Auxiliary notes on Statistical Machine Learning

Daniel Lin; Professor: François Caron

March 4, 2024

This note seeks to offer compelling motivations for the concepts covered in the SB2.2 Statistical Machine Learning course while refining the clarity of mathematical derivations. Please use it in line with the lecture notes.

Contents

1 Symbols

Symbols used in this note have the following default meaning unless stated otherwise.

Symbol	Meaning
Σ	covariance matrix of a random vector
$\mathrm{TV}(X) = \mathrm{trace}(\Sigma)$	total variance of the random vector X
n	number of inputs
p	dimension of input (number of features)
x_i	the ith data point (p-dimensional vector), $x_i \in \mathcal{X}$ (input space)
x_{ij}	the jth feature of ith data point
y_i	label of the ith data point, $y_i \in \mathcal{Y}$(target space)
$\mathcal{D}$	the data set $\{x_i\}_{i=1,\cdots,n}$(unsupervised), $\{y_i\}_{i=1,\cdots,n}$(supervised)
X	the input matrix, with dimensions $n \times p$
P_0	distribution of underlying population
K	dimension of code/latent layer
enc_θ	Encoder with parameter θ
dec_θ	decoder with parameter θ
h_θ	autoencoder with parameter θ
L	loss function
$\widehat{R}$	empirical risk function
h	prediction rule (in supervised learning)
h^*	Bayes prediction rule
$h^{(\mathcal{D})}$	prediction rule estimated by dataset $\mathcal{D}$
θ	parameters (could be a vector)
$\widehat{\theta}_{\mathrm{MLE}}$	maximum likelihood estimator of θ
$\nabla_\theta J(\theta)$	gradient of a multivariate function $J(\theta)$ at the point θ
$\nabla^2_\theta J(\theta)$	Hessian matrix(second derivative) of a multivariate function $J(\theta)$ at the point θ

2 Unsupervised learning

Consider a dataset comprising n individual data points, each denoted as $x_i = (x_{i1}, \cdots, x_{ip}) \in \mathbb{R}^p$. These data points can be organised into a matrix $X = (x_{ij}) \in \mathbb{R}^{n\times p}$. The matrix X may possess complexity, be challenging to interpret, and could be tainted with noise. While assigning labels to all data points transforms the task into a prediction problem, the process of labelling can be expensive, especially when dealing with a vast database of, for instance, millions of images.

In scenarios where labels are not readily available, the focus shifts to *unsupervised learning*. As statisticians, we often resort to constructing parametric models to extract valuable features. In machine learning, the model is called *encoder*. In the realm of image feature extraction, the encoder transforms an image into a concise representation, akin to summarising its features into a few key phrases. In general, an encoder is characterised by a function

$$\mathrm{enc}\theta \,:\, \mathbb{R}^p \to \mathbb{R}^K$$

where K represents the dimension of the latent representation of the data, denoted as $z = (z_i)$, with $z_i := \mathrm{enc}\theta(x_i)$. The term "latent" is employed because this representation is not directly observed.

Conversely, a *decoder* aims to reconstruct something resembling the original data from the latent representation z. A notable example is the image generator in a Generative Adversarial Network (GAN)Kingma and Welling (2019). Decoders are also encapsulated by mathematical functions, denoted as $\mathrm{dec}_\theta : \mathbb{R}^K \to \mathbb{R}^p$. The encoder (recognition model) and decoder (generative model) collaborate, delving into the underlying structure of the data matrix X (see Figure 1).

The latent representation, often referred to as the *bottleneck* layer, serves a crucial role by preventing a direct one-to-one relationship between the input and output layers. In situations where such a direct relation exists, the model may inadvertently learn the noise present in the input data. The intentional reduction in dimensionality at the bottleneck layer compels the model to focus on learning the underlying structure of the data rather than

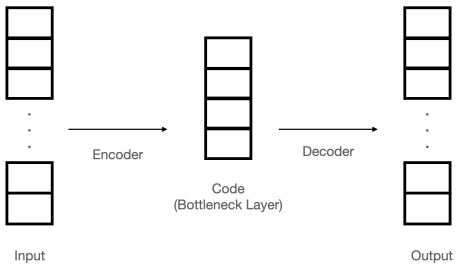


Figure 1: Encoder-decoder architecture

capturing stochastic noisesKramer (1992).

The effectiveness of the encoder-decoder architecture in discovering the correct structure can be assessed through a validation process. This involves encoding the data, decoding it, and then comparing the properties of the reconstructed sample, denoted as $\{\widehat{x}_i\}_{i=1,\cdots,n}$, with the original data.

$$\widehat{x}_i := \mathrm{dec}_\theta(\mathrm{enc}_\theta(x_i))$$

the composite function $\mathrm{dec}_\theta \circ \mathrm{enc}_\theta$ is also called an *auto-encoder*, denoted as h_θ.

The loss function $L : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}^+$ serves to compare the reconstructed sample with the original sample directly. A high value is assigned by $L(x_i, \widehat{x}_i)$ when the two samples exhibit significant dissimilarity. The square loss is the most commonly employed loss function.

$$L(x_i, h_\theta(x_i)) = L(x_i, \widehat{x}_i) := \|x_i - \hat{x}_i\|^2$$

To measure the quality of autoencoder h_θ, the average loss, or the *risk* is defined

$$R(h_\theta) := E_{X\sim p_0}\left[L(X, h_\theta(X))\right]$$

where p_0 is the unknown true population distribution. In practical scenarios, under the assumption that samples x_i are identical and independently distributed (i.i.d.), the risk can be estimated through the sample mean of the loss. This is possible since the distribution of X is already encapsulated by the sample.

Definition 2.1 — **Empirical Risk.** Given a loss function L, the empirical risk is

$$\widehat{R}(h_\theta) := \frac{1}{n}\sum_{i=1}^{n} L(x_i, h_\theta(x_i))$$

Remark 2.2 Here, we are essentially estimating the true distribution p_0 via sample $\mathcal{D} = \{x_i\}_{i=1,\cdots,n}$,

$$\widehat{p}_0(x \,|\, \mathcal{D}) = \frac{1}{n}\sum_{i=1}^{n} \delta(x - x_i)$$

where $\delta(x - x_i) = 1$ if $x = x_i$ and 0 otherwise. Plugging this into the risk yields

$$\begin{aligned}
\widehat{R}(h_\theta) &= \int_{\mathcal{X}} L(x, h_\theta(x))\widehat{p}_0(x \,|\, \mathcal{D})\, dx \\
&= \frac{1}{n}\sum_{i=1}^{n} \int_{\mathcal{X}} \delta(x - x_i) L(x, h_\theta(x))\, dx \\
&= \frac{1}{n}\sum_{i=1}^{n} L(x_i, h_\theta(x_i))
\end{aligned}$$

In dealing with empirical risk, the objective is to identify the parameter θ that minimizes the empirical risk.

$$\theta^* := \arg\min_{\theta} \hat{R}(h_\theta)$$

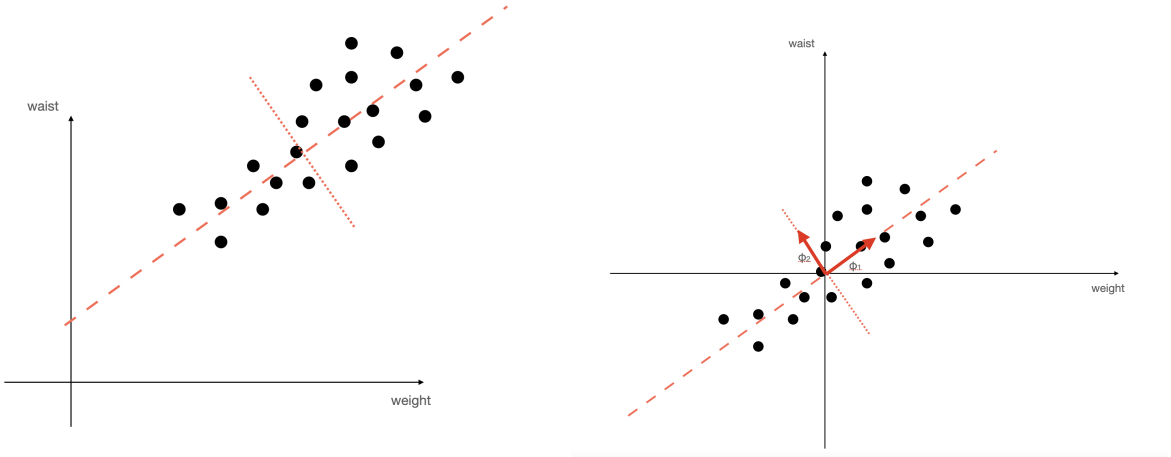
This process is called *empirical risk minimisation*.

The next two chapters will introduce two unsupervised learning algorithms and their corresponding encoder-decoder architectures.

3 Principal Component Analysis

Suppose you have gathered data on waist circumference and weight of patients, and it becomes apparent that one of these variables is redundant due to a correlation – patients with wider waists generally exhibit higher weights. Rather than discarding one of these variables, there exists a better approach.

In Figure 2a, arbitrary data points representing patients are depicted. Notably, the data exhibits the most variation along the dashed line, while the least variation occurs along the dotted line. For ease of analysis, the data is centralized, and the two lines are drawn across the origin (Figure 2b). Here, two vectors, usually assumed to be unit vectors, denoted as v_1 and v_2, suffice to describe these lines. v_1 and v_2 are termed the *principal components*. If all data points are projected onto the dashed line (aligned with v_1), a new dataset with reduced dimensionality by 1 is obtained. This process effectively encodes the information from the two strongly correlated variables into a single dimension, providing a concise summary of the common information contained in waist circumference and weight.



(a) Original version **(b) Original version**

Figure 2: Weights and waist of patients

In Principal Component Analysis (PCA), the objective is to identify directions with the largest variations, as well as those with the least variations (which may be discarded). These identified directions are termed *principal components*.

3.1 An empirical approach

Now we investigate how the principal components can be found mathematically. Suppose data $x_i \in \mathbb{R}^p : i = 1, \cdots, N$ are projected onto the line spanned by a given unit vector v (let's denote this line as axis l). The projected points have a value $z_i := x_i^T v$ on axis l (i.e., the distance to the origin is $|z_i|$). Since the data is already centralised, the sample variance of the projected data is simply given by

$$\frac{1}{n}\sum_{i=1}^{n} z_i^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i^T v)^2 = \frac{1}{n}\sum_{i=1}^{n} v^T x_i x_i^T v = v^T \left(\frac{1}{n}\sum_{i=1}^{n} x_i x_i^T\right) v$$

finding v that maximises this variance ought to find the Eigenvectors of the matrix

$$\sum_{i=1}^{n} x_i x_i^T = X^T X$$

with the largest Eigenvalues.

