mathbf{z}^{I+1}}{\partial \mathbf{z}^I} \\ &= \frac{\partial \ell}{\partial \mathbf{z}^{I+1}} \frac{\partial \mathbf{z}^{I+1}}{\partial \mathbf{a}^I} \frac{\partial \mathbf{a}^I}{\partial \mathbf{z}^I} \\ &= \frac{\partial \ell}{\partial \mathbf{z}^{I+1}} \mathbf{W}^{I+1} \frac{\partial \mathbf{a}^I}{\partial \mathbf{z}^I}\end{aligned}$$

We obtain the derivatives wrt w_{ij}^I and b_i^I using $\frac{\partial \ell}{\partial z^I}$:

- $\frac{\partial \ell}{\partial w_{ij}^I} = \frac{\partial \ell}{\partial z_i^I} \frac{\partial z_j^I}{\partial w_{ij}^I} = \frac{\partial \ell}{\partial z_i^I} a_j^{I-1}$ or $\frac{\partial \ell}{\partial \mathbf{W}^I} = \left(a^{I-1} \frac{\partial \ell}{\partial \mathbf{z}^I}\right)^\top$
- $\frac{\partial \ell}{\partial b_i^I} = \frac{\partial \ell}{\partial z_i^I}$ or $\frac{\partial \ell}{\partial \mathbf{b}^I} = \frac{\partial \ell}{\partial \mathbf{z}^I}$

Takeaway: Instead of computing each derivative separately, we only need to compute $\frac{\partial \ell}{\partial \mathbf{z}^I}$ and use the chain rule.

Computational requirements

- As many matrix multiplications as fully connected layers
- Performed twice during forward nacd backward pass
- Must store $\mathbf{a}^I, \mathbf{z}^I$, and $\frac{\partial \ell}{\partial \mathbf{z}^I}$ for each layer

6.2 Challenges in training deep NNs

Main challenges:

- Backpropagation is computationally expensive, instead use mini-batch SGD
- Overfitting
- May not converge

Saturation issue: Activation function may be in very flat region, thus gradients are very small. Using different loss functions may partially aleviate the issue.[31]

Vanishing gradient problem: Products of many gradients which are small can often be ≈ 0. In backpropagation, one also multiplies with weights, this can explode the gradient instead.

Can avoid saturation using ReLU: $f(z) = \max(0, z)$.

- Only saturates one side as derivative is constant for large z (thus avoids saturation)
- Sparsely activated, some neurons may not activate at all
- Dying ReLU problem: once deactivated, unlikely to reactivate
  - Can use leaky (parametric) ReLU instead, but can no longer have sparse activation:

$$f(z) = \begin{cases} az & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

Initialising weights: Due to non-convexity, we only find local optima. Initial weights are therefore important for convergence and solution.

- Weight initialisation for sigmoid/ReLU units
  - Suppose there are D weights $w_1, \ldots, w_D$
  - Draw w_i randomly from $\mathcal{N}\left(0, \frac{1}{D}\right)$
- Bias initialisation:
  - For sigmoid: Use a random value around 0
  - For ReLU: Use a small positive constant

[31] E.g., using a cross-entropy loss function leads to some cancelling out of activation gradients which can be useful.

Overfitting:

- Early stopping:
  - Measure performance on validation set after each gradient step
  - Stop when validation error stops decreasing
- Modified data:
  - Create fake data by changing brightness, etc.
  - Adversarial training: take trained model and create examples by adding noise impercetbily to human eye such that model makes errors.
- Bagging:
  - General idea, train models on different samples obtained by sampling with replacement.
  - In NN: *Dropout.*
  - Ignore fraction of hidden units in each traning step
  - Prevents co-adoption among different neurons
  - At test time, use complete model
  - Must re-scale weights. E.g., if dropout rate was $1/2$, halve all weights

7 Clustering

Idea: Unsupervised learning problem. Group together data into subsets such that dissimilarity between items within a group is minimized.