Proposition 3.1 For real, symmetric matrix A, the optimisation problem

$$\max_v v^T A v \quad \text{and } v^T v = 1$$

is solved by the Eigenvector with the largest Eigenvalue. Further, if the problem is minimisation instead, it is solved by the Eigenvector with the smallest Eigenvalue.

Proof. A real symmetric matrix A admits a spectral decomposition

$$U^T A U = D$$

where U is orthogonal, and $D = \text{diag}(\lambda_i)$ is a diagonal matrix with Eigenvalues on the diagonal entries. (a proof can be found in linear algebra textbooks)

Define x by changing of variables $x = U^T v$ (by orthogonality, $\|x\| = 1 \Leftrightarrow \|v\| = 1$), under this transformation, assuming $v^T v = 1$ is satisfied,

$$v^T A v = x^T D x = \sum \lambda_i x_i^2 \leq \lambda_{\max} \sum x_i^2 = \lambda_{\max}$$

the largest Eigenvalue $\lambda_{\max}$ is an upper bound of $v^T A v$. And if $v = v_{\max}$ (the unit Eigenvector corresponding to $\lambda_{\max}$),

$$v_{\max}^T A v_{\max} = v_{\max}^T (\lambda_{\max} v_{\max}) = \lambda_{\max}$$

so the upper bound is attained. Indeed $v_{\max}$ solves the maximisation problem.

The minimisation case can be proved similarly. □

Remark 3.2 Spectral decomposition means that all real symmetric matrices represent scaling (encoded in D), rotations and reflections (encoded in orthogonal U). From the proof, we also see that Eigenvector v with a larger Eigenvalue yields a larger value of $v^T A v$.

The sample covariance of X is $S = X^T X/(n-1)$, with the same set of Eigenvectors with $X^T X$. Suppose we have found the (orthonormal) Eigenvectors $v_1, \cdots, v_p$ of S with Eigenvalues $\lambda_1, \cdots, \lambda_p$. For each v_i, the size of corresponding Eigenvalue λ_i quantifies the variance, because

$$\text{sample variance of } Z = v_i^T \left(\frac{1}{n} \sum_{i=1}^n x_i x_i^T \right) v_i = \frac{n-1}{n} v_i^T S v_i = \frac{n-1}{n} v^T (\lambda_i v_i) = \frac{n-1}{n} \lambda_i$$

WOLG, assume v_i are ordered so that $\lambda_1 > \lambda_2 > \cdots > \lambda_p$. We define the (sample) principal components to be $\{v_i\}_{1,\cdots,p}$.

Now, one has to decide how many principal components to keep (i.e., the dimension of the latent representation z, denoted K). In Figure 3, the Eigenvalues are sorted and plotted. In Figure 3(a), this decision is straightforward due to the noticeable jump in Eigenvalue. However, in reality, Eigenvalues may not exhibit such a distinct jump (Figure 3(b)), and the choice of K may impact the result.

Suppose $K < p$ is already chosen, then the latent representation z_i is the x_i projected onto $v_1, \cdots, v_K$.

$$z_i := \begin{pmatrix} x_i^T v_1 \\ x_i^T v_2 \\ \vdots \\ x_i^T v_K \end{pmatrix} = V^T x_i$$

where $V = V_{1:K} \in \mathbb{R}^{p \times K}$ is a matrix with columns being $v_1, \cdots, v_K$. The full data matrix is transformed by $Z = XV$.

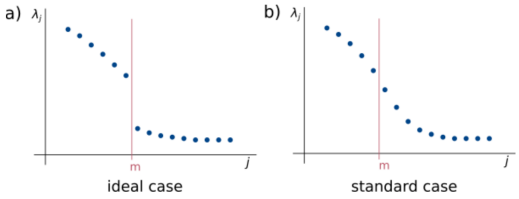


Figure 3: Eigenvalues of sample covariance matrix S, x-axis is simply index, y-axis represents the Eigenvalues.

The total sample variance is

$$\text{trace}(S) = \text{trace}(UDU^T) = \text{trace}(D) = \sum_{k=1}^{p} \lambda_k$$

where UDU^T is the spectral decomposition of the symmetric matrix S, with diagonal entries of D being the Eigenvalues and orthogonal U consisting of the orthonormal Eigenvectors. Note that Trace is not altered by multiplying an orthogonal matrix.
On the other hand, the variance of $\{z_{ij}\}_{i=1,\cdots,n}$ (jth projection) is λ_j as illustrated before. So each principal component explains the $\lambda_j / \sum_{k=1}^{p} \lambda_k$ proportion of variance, and the whole vector z_i explains

$$\frac{\sum_{k=1}^{K} \lambda_k}{\sum_{j=1}^{p} \lambda_j}$$

of the variance.

The Encoder-Decoder Architecture
In this case, the encoder is $x \mapsto V^T x$, and conversely, the decoder is $z \mapsto Vz$. Due to the orthonormal column vectors of the matrix V, it is orthogonal, and the inverse transformation is precisely the transpose. Finally, the auto-encoder reconstructs the sample through $\widehat{x}_i = VV^T x_i$. If all the p principal components are chosen, V is a $p \times p$ matrix and by orthogonality, $V^T V = VV^T = I_p$. So $\widehat{x}_i = x_i$, we have not reduced any dimensionality.

It can be shown that $V = V_{1:K}$ consisting of the K Eigenvectors does minimise the empirical risk defined from square loss,

$$\widehat{R}_n(h_V) = \frac{1}{n} \|x_i - VV^T x_i\|^2$$

(here, the dimension of code/latent space is fixed to K)

3.2 Theoretical justification

In the previous section, we defined principal components as the Eigenvectors of the sample covariance matrix S. However, this dependence on the specific sample obtained raises a question: if we define the (population) principal components as the Eigenvectors of the covariance matrix Σ of the underlying distribution of X, do they still capture the directions with the largest variation? Also, we wish to have orthogonality between principal components, otherwise, there will be redundancy in the variances explained.

Finding the first PC
We aim to identify a unit vector v_1^* such that $Z_1 := v_1^* X$ exhibits maximal variance. Here, the asterisk distinguishes the population PC v_1^* from the sample PC v_1. The variance of Z_1 is given by $\text{Var}(Z_1) = (v_1^*)^T \Sigma v_1^*$. Hence, the optimization problem can be stated as:

$$\max_{v} v^T \Sigma v \quad \text{subject to } v^T v = 1$$

which takes the same form as in the last section. You can apply the same approach as before, taking advantage of the fact that Σ is also real symmetric, though unknown. Alternatively, the Lagrangian method can be employed

to solve this quadratic optimization problem with the equality constraint.

Subsequent PCs
Assuming the first $k-1$ orthogonal principal components are denoted as $v_1^*, \cdots, v_{k-1}^*$ (which are Eigenvectors of Σ), the objective is to find v_k^* such that the variance of $Z_k := v_k^* X$ is maximised while ensuring orthogonality. In other words, v_k^* should capture as many variances not already explained by the first $k-1$ PCs as possible. Therefore, the optimization problem is:

$$\max_v v^T \Sigma v \quad \text{subject to } v^T v = 1,\ v^T v_j^* = 0 \text{ for } j = 1, \cdots, k-1$$

It can be proven that the solution must be an Eigenvector of Σ using the Lagrangian method. Enforcing orthogonality with the first $k-1$ Eigenvectors, the solution corresponds to the Eigenvector with the kth largest Eigenvalue.

3.3 SVD approach

From numerical analysis results, spectral decomposition, or the Eigendecomposition, is computationally expensive. In contrast, the Singular Value Decomposition (SVD) can be computed faster and works for non-square matrices. The SVD of the data matrix X is

$$X = UDV^T$$

where $U \in \mathbb{R}^{n \times n}, V \in \mathbb{R}^{p \times p}$ are orthogonal, and $D \in \mathbb{R}^{n \times p}$ is a (non-square) diagonal matrix where the entries are called singular values σ_i (generally, assume $\sigma_1 \geq \sigma_2 \geq \cdots$). This decomposition is akin to Eigendecomposition. In fact, the columns u_i, v_i of U, V are referred to as *left and right Eigenvectors*, respectively. Singular values possess a meaningful interpretation: The count of non-zero singular values is the rank of X. And if X has full rank p (indicating independent samples), then the optimal approximation of X with rank $m < p$ is given by

$$X_m := \sum_{i=1}^{m} \sigma_i u_i v_i^T$$

Using this decomposition, we can write down that

$$(n-1)S = X^T X = (UDV^T)^T UDV^T = VD^T U^T UDV^T = V(D^T D)V^T$$

which takes the form of Eigendecomposition. The Eigenvectors of S are essentially v_i, with Eigenvalues being $\sigma_i^2/(n-1)$. Therefore, the v_i are already ordered according to the size of Eigenvalues. (because singular values σ_i are in non-increasing order) The projected latent data is

$$Z = XV = UDV^T V = UD$$

so all the information we need for PCA is in the three matrices U, D and V.

When $p > n$, $X^T X \in \mathbb{R}^{p \times p}$ is very large, the Eigenvectors of the following matrix may be used instead

$$B := XX^T = UDV^T (UDV^T)^T = UDV^T VD^T U^T = UDD^T U^T$$

The *Gram matrix* $B \in \mathbb{R}^{n \times n}$ has the same set of Eigenvalues as S.

3.4 Implementation and Limitations

In practice, plotting the first few principal components (PCs) can reveal meaningful features of the dataset. Reduced PCA is often employed, especially when many rows of D are 0 (which is the case when $n > p$).

PCA have the following limitations

- When the noise is considerable or the variance is scattered across all dimensions (as in the standard case depicted in Figure 3), selecting the appropriate number of dimensions K can be challenging. An incorrect choice may incorporate variances corresponding to noise components to the latent variable Z.

- In situations where the variation in one dimension is inherently higher—for example if the inter-quartile range (IQR) of the i-th dimension is 100 while the IQR of other dimensions is below 10—the first principal component can be very close to e_i (the i-th axis). This issue can be addressed by standardising the data or by using the Eigenvectors of the sample correlation matrix instead.
- Outliers have the effect of increasing the variance in the direction in which they lie. Consequently, some principal components may be biased towards these directions influenced by outliers.
- The population's principal components are estimated using samples, emphasizing the importance of a sufficiently large sample size. Shaukat et al. (2016)

4 K-mean

The term *Clustering* refers to the process of categorising data points, which is one of the important unsupervised learning tasks. This can be mathematically defined by assigning cluster labels z_i to data points x_i, where z_i takes values from the set $1, \cdots, K$, and K denotes the total number of clusters. For instance, one might assign the label 1 to males and 0 to females. The clusters themselves can be represented as collections of indices; for instance, $C_i = 2, 3, 5$ signifies that the data points x_2, x_3, x_5 belong to cluster i. Of course, we require $\{C_k\}_{l=1,2,\cdots,K}$ to be a partition of all the indices $\{1, \cdots, n\}$.

K-means is one of the earliest and most widely used clustering methods. As implied by its name, this algorithm aims to identify the centroids (i.e., the mean of all points in a cluster) for a specified number, K, of clusters and utilises these centroids to assign cluster labels. The centroid of cluster k is defined as

$$\overline{x}_k := \frac{1}{|c_k|} \sum_{i \in c_k} x_i$$

4.1 K-mean as within-cluster distance minimiser

Before introducing the K-means algorithm, we will discuss two important quality measures for clusters. Suppose $\rho(x, y)$ measures the dissimilarity between x and y (in Euclidean space, it is usually defined as the Euclidean distance $\rho(x, y) := |x - y|^2$). The *within-cluster distance* measures how closely points in a cluster are located. The left side of Figure 4 shows the pairwise distance, which simply sums up the dissimilarities of all pairs. The pairwise distance of cluster C is $1/2 \sum_{i,j \in C_k} \rho(x_i, x_j)$ (it is divided by 2 due to replicates in the summation). Therefore, the within-cluster distance of the partition $C_k k=1,\cdots,K$ is

$$W_{\text{pair}}(\{C_k\}_{k=1,\cdots,K}) = \frac{1}{2} \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,j \in C_k} \rho(x_i, x_j) \tag{4.1}$$

the distance for each cluster is normalised by the cluster size because there are more pairs of points in a larger class.

The disadvantage of the measure in Equation 4.1 is evident from the figure; the number of pairs increases dramatically as the cluster size grows (e.g., C_2 in Figure 4). On the other hand, we could calculate the total distance to the centroid $\overline{x}_k$ as shown on the right side of Figure 4), which is

$$W_{\text{centroid}}(\{C_k\}_{k=1,\cdots,K}) = \sum_{k=1}^{K} \sum_{i \in C_k} \rho(x_i, \overline{x}_k) \tag{4.2}$$

Surprisingly, these two measures are mathematically equivalent, i.e. $W_{\text{centroid}}(\{C_k\}_{k=1,\cdots,K}) = W_{\text{pair}}(\{C_k\}_{k=1,\cdots,K})$. This is because

$$\frac{1}{2|C_k|} \sum_{i,j \in C_k} \rho(x_i, x_j) = \sum_{i \in C_k} \rho(x_i, \overline{x}_k)$$

So, we may want to find a "good" partition $C_k k = 1, \cdots, K$ by minimising the pairwise within clusters distance Wpair. However, this objective function involves a double summation of dissimilarities ρ, and the centroids $\overline{x}_k$ require another summation by definition. The minimisation process is not straightforward.

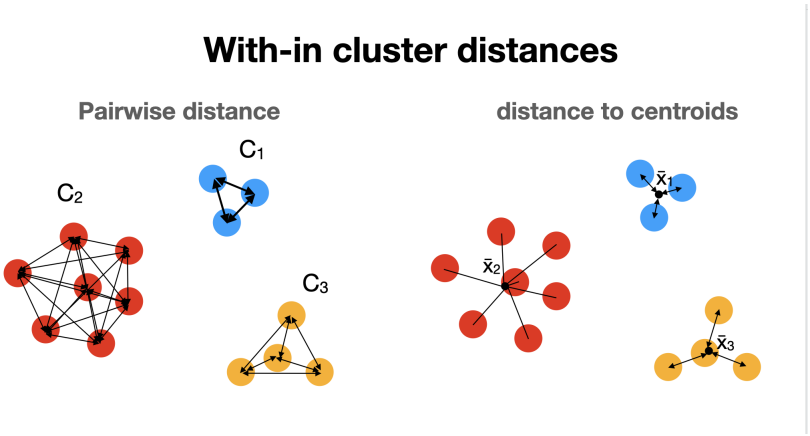


Figure 4: With-in Cluster distances

The solution is to allow the algorithm to choose the centroids, i.e., define the following objective function instead:

$$W(\{C_k\}_{k=1,\cdots,K}, \{\mu_k\}_{k=1,\cdots,K}) = \sum_{k=1}^{K} \sum_{i \in C_k} \rho(x_i, \mu_k) = \sum_{i=1}^{n} \|x_i - \mu_{z_i}\|^2 \tag{4.3}$$

where parameters μ_k represents arbitrary centroids. Then we can adapt separate minimisations over all possible partitions and all possible centroids
(1) If $\{\mu_k\}$ are known, the goal is to find cluster labels z_i that minimize W. Intuitively, assigning each point to the closest centroid μ_k minimizes the overall distances to the centroids. Indeed,

$$z_i := \arg\min_{i=1,\cdots,K} \|x_i - \mu_i\|^2$$

minimises W.
(2) If the clusters $\{C_k\}$ are known, the best centroids μ_k are simply the empirical centroids $\overline{x}_k$. (This can be checked by differentiation)

We seem to be stuck in a dilemma where the evaluation of two quantities relies on each other. But this can be resolved by choosing an initial value for either clusters $\{C_k\}$ or centroids $\{\mu_k\}$ and then minimising according to (1) and (2) in turn until convergence. This is the *coordinate descent method*. An example is illustrated in Figure 5.

- Step 0. By inspection, there are two clusters, so let $K = 2$.
- Step 1. randomly assign the cluster labels(one of green and blue)
- Step 2. find the centroids m_1, m_2 for two clusters.
- Step 3. Update the assignments of points by choosing the cluster l where its centroid m_l is closer to the point (the distances are shown as dotted lines)
- Step 4. The points are already assigned to the correct clusters. We update the centroids again

Note if we try to perform another iteration, there will be no change to the assignment of data, so we claim that the algorithm has converged. This is the *stability criterion* for convergence. In practice, milder convergence criteria include enough fraction of unchanged assignments and setting a tolerance on the centroid adjustment. Alternatively, you may use the cluster quality measures discussed in the next section.

4.2 Measuring Cluster Quality

Apart from within-cluster distance, cluster separation is also important for consideration. Two clusters should not be too close together. One notion of the between-cluster distance is the sum of distances between the centroids

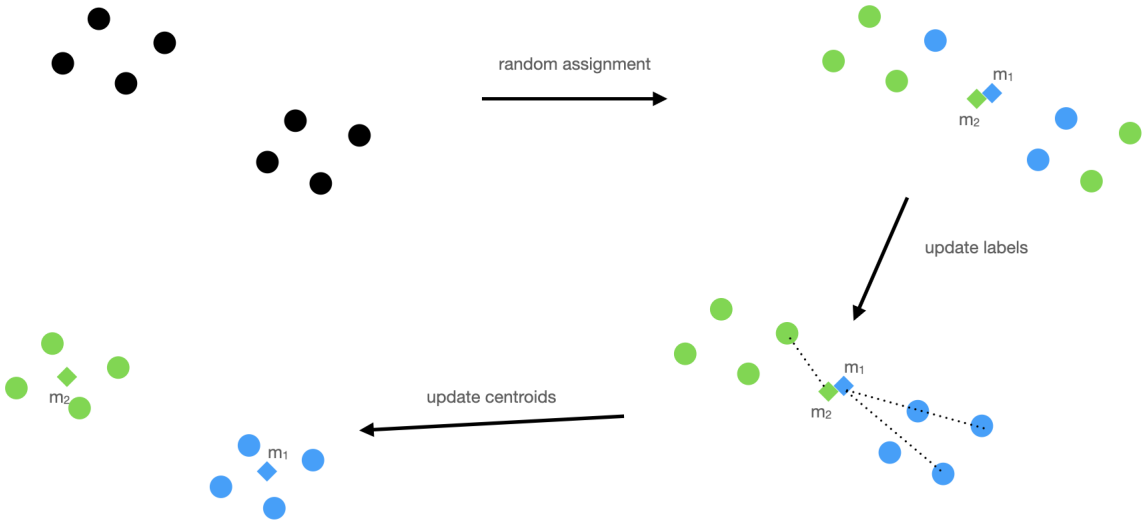


Figure 5: Simple illustration of the K-mean algorithm

and the sample mean $\overline{x}$,

$$B(\{C_k\}_{k=1,\cdots,K}) := \sum_k |C_k| \|\overline{x}_k - \overline{x}\|^2 = \sum_k |C_k| \|\mu_k - \overline{x}\|^2$$

they are weighted by the size of cluster $|C_k|$ because we are more tolerant of large clusters. This distance measure is illustrated in Figure 6.

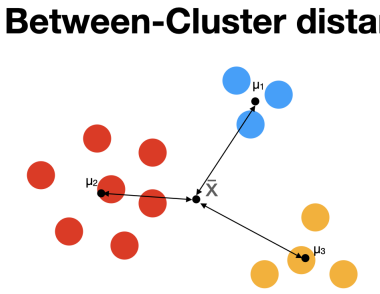


Figure 6: Between-cluster distance

Surprisingly, according to Huygens' theorem Chavent (1998), if the partition C_k minimizes $W(C_k)$, it also maximizes $B(C_k)$. This implies that the K-means algorithm simultaneously minimises the within-cluster distance and maximises the between-cluster distance. However, determining the appropriate value for K can be challenging. (In the Step 0 of the simple example, we estimated it by counting how many clusters we can observe, but this is not practical in higher dimensions) The *elbow method* is one approach, which involves running the K-means algorithm with increasing values of K until the drop in the value of the objective function W becomes slow. However, defining what qualifies as "slow" remains a subjective decision.

Quality measures

A clustering quality measure allows us to select the most suitable K. For example, the *Calinski-Harabasz score* is

defined as

$$\mathrm{CH}(\{C_k\}) = \frac{B(\{C_k\})}{W(\{C_k\})}\frac{n-K}{K-1}$$

the first term assesses whether the clusters are well-separated(large B) and compact (small W), whereas the second term penalises complicated models with too many clusters. Especially, picking $n = K$ produces no useful information. Another measure called the *silhouette score* is assigned to individual points x_i,

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

where $a(i)$ (cohesion) is the average distance from x_i to the other points in the same cluster, and $b(i)$ (separation) is the smallest average distance from x_i to data points in the closest neighbour cluster. There are loads more quality measures to explore in the Python library `sklearn.cluster`.

4.3 Implementation

When implementing the K-means algorithm, it is essential to consider several points for effective execution.