Two types of algorithms:

1. Feature based: points are vectors in $\mathbb{R}^D$
2. Dis-similarity based: data consists of pairwise dissimilarities of data points

K-Means Clustering

Partition data into subsets $C_1, ..., C_k$ with fixed k. Measure quality of partition:

$$W(C) = \frac{1}{2} \sum_{j=1}^{k} \frac{1}{|C_j|} \sum_{i,i' \in C_i} d\left(\mathbf{x}_i, \mathbf{x}_{i'}\right)$$

With euclidian distance, this simplifies to $W(C) = \sum_{j=1}^{k} \sum_{i \in C_j} \left\| \mathbf{x}_i - \boldsymbol{\mu}_j \right\|_2^2$ where $\boldsymbol{\mu}_j = \frac{1}{|C_j|} \sum_{i \in C_j} \mathbf{x}_i$.

Objective: minimise the sum of squares of distances to the mean within each cluster.

- Joint minimization over partitions and means is np-hard due to combinatorial nature of problem.
- Instead use **Lloyd's Algorithm**: After initialising means randomly, repeat until convergence (means to do not change).

  1. Construct clusters $C_1, \ldots, C_k$ by assigning the data to clusters represented by their means:

  $$C_j = \left\{ i \mid j = \operatorname*{argmin}_{j'} \left\| \mathbf{x}_i - \boldsymbol{\mu}_{j'} \right\|_2^2 \right\}$$

  2. Update means using the current cluster assignments:

  $$\boldsymbol{\mu}_j = \frac{1}{|C_j|} \sum_{i \in C_j} \mathbf{x}_i$$

- Algorithm always converges to a *local* minimum as only finitely many partitions.[32]
- k hyperparamter, chosen based on kink in test MSE
- Extensions:
  - k-medoids: use data points as cluster means, use any distance not necessarily euclidian.
  - k-center: Objective: maximum over all dissimilarities between data and anchor
- Note: Decision boundary will be linear. Same distance from two centers so $||\mathbf{x} - \mu_1||^2 = ||\mathbf{x} - \mu_2||^2$. Expaning and simplyfing leads to

  $$(\mu_1 - \mu_2)\, \mathbf{x} + \left(||\mu_1||^2 - ||\mu_2||^2 \right) / 2 = 0$$

  which is a hyperplane.

[32] At each iteration, means only change if a point changed clusters, in which case the mean updating will automatically lead to a lower value of the target function (as the mean minmizes the function $\sum_i (x_i - \mu)^2$). Thus value function always decreases until convergence. After each possible assignment was seen once it cannot be that the objective function still decreases, thus the algorithm must converge in k^N iterations.

Transforming between dissimilarities and numeric features

From $\mathbb{R}^D$ to dissimilarities:

Given feature vectors, we can obtain dissimilarities by applying

$$d(\mathbf{x}, \mathbf{x}') = f\left(\sum_{i=1}^{D} w_i d_i(x_i, x_i')\right).$$

From dissimilarities to $\mathbb{R}^D$:

Assume $D_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$, for $i, j \in [N]$ is given, we want to find $\mathbf{x}_i$ for $i = 1, \ldots, N$. I.e., we have dissimilarities and want to recover points in the euclidian space from this dissimilarity matrix.[33] Finding a Euclidian embedding of data that approximately respects original dissimilarities is called **multidimensional scaling.**

- Given assumption of euclidian distance, we have

$$\begin{aligned} D_{ij} &= \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \\ &= \mathbf{x}_i^\top \mathbf{x}_i - 2\mathbf{x}_i^\top \mathbf{x}_j + \mathbf{x}_j^\top \mathbf{x}_j \\ &= M_{ij} - 2M_{ij} + M_{ij} \end{aligned}$$

  where $\mathbf{M} = \mathbf{X}\mathbf{X}^\top$ is the $N \times N$ matrix of dot products: $M_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j$