- **Initialisation**: assign initial centroids as some randomly chosen training data points.
- **Dissimilarity**: if you inspect strong correlations between features, use the Mahalanobis distance, defined by

  $$\rho(x, y) := \|x - y\|_M = \sqrt{(x-y)^T M^{-1}(x-y)}$$

  where M is any positive semi-definite matrix (e.g. sample covariance), instead of Euclidean distance.
- Be careful with the obtained results if the true cluster shapes are non-convex, or there are outliers.
- For extremely large datasets, use *stochastic K-mean* instead. At step t, randomly pick x_i, assign cluster label z_i to the nearest centroid and pull that centroid slightly to the data point x_i

  $$\mu_{z_i} += \alpha_t(x_i - \mu_{z_i})$$

  where α_t is some step size that has to be pre-defined.

4.4 Autoencoder and ERM

The encoder-decoder architecture for K-mean can be defined as below

$$\mathrm{enc}_\theta(x_i) = \underset{k=1,\cdots K}{\arg\min} \|x_i - \mu_k\|^2$$

$$\mathrm{dec}_\theta(x_i) = \mu_{z_i}$$

the parameter $\theta = (K, \{\mu_k\}_{k=1,\cdots,K})$ in this case. The latent representation is a vector of cluster labels, and the reconstructed data consists of the centroids of the clusters to which each data point belongs. This is commonly used in data compression, such as image compression (where a smaller K corresponds to coarser compression). It can be proved that the empirical risk is the partial minimisation of W (defined in Equation 4.3) over $\{C_k\}$. i.e.

$$\widehat{R}_n(h_\theta) = \min_{C_1,\cdots,C_K} \frac{1}{n} W(\{C_k\}, \{\mu_k\})$$

So the K-mean algorithm is indeed an ERM.

5 Supervised Learning

In supervised learning, the dataset $\mathcal{D} := \{(x_i, y_i)\}_{i=1,\cdots,n}$ contains labels y_i for each data point x_i. The objective is to predict the outcome y. More precisely, the goal is to find a mapping h from input x to outcome y with good

- **training performance**: performs well on the sample $\mathcal{D}$, i.e., $h(x_i) \approx y_i \ \forall i$
- **generalisation performance**: generalises to the entire population, i.e., $h(X) \approx Y$ for $(X, Y) \sim P_0$ (joint distribution of the underlying population).

The map $h : \mathcal{X} \to \mathcal{Y}$ is called a *prediction rule*. And $\mathcal{F} := \{h : \mathcal{X} \to \mathcal{Y}\}$ is the set of all prediction rules. Note it is an extremely large set.

Example 5.1 The first scenario involves classification($\mathcal{Y} = \{1, \cdots, K\}$), where the prediction rule h assigns labels to input $x \in \mathcal{X}$.

In the second scenario, regression is the goal, with $\mathcal{Y} = \mathbb{R}$. For instance, x could represent the hours of work per day, and y is the corresponding income. Here, the prediction rule h forecasts the income based on the hours of work. The set of prediction rules $\mathcal{F}$ in this case encompasses all functions $\mathbb{R} \to \mathbb{R}$, including funny functions like the Weierstrass function, which is continuous everywhere but differentiable nowhere.

As the set $\mathcal{F}$ is often too extensive, it's common to consider a hypothesis class of decision rules $\mathcal{H} \subset \mathcal{F}$, such as linear functions.

Definition 5.2 — linear prediction rule. If $\mathcal{Y} = \mathbb{R}$, the linear prediction rule is in the form of $h(x) = \beta_0 + \beta^T x$ where $\beta_0 \in \mathbb{R}$, $\beta \in \mathbb{R}^p$. If $\mathcal{H}$ is the set of linear prediction rules, the regression task becomes the familiar *linear regression*.

Certainly, it's essential to ensure that $\mathcal{H}$ captures the underlying data structure. Proposing $\mathcal{H} = h, :, h(x) = \beta_0 + \beta^T x$ is effective only when X and Y exhibit a linear relationship.

The prediction rule learnt from data set $\mathcal{D}$ is denoted $\widehat{h}^{\mathcal{D}}$, and the predicted values $\widehat{y}_i := \widehat{h}^{\mathcal{D}}(x_i)$. Note that both $\widehat{h}^{\mathcal{D}}$ and $\widehat{y}_i$ are random because they depend on the random data $\mathcal{D}$ with data points (X_i, Y_i) taken from the true population distribution.

5.1 Decision-Theory Framework

As the unsupervised learning, we need to specify a loss function before defining the notion of a "good" prediction rule.

Definition 5.3 — Loss function. The loss function is

$$L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$$

For example, the loss of the data point i under prediction rule h is $L(y_i, h(x_i))$.
The notion of risk function follows, which is the average loss over all possible data.

Definition 5.4 — Risk. For given loss L, the risk of prediction rule h is

$$R(h) := E_{X,Y \sim P_0}[L(Y, h(X))]$$

Remark 5.5 The loss function for supervised learning is defined on the target space $\mathcal{Y}$ instead of $\mathcal{X}$, and the risk is expected on the joint distribution of X, Y.

The best prediction rule in the sense of minimising the risk is called *Bayes prediction rule*.

Definition 5.6 — Bayes Prediction rule. Bayes prediction rule h^* is

$$h^* := \arg\min_{h \in \mathcal{F}} R(h)$$

Empirical Risk Minimisation The risk cannot be directly evaluated since P_0 is unknown. Therefore, akin to unsupervised learning, we resort to empirical risk minimisation to determine the optimal prediction rule.

Definition 5.7 — Empirical Risk Minimiser. For hypothesis space $\mathcal{H}$, loss function L and a dataset $\mathcal{D}$(of size n), the empirical risk minimiser is

$$\widehat{h}^{(\mathcal{D})} := \arg\min_{h \in \mathcal{H}} \widehat{R}_n(h)$$

where $\widehat{R}_n$ is the empirical risk

$$\widehat{R}_n(h) := \frac{1}{n} \sum_{i=1}^{n} L(y_i, h(x_i))$$

Remark 5.8 Once again, the justification for employing empirical risk hinges on the assumption that the sample $\mathcal{D} = (x_i, y_i)_{i=1,\cdots,n}$ is representative of the true underlying distribution P_0.

You may have noticed that the minimisation in Definition 5.7 is restricted to the proposed subspace $\mathcal{H} \subseteq \mathcal{F}$. The choice of hypothesis class matters, as we will see in the next section.

Plug-in Methods Apart from ERM, There is another way to estimate the risk. Using the tower rule of expectation,

$$R(h) = E_{X,Y}[L(Y, h(X))] = E_X\left[E_{Y|X=x}[L(Y, h(X))|X = x]\right]$$

the Bayes rule can be obtained by pointwise minimisations on x, i.e.

$$h^*(x) = \arg\min_{y^* \in \mathcal{Y}} E_{Y|X=x}[L(Y, y^*)|X = x] \tag{5.1}$$

Therefore, for given input x, prediction rule h^* should return y^* that minimises the conditional loss. This can be estimated if you have a parametric model f_θ for $Y|X = x$, if $\widehat{\theta}$ is the estimator for data $\mathcal{D}$, then

$$\widehat{h}^{(\mathcal{D})}(x) = \arg\min_{y^* \in \mathcal{Y}} \int_{\mathcal{Y}} f_{\widehat{\theta}}(y|x) L(y, y^*)\, dy$$

This is the *conditional plug-in method*. Alternatively, you may model the joint distribution of X, Y by the parametric form $\pi_\theta(x, y)$, and estimate the distribution of $Y|X = x$ via the Bayes rule

$$\hat{f}_{Y|X}(y|X = x) = \frac{\pi_\theta(x, y)}{\int_y \pi_\theta(x, y)\, dy}$$

again, this can be plugged into the risk, and this is called *generative plug-in method*. To sum up, we have three ways to obtain an estimated prediction rule.

- **No model on** X, Y: empirical risk minimisation
- **Model on** $Y|X$: conditional plug-in method
- **full model on** X, Y: generative plug-in method

The second method is referred to as *semi-parametric* because the model for X is not specified, but the conditional distribution $Y|X$ is modelled. The third method models the joint distribution $\pi_\theta(x, y)$, allowing the generation of new samples from this distribution. Therefore, this method belongs to *generative learning*. In contrast, the first two methods are *discriminative learning*, where h is directly learnt without knowing the full generative process behind the data.

Example 5.9 Consider the one-dimensional linear regression task where the hypothesis class $\mathcal{H} = \{h : \mathcal{X} \to \mathcal{Y} \,|\, h(x) = \beta_0 + \beta_1 x \text{ for some } \beta_0, \beta_1 \in \mathbb{R}\}$ is the set of linear prediction rules, and assign squared error loss $L(y, h(x)) = (y - h(x))^2$. The empirical risk minimisation method is formulated as

$$(\widehat{\beta}_0, \widehat{\beta}_1) := \underset{\beta_0, \beta_1 \in \mathbb{R}}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta_0 + \beta_1 x_i)^2$$

and this may be solved by differentiation directly. So the ERM predictor is $h^{(\mathcal{D})}(x) = \widehat{\beta}_0 + \widehat{\beta}_1 x$.

On the other hand, if we assign $Y|X = x \sim N(\beta_0 + \beta_1 x, \sigma^2)$ (assume σ^2 is known), then the PDF $f_\theta = f_{\beta_0, \beta_1}$ is

$$f_{\beta_0, \beta_1}(y|x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{(y_i - \beta_0 + \beta_1 x)^2}{2\sigma^2} \right\}$$

Using the maximum likelihood estimator $\widehat{\theta}_{\text{MLE}}$, it can be shown that the corresponding estimators $\widehat{\beta}_1$ and $\widehat{\beta}_2$ are the same as the ones obtained from ERM. Then the plug-in predictor is

$$\begin{aligned}
h^{(\mathcal{D})}(x) &= \underset{y^* \in \mathcal{Y}}{\arg\min} \int_{\mathcal{Y}} f_{\widehat{\theta}}(y|x)(y - (y^*)^2)\, dy \\
&= \underset{y^* \in \mathcal{Y}}{\arg\min} -2y^* \int_{\mathcal{Y}} f_{\widehat{\theta}}(y|x) y \, dy + (y^*)^2 \int_{\mathcal{Y}} f_{\widehat{\theta}}(y|x)\, dy \quad \text{a term independent of } y^* \text{ has been removed} \\
&= \underset{y^* \in \mathcal{Y}}{\arg\min} -2y^* E_{\widehat{\theta}}[Y|X = x] + (y^*)^2 \\
&= E_{\widehat{\theta}}[Y|X = x] = \widehat{\beta}_0 + \widehat{\beta}_1 x
\end{aligned}$$

which is the same as the ERM predictor.

Note that in the above example, the plug-in predictor is the estimated mean of $Y|X = x$, this is true if the MLE estimator $\widehat{\theta}_{\text{MLE}}$ is used.

Proposition 5.10 Let $\mathcal{Y} = \mathbb{R}$. Suppose that in the conditional plug-in method, an estimator of parameter θ is taken to be

$$\widehat{\theta} = \widehat{\theta}_{\text{MLE}} = \underset{\theta \in \Theta}{\arg\max} \sum_{i=1}^{n} \log(f_\theta(y_i|x_i))$$

Then the plug-in predictor $h^{(\mathcal{D})}(x) = E_{\widehat{\theta}}[Y|X = x]$.

Remark 5.11 An implicit assumption made here is that (x_i, y_i) are independent samples from the joint distribution of X, Y.

5.2 Excess risk and Generalisation

In the last section, we have addressed the training performance mentioned at the beginning of this chapter, the generalisation performance of prediction rule h can be assessed by the *generalisation risk* $R(h)$. This error represents the average loss across the entire underlying population, with distribution P_0. Another way to define this error is $R(h) - R(h^*)$ (denoted as the *excess risk* of h) so that the generalisation error of the true Bayes rule h^* is 0.

Over-fitting On the other hand, the *generalisation gap* $R(h) - \widehat{R}_n(h)$ checks for *over-fitting*, which occurs when the training loss (empirical risk) is low, but the true population risk $R(h)$ is high. In fact, the empirical risk always underestimates the true risk in the sense that

$$E_{\mathcal{D}}\left[\widehat{R}_n(\widehat{h}^{\mathcal{D}})\right] \leq R(h^*_{\mathcal{H}}) \leq R(\widehat{h}^{\mathcal{D}})$$

An example of over-fitting would be simply drawing a curve that goes through every point for a regression task. See Figure 7, where the grey dots represent individuals in the population that are not sampled, so the right two

figures correspond to the situation in the whole population. The risks are the average loss between the points and the prediction rule (dotted lines and curves).

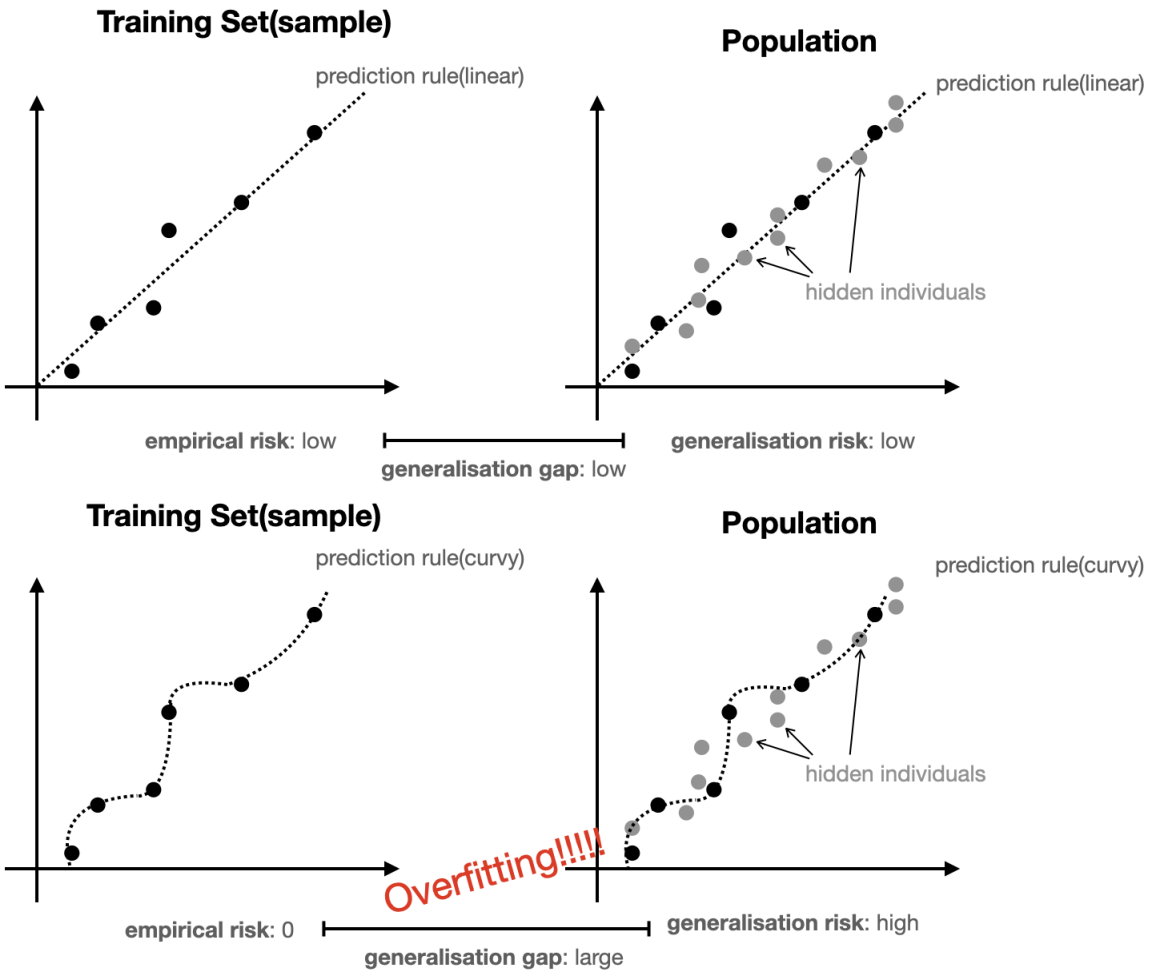


Figure 7: Illustration of over-fitting, and how the generalisation gap detects it.

increasing sample size As the sample size $n \to \infty$, both $R(\widehat{h}^{\mathcal{D}})$ and $\widehat{R}_n(\widehat{h}^{\mathcal{D}})$ converge to $R(h^*_{\mathcal{H}})$. So the generalisation gaps converges to 0.

Regularised ERM One method to avoid over-fitting is via considering the regularised minimisation instead, which adds a penalty function $\text{pen}(h)$ for the complexity of the prediction rule h

$$\arg\min_{h \in \mathcal{H}} \left\{ \widehat{R}_n(h) + \frac{\lambda}{n} \text{pen}(h) \right\}$$

The parameter $\lambda \geq 0$ is termed the *regularization parameter*. Setting $\lambda = 0$ restores the standard Empirical Risk Minimization (ERM), while larger values of λ encourage the learning of simpler models. Experimenting with different λ values helps identify a suitable model. The penalty function $\text{pen}(h)$ is divided by n since models trained with larger samples are less susceptible to over-fitting. The regularised ERM problem converges to $\arg\min_{h \in \mathcal{H}} R(h) = h^*$ as $n \to \infty$, which means it yields consistent estimators for the Bayes prediction rule.

Example 5.12 For linear regression problem, $\mathcal{X} = \mathbb{R}^p$ and the hypothesis class $\mathcal{H} = \{h : \mathcal{X} \to \mathcal{Y} \mid h(x) =$

$\beta^T x$ for some $\beta \in \mathbb{R}^p\}$. The Tikhonov regularisation uses

$$\text{pen}(h) := \|\beta\|^2 = \sum_{j=1}^{p} \beta_j^2$$

the corresponding ERM problem (called ridge regression) is

$$\widehat{\beta} := \arg\min_{\beta \in \mathbb{R}^p} \sum_{i=1}^{n} \left(y_i - x_i^T \beta\right)^2 + \lambda \|\beta\|^2$$

The estimator $\widehat{\beta}$ is termed *ridge regression estimator*, which has the explicit form

$$\widehat{\beta} = (X^T X + \lambda I)^{-1} X^T \boldsymbol{y}$$

all coefficients are smaller compared to the ordinary regression estimator $(X^T X)^{-1} X^T \boldsymbol{y}$.

If the L1 penalty $\text{pen}(h) := 2\|\beta\|_1 = 2\sum_j |\beta_j|$ is used, the regression is called *LASSO* regression. Ridge and LASSO regressions can be combined by using the penalty $\text{pen}(h) := 2\delta\|\beta\|_1 + (1-\delta)\|\beta\|_2^2$, where $\delta \in [0,1]$. This is called *Elastic Net regression*. Motivations behind using these penalty functions and the effect of them on the parameters can be found in Chapter 2 of Auxiliary Notes on Data Science.

Over-parametrisation It was thought that the number of parameters should be controlled, because higher complexity yields larger generalisation risk $R(h)$. However, if there are more parameters than sample size, then $R(h)$ will descend again, which is described as *double-descent*. (see Figure 8) The paper Sa-Couto et al. (2022) sketches the theoretical reason behind double descent.

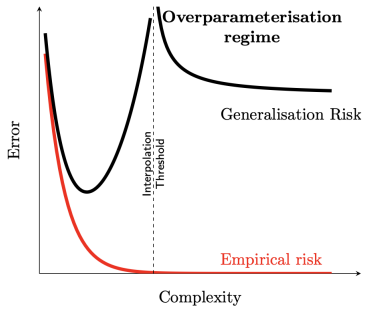


Figure 8: Illustration of double-descent taken from lecture notes

Example 5.13 For linear regression, instead of regressing y on $x \in \mathbb{R}^p$, one could regress y on $\phi(x) \in \mathbb{R}^M$ where $M \geq n$. The extended space $\mathbb{R}^M$ is called *feature space*. The prediction rule takes the form $h(x) = \phi(x)^T \beta$ instead. Let $\Phi(X) := (\phi(x_1), \cdots, \phi(x_n))^T \in \mathbb{R}^{n \times M}$ denote the design matrix, $\Phi^T \Phi$ is not invertible and minimisers of the squared error $\|y - \Phi\beta\|^2$ are not unique. However, one may consider the *minimum-norm* problem

$$\arg\min_{\beta \in \mathbb{R}^M} \|\beta\|^2 \quad \text{subjected to } y = \Phi(X)\beta$$

which has solution $\widehat{\beta} := \Phi^T (\Phi \Phi^T)^{-1} \boldsymbol{y}$.

Another example of over-parametrisation is the neural network, which will be covered in later chapter.