- $\mathbf{M}$ can be recovered from $\mathbf{D}$ up to translation.
- Obtain $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_N$ from $\mathbf{M}$ using SVD: For $\mathbf{X} \in \mathbb{R}^{N \times D}$, the SVD is $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ where
  - $\mathbf{U} \in \mathbb{R}^{N \times D}$ and $\mathbf{V} \in \mathbb{R}^{D \times D}$ are orthonormal matrices
  - $\mathbf{\Sigma} \in \mathbb{R}^{D \times D}$ is diagonal with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_D \geq 0$
  - $\mathbf{U}^\top \mathbf{U} = \mathbf{V}^\top \mathbf{V} = \mathbf{I}_D$
- For square, symmetirc, psd matrix $\mathbf{M}$, we can obtain Eigendecomposition: $\mathbf{V} = \mathbf{U} : \mathbf{M} = \mathbf{U}\Sigma\mathbf{U}^\top$
  1. Let $\tilde{\mathbf{X}} = \mathbf{U}\mathbf{\Sigma}^{1/2}$
  2. Then

$$\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top = \mathbf{U}\mathbf{\Sigma}^{1/2}\left(\mathbf{U}\mathbf{\Sigma}^{1/2}\right)^\top = \mathbf{U}\mathbf{\Sigma}^{1/2}\mathbf{\Sigma}^{1/2}\mathbf{U}^\top = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^\top = \mathbf{M}$$

  3. We recover $\tilde{\mathbf{x}}_i$ as rows of $\mathbf{U}\mathbf{\Sigma}^{1/2}$.

[33] As we require points in the euclidian space to apply k-means.

More generally, we use a mercer kernel to define the similarity between points, thus $\mathbf{M}$ is symmetric and psd, we can find points $\tilde{\mathbf{x}}_i$ using above derivation.

We can view this problem as a non-convex optimization problem,

$$\underset{\tilde{\mathbf{x}}_1,\ldots,\tilde{\mathbf{x}}_N}{\operatorname{argmin}} \sum_{i \neq j} \left(\|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2^2 - D_{ij} \right)^2$$

which gives us the degree to which we can represent $\mathbf{D}$ in Euclidian space.

Hierarchical Clustering

Idea: build larger clusters and smaller clusters within. Two approaches:

1. Agglomerative: bottom-up, merge smaller clusters
2. Divisie: top-down, split larger clusters

Can be visualized in a *dendrogram*. Cutting this binary tree at a certain levels gives a data partition.

Suppose we have dissimilarity at datapoint level, e.g., $d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|$.[34] Define dissimilarity at cluster level, say C and C'

- Single Linkage
$$D(C, C') = \min_{\mathbf{x} \in C, \mathbf{x}' \in C'} d(\mathbf{x}, \mathbf{x}')$$
- Complete Linkage
$$D(C, C') = \max_{\mathbf{x} \in C, \mathbf{x}' \in C'} d(\mathbf{x}, \mathbf{x}')$$
- Average Linkage
$$D(C, C') = \frac{1}{|C| \cdot |C'|} \sum_{\mathbf{x} \in C, \mathbf{x}' \in C'} d(\mathbf{x}, \mathbf{x}')$$

Linkage-based clustering algorithm:

1. Initialise clusters as singletons $C_i = \{i\}$
2. Initialise clusters available for merging $S = \{1, \ldots, N\}$
3. Repeat

[34] Note: we can therefore apply hierarchical clustering if we only have dissimilarities, we do not require points in Euclidian space.

(a) Pick 2 most similar clusters, $(j,k) = \underset{j,k \in S}{\operatorname{argmin}}\, D(j,k)$

(b) Let $C_I = C_j \cup C_k$ for some new index I

(c) If $C_I = \{1, \ldots, N\}$, break;

(d) Update $S := (S \backslash \{j,k\}) \cup \{I\}$

(e) Update $D(i,I)$ for all $i \in S$ (using desired linkage property)

Spectral Clustering

Method for finding non-convex clusters.