Test sets Given that the true population distribution P_0 is unknown, the generalisation error $R(h) = E_{X,Y \sim P_0}[L(h(X), Y)]$ and generalisation gap $R(h) - \widehat{R}_n(h)$ cannot be evaluated directly. This is often solved by leaving out a subset of

samples $\mathcal{D}_{\text{test}} \subseteq \mathcal{D}$, referred to as the *testing set*. This set is distinct from the *training set*, denoted as $\mathcal{D}_{\text{train}}$, which is employed to train the model and obtain the predictive rule $\widehat{h}^{(\mathcal{D}_{\text{train}})}$. With this setup, the generalisation error can be estimated on the test set via

$$\widehat{R}^{(\mathcal{D}_{\text{test}})}\left(\widehat{h}^{(\mathcal{D}_{\text{train}})}\right) := \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{i \in \text{test}} L\left(y_i, \widehat{h}^{(\mathcal{D}_{\text{train}})}(x_i)\right)$$

and so the generalisation gap can be estimated similarly via

$$\widehat{R}^{(\mathcal{D}_{\text{test}})}\left(\widehat{h}^{(\mathcal{D}_{\text{train}})}\right) - \widehat{R}^{(\mathcal{D}_{\text{train}})}\left(\widehat{h}^{(\mathcal{D}_{\text{train}})}\right)$$

where $\widehat{R}^{(\mathcal{D}_{\text{train}})}$ is the empirical risk in the training set.

Model selection and Validation set If there are several hypothesis classes $\mathcal{H}_1, \cdots, \mathcal{H}_M$ to choose from, for example, the same model with different hyper-parameters, the model selection can be incorparated into the training stage by leaving out a set called *validation set* $\mathcal{D}_{\text{val}}$ which is not used for training. ERM on each class $\mathcal{H}_j$ is performed, obtaining

$$\widehat{h}_j^{\mathcal{D}_{\text{train}}} := \arg\min_{h \in \mathcal{H}_j} \widehat{R}^{\mathcal{D}_{\text{train}}}(h)$$

The best model is chosen by evaluating the generation error $R(h)$ using the validation set, i.e.

$$\widehat{m} := \arg\min_j \widehat{R}^{\mathcal{D}_{\text{val}}}(\widehat{h}_j^{\mathcal{D}_{\text{train}}})$$

The returned prediction rule is trained using both the training set and validation set, i.e.

$$\widehat{h}^{(\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{val}})} := \arg\min_{h \in \mathcal{H}_{\widehat{m}}} \widehat{R}^{(\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{val}})}(h)$$

The size of training set, validation set and test sets should be chosen carefully. Small validation and test sets lead to under-estimation of the generalisation error, whereas small training sets have large estimation error.

Cross-validation *T-fold cross-validation* splits the training set into T folds $\mathcal{D}_t$, each fold acts as a validation set in turn. The model selection uses the average validation error of all folds. The full procedure is shown in Algorithm 1. Cross-validation is especially useful when a lot of data is required to train the learner, or the size of whole dataset is small.

Algorithm 1 T-fold Cross-validation

Input:

- Dataset $\mathcal{D}$
- train-test split rate $a \in (0,1)$
- number of folds T
- empirical risk function $\widehat{R}$
- proposed models/hypothesis classes: $\mathcal{H}_m$, $m = 1, \cdots, M$

Randomly split the data into test set $\mathcal{D}_{\text{test}}$ of size $(1-a)n$ and train set $\mathcal{D}_{\text{train}}$ of size an.
Partition the training set $\mathcal{D}_{\text{train}}$ into T folds $\mathcal{D}_t$
Define the t'th training set $\mathcal{D}_{\text{train},t} := \bigcup_{t' \neq t} \mathcal{D}_{t'}$ and t'th validation set $\mathcal{D}_{\text{val},t} := \mathcal{D}_t$
for $j = 1, \cdots, M$ **do**
    for $t = 1, \cdots, T$ **do**
        Compute the ERM estimator within $\mathcal{H}_j$:

$$\widehat{h}_j^{(\mathcal{D}_{\text{train},t})} := \arg\min_{h \in \mathcal{H}_j} \widehat{R}^{(\mathcal{D}_{\text{train},t})}(h)$$

    end for
    Compute the average risk of cross-validation

$$r_j := \frac{1}{T} \sum_{t=1}^{T} \widehat{R}^{(\mathcal{D}_{\text{val},t})}(\widehat{h}_j^{(\mathcal{D}_{\text{train},t})})$$

end for
pick the best model $\widehat{m}$ by $\widehat{m} := \arg\min_j r_j$
retrain the model using all data by

$$\widehat{h}^{(\mathcal{D}_{\text{train}}} := \arg\min_{h \in \mathcal{H}_{\widehat{m}}} \widehat{R}^{(\mathcal{D}_{\text{train}})}(h)$$

return

- the estimated prediction rule $\widehat{h}^{(\mathcal{D}_{\text{train}})}$
- training performance $\widehat{R}^{(\mathcal{D}_{\text{train}})}(\widehat{h}^{(\mathcal{D}_{\text{train}})})$
- approximate generalisation error $\widehat{R}^{\mathcal{D}_{\text{test}}}(\widehat{h}^{(\mathcal{D}_{\text{train}})})$

5.3 Trade-offs

Approximation-estimation trade-off Define the best possible prediction rule in hypothesis class $\mathcal{H}$ as

$$h_{\mathcal{H}}^* := \arg\min_{h \in \mathcal{H}} R(h)$$

then the excess risk of $\widehat{h}^{(\mathcal{D})}$ can be decomposed in the following manner

$$R(\widehat{h}^{(\mathcal{D})}) - R(h^*) = (R(h_{\mathcal{H}}^*) - R(h^*)) + R(\widehat{h}^{(\mathcal{D})}) - R(h_{\mathcal{H}}^*)$$

The expression $R(h_{\mathcal{H}}^*) - R(h^*)$ represents the *approximation error* associated with the chosen hypothesis class $\mathcal{H}$, while the second term denotes the *generalisation/estimation error* within the class $\mathcal{H}$. The significance of both the selection of $\mathcal{H}$ and the quality of the chosen prediction rule $h^{(\mathcal{D})}$ is evident in their collective impact on the overall performance.

- For larger hypothesis class $\mathcal{H}$: potentially larger estimation errors due to large number of parameters; potentially smaller approximation error due to better flexibility of the model
- For smaller hypothesis class $\mathcal{H}$: potentially smaller estimation errors due to less number of parameters; potentially larger approximation error due to stronger model assumptions

You may alternatively consider the *expected excess risk*, which bears the same decomposition

$$E_{\mathcal{D}}\left[R(\widehat{h}^{(\mathcal{D})}) - R(h^*)\right] = (R(h^*_{\mathcal{H}}) - R(h^*)) + \left(E_{\mathcal{D}}\left[R(\widehat{h}^{(\mathcal{D})})\right] - R(h^*_{\mathcal{H}})\right)$$

Remark 5.14 $E_{\mathcal{D}}$ is computed w.r.t. the training set, addressing the question: what if a different sample is drawn from the population? Assuming an infinite population, $P = \mathbb{N}$, the sample $\mathcal{D} \subseteq P$ essentially represents a set of indices. The distribution of $\mathcal{D}$ relies on the sampling method. Conversely, the expectation $E_{(X,Y)\sim P_0}$ used in the risk R pertains to the distribution of the variables X, Y themselves, which is unknown.

bias-variance trade-off Under the square loss, the excess risk of any prediction rule h is

$$R(h) - R(h^*) = E_X[(h(X) - h^*(X))^2]$$

Proof. Bridge the risk of h^* to involve h:

$$\begin{aligned} R(h^*) &= E_{X,Y\sim P_0}[(h^*(X) - Y)^2] = E_{X,Y}[(h^*(X) - h(X) + h(X) - Y)^2] \\ &= E_X[(h^*(X) - h(X))^2] + E_{X,Y}[(h(X) - Y)^2] + 2E_{X,Y}[(h(X) - Y)(h^*(X) - h(X))] \\ &= E_X[(h^*(X) - h(X))^2] + R(h) + 2E_{X,Y}[(h(X) - Y)(h^*(X) - h(X))] \end{aligned}$$

The last term can be resolved by tower rule:

$$\begin{aligned} E_{X,Y}[(h(X) - Y)(h^*(X) - h(X))] &= E_X[E_{Y|X}[(h(X) - Y)(h^*(X) - h(X))] \mid X] \\ &= E_X[(h^*(X) - h(X))(h(X) - E_{Y|X}[Y \mid X])] \\ &= -E_X[(h^*(X) - h(X))^2] \quad \text{because the Bayes rule is } E_{Y|X}[Y \mid X] \text{ under square loss} \end{aligned}$$

and so the desired equation follows. □

If $h = \widehat{h}^{(\mathcal{D})}$, which is trained from dataset $\mathcal{D}$, then the expected excess risk is

$$E_{\mathcal{D}}[R(\widehat{h}^{(\mathcal{D})}) - R(h^*)] = E_{\mathcal{D}}[E_X[(\widehat{h}^{(\mathcal{D})}(X) - h^*(X))^2]] = E_X[E_{\mathcal{D}}[(\widehat{h}^{(\mathcal{D})}(X) - h^*(X))^2]]$$

where the second equation holds by Fubini's theorem. The inner term $E_{\mathcal{D}}[(\widehat{h}^{(\mathcal{D})}(X) - h^*(X))^2]$ admits the bias-variance decomposition. Let $\overline{h}(x) = E_{\mathcal{D}}[\widehat{h}^{(\mathcal{D})}(X)]$, then

$$E_{\mathcal{D}}[(\widehat{h}^{(\mathcal{D})}(X) - h^*(X))^2] = \underbrace{(\overline{h}(X) - h^*(X))}_{=:b_{\overline{h}}(X)}{}^2 + \underbrace{E_{\mathcal{D}}\left[\left(\widehat{h}^{(\mathcal{D})}(X) - \overline{h}(X)\right)^2\right]}_{=:v_{\overline{h}}(X)}$$

the proof of this equation is left as an exercise. And so

$$E_{\mathcal{D}}[R(\widehat{h}^{(\mathcal{D})}) - R(h^*)] = E_{\mathcal{X}}[b_{\overline{h}}(X)^2] + E_{\mathcal{X}}[v_{\overline{h}}(X)^2]$$

The first term represents *bias*, quantifying the average deviation of the estimated prediction rule $\widehat{h}^{(\mathcal{D})}$ from the Bayes rule. Conversely, the second term measures the variation of $\widehat{h}^{(\mathcal{D})}$ when different training sets $\mathcal{D}$ are employed. A *trade-off* between bias and variance is inherent. For instance, when $\widehat{h}^{(\mathcal{D})}(X)$ is the Empirical Risk Minimization (ERM) under the hypothesis class $\mathcal{H}$, smaller bias is achieved with a larger $\mathcal{H}$, while smaller variance is attained with a smaller $\mathcal{H}$. In plug-in methods, the trade-off relates to the number of parameters (i.e. the dimension of θ) used in the model.

6 Optimisation

As demonstrated in the previous chapter, empirical risk minimisation often involves solving optimization problems of the form

$$\arg\min_{\theta\in\Theta} f(\theta)$$

potentially with constraints such as $g(\theta) \geq 0$ or $h(\theta) = 0$, where g and h are functions. For instance, in ridge regression, we control complexity by constraining the parameter β using $g(\beta) := |\beta|_2^2 \leq t$ for some $t > 0$. This is equivalent to minimizing the Lagrangian function $J(\beta) = f(\beta) + \lambda g(\beta)$, where f is the square error and $\lambda \geq 0$. This formulation precisely aligns with the regularised empirical risk minimization. The relationship between constraints and Lagrangian functions is introduced in SC4 Advanced Statistical Machine Learning, with technical details available in Auxiliary Notes on Optimization.

Gradient descent As long as the gradient of the Laplacian J exists, the gradient descent algorithm can be applied. Intuitively, consider $J(\theta)$ as the elevation of the ground at θ, representing a landscape with mountains. The objective is to locate the lowest point. The gradient at θ indicates the direction of the steepest ascent (see gradient(Khan Academy) for an intuitive explanation). Taking a step along $-\nabla_\theta J(\theta)$ corresponds to the steepest downhill direction, and repeating this process converges toward the lowest point. (see graphical illustration in 9) The choice of the initial point is crucial; a poor selection may result in getting stuck in a local minimum point. It is recommended to randomly select several starting points and run the algorithm to observe which one converges first. This approach helps mitigate the sensitivity to the initial conditions and increases the likelihood of finding the global minimum.

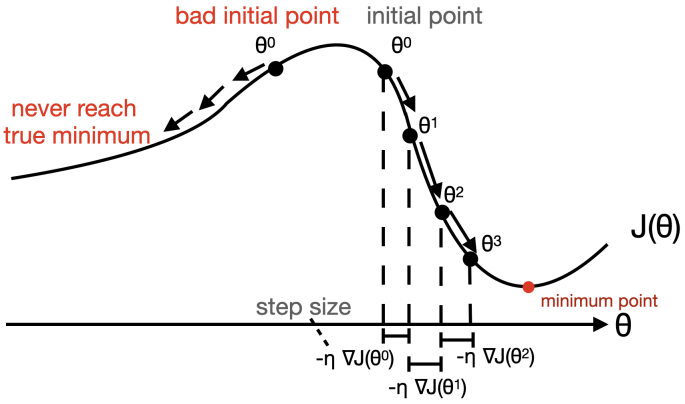


Figure 9: Illustration of gradient descent algorithm in a simple one-dimensional case

The step size η, also known as the *learning rate*, significantly influences the algorithm's performance, as illustrated in Figure 10. Experimentation with various values of η is often necessary to determine the most suitable choice. Adjusting the learning rate allows for finding a balance between convergence speed and stability in the optimisation process. Sometimes we consider flexible learning rate η_t, and let $\eta_t \to 0$ as $t \to 0$ to allow more delicate controls when the algorithm is close to convergence. For example,

$$\eta_t := \frac{\eta}{(1 + t/t_0)^a}$$

for some $t_0 > 0$ and tuning parameter $a \in (1/2, 1]$.

Newton's method If the Lagrangian (objective) $J(\theta)$ is twice differentiable, then an appropriate choice for the step size is $\eta = (\nabla_\theta^2 J(\theta))^{-1}$, leading to the method known as *Newton-Raphson gradient descent*. The rationale

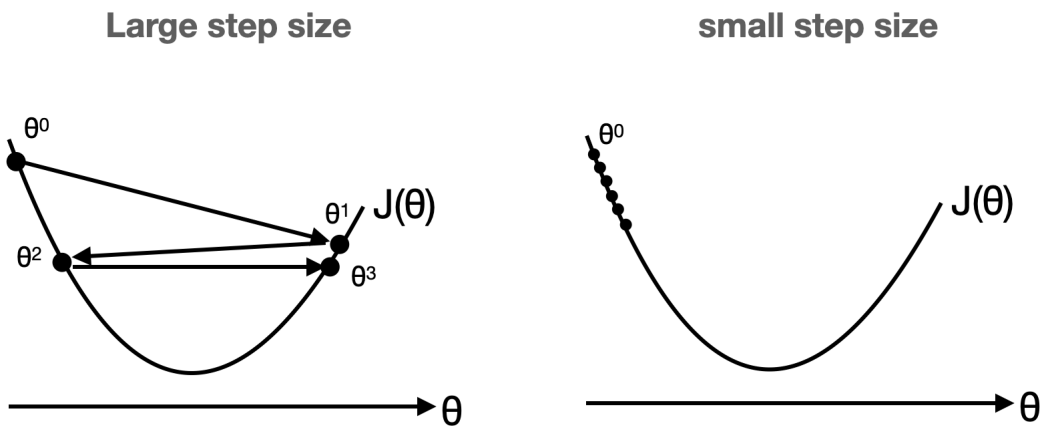


Figure 10: Illustration of the bad choices of step size. Large step size may not converge, whereas small step size suffers from slow convergence

behind this lies in considering the quadratic approximation of the objective function

$$J(\theta + \delta) \approx J(\theta) + \nabla_\theta J(\theta)^T \delta + \frac{1}{2} \delta^T \nabla_\theta^2 J(\theta) \delta$$

The descent of J in direction δ is $J(\theta) - J(\theta + \delta) = -\nabla_\theta J(\theta)^T \delta - \frac{1}{2} \delta^T \nabla_\theta^2 J(\theta) \delta$ with maximum attained when $\delta = -(\nabla_\theta^2 J(\theta))^{-1} \nabla_\theta J(\theta)$.

7 Classifiers

At the outset of this chapter, we will employ the decision theory framework discussed in the previous chapter to derive the general form of the prediction rule for classification.

For classification, $\mathcal{Y} = \{1, \cdots, K\}$, and prediction rule $h : \mathcal{X} \to \mathcal{Y}$ assigns a class label to input x. A typical loss function is the 0-1 loss

Definition 7.1 — 0-1 loss. The 0-1 loss for the classification task is defined as

$$L(y, h(x)) = 1 - 1_{y=h(x)} = \begin{cases} 1, & y \neq h(x) \\ 0, & y = h(x) \end{cases}$$

i.e. it is a binary indicator, where 1 represents a misclassification, and 0 denotes a correct label assignment.

Under a 0-1 loss, using Equation 5.1, the Bayes rule is

$$\begin{aligned} h^*(x) &= \arg\min_{y^* \in \mathcal{Y}} E_{Y|X=x}[1 - 1_{y=y^*}] \\ &= \arg\min_{y^* \in \mathcal{Y}} 1 - P(Y = y^*|X = x) \\ &= \arg\max_{k \in \{1, \cdots, K\}} P(Y = k|X = x) \end{aligned}$$

and the corresponding risk is

$$R(h^*) = 1 - E_X \left[max_{k \in \{1, \cdots, K\}} P(Y = k|X) \right]$$

Therefore, a classifier h should take the form

$$h(x) = \arg\max_{k \in \mathcal{Y}} f_k(x) \tag{7.1}$$

where the *discriminant functions* $f_k(x)$ should estimate $P(Y = k|X = x)$, the probability that x belongs to class k.

In the case of binary classification, $\mathcal{Y} = \{0, 1\}$ (alternatively, $\mathcal{Y} = \{-1, 1\}$ is used), there are only two discriminant functions and the larger one should be selected. Note that $P(Y = 1 \,|\, X = x) + P(Y = 0 \,|\, X = x) = 1$, so

$$h^*(x) = \arg\max_{k \in \{1, -1\}} P(Y = k \,|\, X = x) = \begin{cases} 1, & P(Y = 1 \,|\, X = x) \geq 1/2 \\ 0, & P(Y = 1 \,|\, X = x) < 1/2 \end{cases}$$

Therefore, one discriminant function $f(x) = f_1(x)$ is enough for binary classification (another discriminant function takes the form $f_0(x) = 1 - f_1(x)$, which is redundant), and prediction rule h should take the form

$$h(x) = \begin{cases} 1, & f(x) \geq 1/2 \\ 0, & f(x) < 1/2 \end{cases}$$

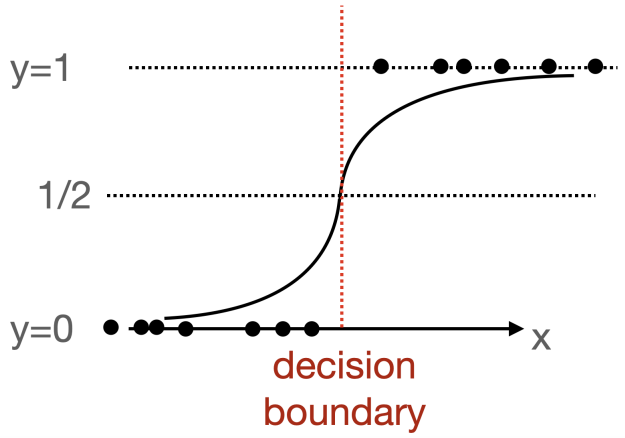


Figure 11: logistic regression with one feature

In *logistic regression*, the sigmoid function is employed for the class probability. A straightforward example is illustrated in Figure 11, where the solid line represents the fitted logistic function. In this case, the dots on the x-axis are from class 0, and the dots on $y = 1$ are from class 1. The decision boundary between two classes, marked by a red dotted line, occurs at the x value where $f_1(x) = 1/2$. Further technical details on logistic regression will be provided in section 7.2.

Definition 7.2 — Decision Boundary. For a classifier $h(x) := \arg\max_k f_k(x)$, the decision boundary between two classes k and k' is

$$\{x \in \mathcal{X} : f_k(x) = f_{k'}(x)\}$$

7.1 Performance measures

The evaluation of classifier performance often relies on *contingency tables*. For the binary case $\mathcal{Y} = 0, 1$, the table (confusion matrix) is presented below.

	$y = 1$	$y = 0$
$h(x) = 1$	TP(true positive)	FP(false positive)
$h(x) = 0$	FN(false negative)	TN(true negative)

For example, the empirical risk under 0-1 loss is

$$\widehat{R}_n^{(\mathcal{D})}(h) = \frac{\text{FN} + \text{FP}}{n}$$

where $n = \text{TP} + \text{TN} + \text{FP} + \text{FN}$.

Often the classes bear specific meanings. For example, $y = 1$ means a patient has a specific disease and x_i are be symptoms of the patient. A test $h(x)$ detects whether the patient has the disease or not based on the symptoms. Some commonly used measures using entries of the contingency table are

Precision:

$$\frac{\text{TP}}{\text{TP} + \text{FP}}$$

how confident we are when the classifier predicts positive ($h(x) = 1$).

Recall

$$\frac{\text{TP}}{\text{TP} + \text{FN}}$$

the proportion of positives identified by the classifier.