Approach:

1. Construct K-nearest neighbour graph from data
   - One node for every point in dataset
   - (i,j) is an edge if either i is among the K nearest neighbours of j or vice versa
   - The weight of edge (i,j), if exists, given by similarity measure $s_{i,j}$

$$s_{i,j} = \exp\left(-\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2 / \sigma\right)$$

2. Use graph partitioning algorithms, e.g., Laplacian:[35]
   (a) $\mathbf{W}$ is the weighted adjacency matrix, $\mathbf{D}$ is (diagonal) degree matrix $D_{ii} = \sum_j W_{ij}$, Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{W}$
   (b) Calculate eigenvectors of $\mathbf{L}$
   (c) Use matrix of eigenvectors $[\mathbf{v}_2 \ldots \mathbf{v}_d]$ as $N \times (d-1)$ feature matrix[36]
   (d) Apply K-means to this representation

The usefulness of the Laplacian comes from the idea of minimizing cut costs while relaxing some constraints. Originally, goal would be to find value binary values $x = \pm 1$ such that each point, depending on the cluster to which the point is assinged equals 1 or -1. We do this to optimize the edge weight objective function given by $\sum_{i,j} w_{ij} (x_i - x_j)^2$. We relax this, and allow for real-valued x, while requiring $\sum_i x_i = 0$ (balanced partition) and $x^T x = 1$ (normalisation).

[35] We want to find cuts such that the cut cost is minimized. I.e., weights of edges between two points at boundary should be small. Mincuts with only one node on each side of the cut can often give bad cuts, multi-way cuts are NP-hard. Instead we relax the problem.

[36] Note that vector of one's is always an eigenvector due to structure of $\mathbf{L}$. We are interested in the eigenvectors corresponding to the smallest eigenvalues.

We have

$$\begin{aligned} x^\top \mathbf{L} x &= x^\top (\mathbf{D} - \mathbf{W}) x \\ &= \sum_i \sum_j w_{ij} \left(\frac{1}{2} x_i^2 + \frac{1}{2} x_j^2 \right) - \sum_i \sum_j w_{ij} x_i x_j \\ &= \sum_i \sum_j w_{ij} \left(x_i - x_j \right)^2 / 2 \end{aligned}$$

Thus, minimizing $x^T \mathbf{L} x$ is closely related to minimizing cut cost. Since $x^T x = 1$, we have

$$\min x^T \mathbf{L} x = \min \frac{x^T \mathbf{L} x}{x^T x}$$

This is the *Rayleigh quotient* which is minimized when choosing x to be an eigenvector of $\mathbf{L}$.

8 PCA

Real-life data may be redundant (e.g., correlation among features). Goal **Dimensionality Reduction**: Obtain a lower dimensional representation of data while minimizing information loss.

PCA in a nutshell:

1. Standardize input data $\mathbf{X} \in \mathbb{R}^{N \times D}$
2. Compute covariance matrix $\mathbf{X}^T \mathbf{X}$ to identify correlations
3. Compute eigenvectors of $\mathbf{X}^T \mathbf{X}$ to identify principal components. k-th PC is eigenvector corresponding to k-th largest eigenvalue

The idea is that eigenvectors represent the directions of the data that explain the largest variance in the data and thus also minimize information loss. Two equivalent views:

1. Maximum variance: find k directions to maximize variance in data
2. Best reconstruction: find k dimensional subspace with smallest reconstruction error

Maximum variance view

- Given: Dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$ with N data points of D dimensions as row vectors
- Goal: Find unit vector $\mathbf{v}_1 \in \mathbb{R}^D$ that maximises the variance of the dataset of the projections $\mathbf{x}_i \cdot \mathbf{v}_1$ of the data points $\mathbf{x}_i$ onto $\mathbf{v}_1$
- Solution:

$$\begin{aligned} \mathbf{v}_1 &= \underset{\mathbf{v}_1 : \|\mathbf{v}_1\|=1}{\operatorname{argmax}} \left(\sum_{i \in [N]} (\mathbf{x}_i \cdot \mathbf{v}_1)^2 \right) = \underset{\mathbf{v}_1 : \|\mathbf{v}_1\|=1}{\operatorname{argmax}} \left(\|\mathbf{X}\mathbf{v}_1\|^2 \right) \\ &= \underset{\mathbf{v}_1 : \|\mathbf{v}_1\|=1}{\operatorname{argmax}} \left(\mathbf{v}_1^\top \mathbf{X}^\top \mathbf{X}_1 \mathbf{v}_1 \right) \end{aligned} \tag{3}$$

- Next find $\mathbf{v}_2, \mathbf{v}_3, \ldots, \mathbf{v}_k$ that are all successively orthogonal to previous directions and maximise (as yet unexplained) variance[37]

$$\widehat{\mathbf{X}}_j = \mathbf{X} - \sum_{s=1}^{j-1} \mathbf{X}\mathbf{v}_s\mathbf{v}_s^\top, \quad 1 \leq j \leq k$$

Note that the solution for (3) again follows from Rayleigh's quotient. The optimal vector is the eigenvector corresponding to the largest eigenvalue of $\mathbf{X}^\top\mathbf{X}$.

Reconstruction view

- Given: Dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$ with N rows vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$
- Goal: Find a k-dimensional linear projection that best represents the data Solution in case of one PC:
  1. Let $\mathbf{v}_1$ be the direction of projection
  2. The point $\mathbf{x}_i$ is mapped to $\tilde{\mathbf{x}}_i = (\mathbf{v}_1 \cdot \mathbf{x}_i)\,\mathbf{v}_1$, where $\|\mathbf{v}_1\| = 1$[38]
  3. Minimise reconstruction error $\sum_{i=1}^N \|\mathbf{x}_i - \tilde{\mathbf{x}}_i\|^2$
- Solution in case of k PCs:

[37] Note that

$$\begin{aligned} \widehat{\mathbf{X}}_j\mathbf{v}_j &= \mathbf{X}\mathbf{v}_j - \mathbf{X}\mathbf{v}_1\mathbf{v}_1^\top\mathbf{v}_j - \cdots - \mathbf{X}\mathbf{v}_{j-1}\mathbf{v}_{j-1}^\top\mathbf{v}_j \\ &= \mathbf{X}\mathbf{v}_j \text{ (due to orthogonality)} \end{aligned}$$

so we obtain $\mathbf{v}_j = \underset{\mathbf{v}_j : \|\mathbf{v}_j\|=1, \mathbf{v}_j \perp \mathbf{v}_1, \ldots, \mathbf{v}_j \perp \mathbf{v}_{j-1}}{\operatorname{argmax}} \left(\mathbf{v}_j^\top \mathbf{X}^\top \mathbf{X}\mathbf{v}_j \right)$.

[38] Recall: The vector projection of a vector $\mathbf{v}$ on a nonzero vector $\mathbf{u}$ is defined as $\operatorname{proj}_{\mathbf{u}}(\mathbf{v}) = \frac{\langle \mathbf{v}, \mathbf{u} \rangle}{\langle \mathbf{u}, \mathbf{u} \rangle}\mathbf{u}$. Thus, $\tilde{\mathbf{x}}_i$ is the projection of $\mathbf{x}_i$ onto the vector $\mathbf{v}_1$.

- Suppose $\mathbf{V}_k \in \mathbb{R}^{D \times k}$ is such that columns of $\mathbf{V}_k$ are orthogonal
- Project data $\mathbf{X}$ onto subspace defined by $\mathbf{V}_k : \mathbf{Z} = \mathbf{X}\mathbf{V}_k$ are coordinates of our data in the lower-dimensional space
- Mapping back to the original space, $\tilde{\mathbf{X}} = \mathbf{X}\mathbf{V}_k\mathbf{V}_k^\top$[39]
- Minimise reconstruction error $\sum_{i=1}^N \left\| \mathbf{x}_i - \mathbf{V}_k\mathbf{V}_k^\top \mathbf{x}_i \right\|^2$

Equivalence

- Maximum Variance: Find $\mathbf{v}_1$ that maximises $\sum_{i=1}^N (\mathbf{v}_1 \cdot \mathbf{x}_i)^2$
- Best Reconstruction: Find $\mathbf{v}_1$ that minimises:

$$\begin{aligned}
\sum_{i=1}^N \|\mathbf{x}_i - \tilde{\mathbf{x}}_i\|^2 &= \sum_{i=1}^N \left(\|\mathbf{x}_i\|^2 - 2(\mathbf{x}_i \cdot \tilde{\mathbf{x}}_i) + \|\tilde{\mathbf{x}}_i\|^2 \right) \\
&= \sum_{i=1}^N \left(\|\mathbf{x}_i\|^2 - 2(\mathbf{v}_1 \cdot \mathbf{x}_i)^2 + (\mathbf{v}_1 \cdot \mathbf{x}_i)^2 \|\mathbf{v}_1\|^2 \right) \\
&= \sum_{i=1}^N \|\mathbf{x}_i\|^2 - \sum_{i=1}^N (\mathbf{v}_1 \cdot \mathbf{x}_i)^2
\end{aligned}$$

PCA and SVD

The maximum variance view shows us that we can obtain our principal components from the eigenvectors of $\mathbf{X}^\top\mathbf{X}$. From the reconstruction point, we have a useful theoretical result which we can use:

Theorem 4 (Eckart-Young)**.** *Let $\mathbf{X}$ be a real $N \times D$ matrix and denote by $\mathbf{X}_k$ the singular value decomposition of $\mathbf{X}$ truncated at rank k. Then for any $k \in \mathcal{N}$ and any $N \times D$ matrix $\mathbf{Y}$ of rank at most k, we have $\|\mathbf{X} - \mathbf{X}_k\|_F \leq \|\mathbf{X} - \mathbf{Y}\|_F$.*[40]

I.e., we can find our optimal reconstruction by considering the truncated singular value decomposition of $\mathbf{X}$.

[39] Note that this follows from the usual form of the projection matrix since $\mathbf{V}_k$ is orthonormal. I.e., $\mathbf{P}_\mathbf{X} = \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top$ which reduces to $\mathbf{X}\mathbf{X}^\top$.

[40] The Frobenius norm of the matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$, denoted $\|\mathbf{X}\|_F$, is defined by

$$\|\mathbf{X}\|_F = \sqrt{\sum_{i=1}^N \sum_{j=1}^D x_{ij}^2} = \sqrt{\operatorname{tr}(\mathbf{X}^\top\mathbf{X})}$$

Since $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, we obtain

$$\mathbf{X}^\top\mathbf{X} = \left(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top\right)^\top \left(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top\right) = \mathbf{V}\mathbf{\Sigma}^\top\mathbf{\Sigma}\mathbf{V}^\top$$

Eivdently, this is the eigendecomposition of the psd matrix $\mathbf{X}^\top\mathbf{X}$. Thus,

- the eigenvectors of $\mathbf{X}^\top\mathbf{X}$ are the right singular vectors $\mathbf{v}_i$ of $\mathbf{X}$
- the eigenvalues of $\mathbf{X}^\top\mathbf{X}$ are the squares of the singular values σ_i of $\mathbf{X}$
- Similarly, the eigenvectors $\mathbf{u}_j$ of $\mathbf{X}\mathbf{X}^\top$ with eigenvalue σ^2 are the left singular vectors of $\mathbf{X}$

Similar to above, we want to have $\mathbf{Z} = \mathbf{X}\mathbf{V}$. Plugging in the SVD and truncating, as we only use k eigenvectors corresponding to largest eigenvalues, we get:[41]

$$\begin{aligned}\mathbf{Z} &= \mathbf{X}\mathbf{V} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top\mathbf{V} = \mathbf{U}\mathbf{\Sigma} \\ \mathbf{Z}_k &= \mathbf{U}_k\mathbf{\Sigma}_k = \mathbf{X}\mathbf{V}_k\end{aligned}$$

Note that this also shows that principal components are a linear combination of the original features. (Further, this is an orthonormal linear combination.)

Our reconstruction is then $\tilde{\mathbf{X}} = \mathbf{Z}_k\mathbf{V}_k^\top = \mathbf{X}\mathbf{V}_k\mathbf{V}_k^\top = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^\top$ which is precisely the rank-k truncated SVD of $\mathbf{X}$.

- Eckart-Young Theorem shows that minimum reconstruction error is obtained by using the truncated SVD $\tilde{\mathbf{X}} = \mathbf{Z}_k\mathbf{V}_k^T = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^\top$
- Comparison to eigendecomposition: $\mathbf{V}_k$ contains the k eigenvectors of $\mathbf{X}^\top\mathbf{X}$ corresponding to largest eigenvalues
- Columns of $\mathbf{V}_k$ are the principal components of $\mathbf{X}$
- Columns of $\mathbf{Z}_k$ are the coefficients used to approximate each data point as a linear combination of the principal components

Algorithms for PCA Constructions

Suppose $N \gg D$: Either,

1. Variant 1: Construct $\mathbf{X}^\top\mathbf{X}$ in $O\left(D^2N\right)$ and its eigenvectors in $O\left(D^3\right)$
2. Variant 2: Iterative methods to get top k singular (right) vectors directly:

[41] $\mathbf{U}_k$ is $N \times k$, $\mathbf{\Sigma}_k$ is $k \times k$, $\mathbf{V}_k^\top$ is $k \times D$ so we end up with $\mathbf{Z}_k\mathbf{V}_k^\top$ again being $N \times D$.

(a) Initiate $\mathbf{v}^0$ to be random unit norm vector

(b) Iterative Update:

(c) $\mathbf{v}^{t+1} = \mathbf{X}^\top \mathbf{X}\mathbf{v}^t$

(d) $\mathbf{v}^{t+1} = \mathbf{v}^{t+1} / \left\|\mathbf{v}^{t+1}\right\|$ until (approximate) convergence

(e) Update step takes $O(ND)$ time[42]

(f) This gives the singular vector corresponding to the largest singular value. Subsequent singular vectors obtained by choosing $\mathbf{v}^0$ orthogonal to previously identified singular vectors

$D \gg N$:

- Construct $\mathbf{X}\mathbf{X}^\top$ in $O\left(N^2 D\right)$; eigenvectors of $\mathbf{X}\mathbf{X}^\top$ in $O\left(N^3\right)$
- The eigenvectors give the left singular vectors, $\mathbf{u}_i$ of $\mathbf{X}$
- To obtain $\mathbf{v}_i$, we use $\mathbf{v}_i = \sigma^{-1}\mathbf{X}^\top \mathbf{u}_i$

Reconstruction Error

Suppose we have $\mathbf{Z} = \mathbf{X}\mathbf{V}_k = \mathbf{U}_k\mathbf{\Sigma}_k$, we reconstruct $\tilde{\mathbf{X}} = \mathbf{Z}\mathbf{V}_k^T$. The reconstruction error is:[43]

$$\sum_{i=1}^{N} \left\|\mathbf{x}_i - \mathbf{V}_k\mathbf{V}_k^\top \mathbf{x}_i\right\|^2 = \sum_{j=k+1}^{D} \sigma_j^2$$

We may identify the number of PCs to keep by identifiyng a 'kink' in plotting k vs reconstruction error, or by using a threshold t such that $\frac{\|\mathbf{X}-\mathbf{X}_k\|_F^2}{\|\mathbf{X}\|_F^2} = \frac{\sigma_{k+1}^2+\cdots+\sigma_D^2}{\sigma_1^2+\cdots+\sigma_D^2} \leq t$.

[42] Compute $\mathbf{X}\mathbf{v}^t$ first, then $\mathbf{X}^\top\left(\mathbf{X}^t\right)$)

[43] This follows from

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \sum_{j=1}^{D} \sigma_j \mathbf{u}_j \mathbf{v}_j^\top;\ \tilde{\mathbf{X}} = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^\top = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^\top;\ \|\mathbf{X}-\tilde{\mathbf{X}}\|_F^2 = \sum_{j=k+1}^{D} \sigma_j^2$$

A Mathematical Appendix

Calculation and Derivative Rules for Matrices

$$\begin{aligned}
\mathbf{A}(\mathbf{B}+\mathbf{C}) &= \mathbf{AB}+\mathbf{AC} \\
(\mathbf{AB})^T &= \mathbf{B}^T\mathbf{A}^T \\
(\mathbf{AB})^{-1} &= \mathbf{B}^{-1}\mathbf{A}^{-1} \\
&\text{Derivatives:} \\
\nabla_{\mathbf{x}}\left(\mathbf{c}^\top\mathbf{x}\right) &= \mathbf{c} \\
\nabla_{\mathbf{x}}\left(\mathbf{x}^\top\mathbf{x}\right) &= 2\mathbf{x} \\
\nabla_{\mathbf{x}}(\mathbf{Ax}) &= \mathbf{A}^\top \\
\nabla_{\mathbf{x}}\left(\mathbf{x}^\top\mathbf{Ax}\right) &= \mathbf{Ax}+\mathbf{A}^\top\mathbf{x}
\end{aligned}$$

Determinant calculation

Let $\mathbf{A}$ be an $n \times n$ matrix for some $n \geq 1$. If $\mathbf{A}$ consists of a single element, then the determinant $\det(\mathbf{A})$ of $\mathbf{A}$ is the element itself. Otherwise:

$$\det(\mathbf{A}) = \begin{vmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{vmatrix} = \sum_{j=1}^{n} (-1)^{i+j} a_{i,j} M_{i,j},$$

where $i \in [n]$ is arbitrary and $M_{i,j}$ (called minor) is the determinant of $\mathbf{A}$ after removing the i-th row and j-th column.

For a 2×2 matrix:

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc$$

Inverse calculation

For a 2×2 matrix A, we have

$$A^{-1} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{\det A} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

For a 3x3 matrix A, its inverse A^{-1} can be calculated using the formula:

$$A^{-1} = \frac{1}{\det(A)} \cdot \text{adj}(A)$$

where $\det(A)$ is the determinant and $\text{adj}(A)$ is the adjugate matrix.

1. For each element a_{ij}, calculate its cofactor C_{ij} using:
$$C_{ij} = (-1)^{i+j} \cdot M_{ij}$$
where M_{ij} is the minor (determinant of the 2x2 matrix formed by deleting row i and column j).

2. The adjugate matrix is the transpose of the cofactor matrix:
$$\text{adj}(A) = C^T$$

3. Multiply the adjugate matrix by the reciprocal of the determinant:
$$A^{-1} = \frac{1}{\det(A)} \begin{pmatrix} C_{11} & C_{21} & C_{31} \\ C_{12} & C_{22} & C_{32} \\ C_{13} & C_{23} & C_{33} \end{pmatrix}$$

B Complexity of matrix computations

- Matrix Inversion ($N \times N$ matrix):
  - Complexity: $O(N^3)$
  - Explanation: N steps of elimination, each with $O(N^2)$ operations
- Matrix-Matrix Multiplication: $N \times M$ matrix times $M \times P$ matrix:
  - Complexity: $O(N \times M \times P)$
  - Explanation: M operations for each element of $N \times P$ result
- Matrix-Vector Multiplication: $N \times M$ matrix times $M \times 1$ vector:
  - Complexity: $O(N \times M)$
  - Explanation: M operations for each of N output elements
- Vector-Vector Multiplication:
  - Inner product ($1 \times N$ vector times $N \times 1$ vector):
    - Complexity: $O(N)$
    - Explanation: N multiplications, $N - 1$ additions
  - Outer product ($N \times 1$ vector times $1 \times M$ vector):
    - Complexity: $O(N \times M)$
    - Explanation: One operation per matrix element