Specificity

$$\frac{\text{TN}}{\text{TN} + \text{FP}}$$

the proportion of negatives identified by the classifier.

Accuracy

$$\frac{\text{TP} + \text{TN}}{n}$$

proportion of correct predictions by the classifier.

F-score

$$2\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

It encapsulates two key pieces of information: whether precision and recall are high, and the proximity of precision and recall. Refer to Figure 12 for a 3-D plot illustrating the F-score function.

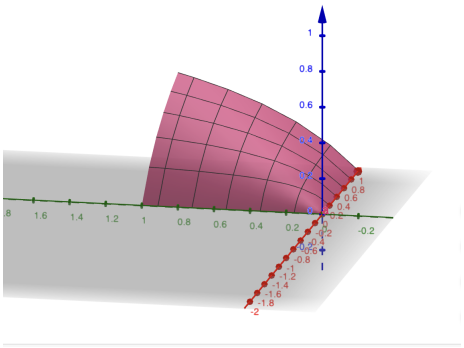


Figure 12: Plot of F-score function, with the F-score on z-axis and precision, recall on x, y axes

Flexible Loss Type I error (false positive) refers to $h(x) = 1, y = -1$ and Type II error (false negative) is when $h(x) = -1, y = 1$. The following loss assigns different weights to two errors:

$$L(y, h(x)) = a1_{y=-1,h(x)=1} + b1_{y=1,h(x)=-1}$$

Suppose $\eta(x) = P(Y = 1|X = x)$, the decision boundary of Bayes classifier is at $t := a/(a + b) \in [0, 1]$ and

$$h^*(x) = \begin{cases} 1, & \eta(x) \geq t \\ 0, & \eta(x) < t \end{cases}$$

ROC curve The performance of an estimate of η, $\widehat{\eta}(x)$ is assessed by the Receiver-operating characteristic(ROC) curve. Suppose it has decision boundary $t \in [0, 1]$, let $\widehat{h}_t(x)$ be the prediction rule. Define true positive rate $\alpha(t) := P(\widehat{h}_t(X) = 1 \,|\, Y = 1)$ and false positive rate $\beta(t) := P(\widehat{h}_t(X) = 1 \,|\, Y = 0)$. Note that $\alpha(1) = \beta(1) = 0$ (decision boundary at 1, $\widehat{h}_t(X) = 0$ with probability 1) and $\alpha(0) = \beta(0) = 1$. ROC curve is a plot of $\alpha(t)$ against $\beta(t)$, which always hits the two points $(0, 0)$ and $(1, 1)$. Area under curve (AUC) is a single numerical summary of the ROC.

An ideal classifier: TPR $\alpha(t) = 1$ unless $t = 1$ (extreme decision boundary). AUC $= 1$

A random classifier ($\eta(x)$ randomly sampled from $[0, 1]$): $\alpha(t) = \beta(t) = t$. AUC $= 0.5$

So a good classifier should have AUC close to 1, and the ROC curve closer to that of the ideal classifier. (see Figure 13)

Definition 7.3 — Area under curve(AUC). The area under ROC curve is defined as

$$\text{AUC} := \int_1^0 \beta(t) d\alpha(t)$$

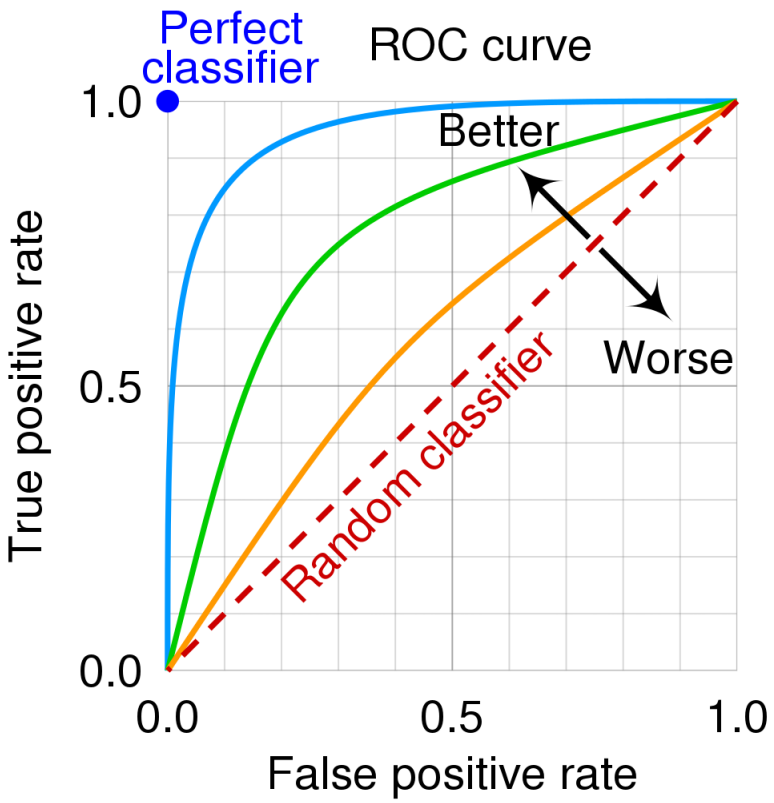


Figure 13: ROC curves (taken from wikipidea)

7.2 Logistic Regression

We begin with a straightforward classifier for the binary classification task ($\mathcal{Y} = \{\pm 1\}$). The conditional plug-in estimator only necessitates the specification of $P(Y = 1 \mid X = x)$, as $P(Y = -1 \mid X = x) = 1 - P(Y = 1 \mid X = x)$. If we use the *sigmoid function* with parameter β, i.e.

$$P(Y = 1 \mid X = x) = \text{sig}(\beta^T x) := \frac{1}{1 + e^{-\beta^T x}}$$

then $P(Y = -1 \mid X = x) = \text{sig}(-\beta^T x)$. By the wise choice of class labels, the two probabilities can be compactly written as $P(Y = y \mid X = x) = \text{sig}(y\beta^T x)$, i.e. $(Y|X = x) \sim \text{Bern}(\sigma(\beta^T x))$.
Recall that the Bayes prediction rule h^* should select the class with highest class probability. Therefore, using the identity below

$$\text{logit}\left(P(Y = 1 \mid X = x)\right) = \log\left(\frac{P(Y = 1 \mid X = x)}{P(Y = -1 \mid X = x)}\right) = \log\left(\frac{1 + e^{\beta^T x}}{1 + e^{-\beta^T x}}\right) = \beta^T x$$

where the logit function $\text{logit}\,(p) = \log(1/(1-p))$ is the inverse of sigmoid function. The logit of class probability is linear, so the problem of fitting a value for β is a generalised linear regression with logit link, named *logistic regression*.

Linear Discriminant Instead of using the class probability as a discriminant function, f can be chosen to be the linear function $f(x) = \beta^T x$ instead, and the Bayes prediction rule is

$$h^*(x) := \underset{y \in \{\pm 1\}}{\arg\max} = \begin{cases} 1, & \text{if } f(x) = \beta^T x \geq 0 \\ -1, & \text{if } \beta^T x < 0 \end{cases} \tag{7.2}$$

The linear hypothesis class $\{f : f(x) = \beta^T x \text{ where } \beta \in \mathbb{R}^p\}$ is easier to optimise over. The conditional plug-in methods estimates h^* using an estimator for β, for example, the MLE estimator can be found by maximising the

log-likelihood

$$\begin{aligned} l(\beta; x, y) := \log(L(\beta; x, y)) &= \log\left(\prod_i P(Y = y_i \mid X = x_i)\right) \\ &= \sum_i \log\left(\mathrm{sig}(y_i \beta^T x_i)\right) \\ &= -\sum_i \log\left(1 + e^{-y_i \beta^T x_i}\right) \end{aligned}$$

and so the MLE is

$$\widehat{\beta}_{\mathrm{MLE}} = \underset{\beta \in \mathbb{R}^p}{\arg\max}\ l(\beta; x, y) = \underset{\beta \in \mathbb{R}^p}{\arg\min} \frac{1}{n} \sum_i \log\left(1 + e^{-y_i \beta^T x_i}\right) \tag{7.3}$$

adding the factor $1/n$ is to make the objective function

$$J(\beta) := \frac{1}{n} \sum_i \log\left(1 + e^{-y_i \beta^T x_i}\right)$$

easier to handle. Analytical solution of the minimisation problem can be hard to find. Noting that J is differentiable, one could use (stochastic) gradient descent. The gradient and Hessian are

$$\nabla J(\beta) = -\frac{1}{n} \sum_i \mathrm{sig}(-y_i\, \beta^T x_i) y_i x_i$$

$$\nabla^2 J(\beta) = \frac{1}{n} \sum_i \mathrm{sig}(y_i\, \beta^T x_i) \mathrm{sig}(-y_i\, \beta^T x_i) x_i x_i^T \succ 0$$

The Hessian is positive semi-definite, so J is convex, which allows various convex optimisation techniques.

Newton-Raphson and IRLS Making use of the Hessian, one may employ the Newton-Raphson algorithm instead:

$$\beta^{(t+1)} = \beta^{(t)} - \left(\nabla^2 J(\beta^{(t)})\right)^{-1} \nabla J(\beta^{(t)})$$

Denoting $\mu_i := \mathrm{sig}(x_i^T \beta)$, we have $1 - \mu_i = 1 - \mathrm{sig}(\beta^T x_i) = \mathrm{sig}(-\beta^T x_i)$, so the gradient and Hessian takes compact vector/matrix form as below:

$$\begin{aligned} n \nabla J(\beta) &= -\sum_i \mathrm{sig}(-y_i\, \beta^T x_i) y_i x_i \\ &= \sum_i \begin{cases} -\mathrm{sig}(-\beta^T x_i) x_i, & \text{if } y_i = 1 \\ \mathrm{sig}(\beta^T x_i) x_i, & \text{if } y_i = -1 \end{cases} \\ &= \sum_i \begin{cases} -(1 - \mu_i) x_i = \mu_i x_i - x_i, & \text{if } y_i = 1 \\ \mu_i x_i, & \text{if } y_i = -1 \end{cases} \\ &= \sum_i (\mu_i - c_i) x_i \quad \text{where } c_i := 1_{y_i = 1} \\ &= X^T(\mu - c) \quad \text{where design matrix } X \text{ has rows } x_i^T \end{aligned}$$

and

$$\begin{aligned} n \nabla^2 J(\beta) &= \sum_i \mathrm{sig}(y_i\, \beta^T x_i) \mathrm{sig}(-y_i\, \beta^T x_i) x_i x_i^T \\ &= \sum_i \mu_i (1 - \mu_i) x_i x_i^T \\ &= X^T S X \quad \text{where } S := \mathrm{diag}(\mu_i (1 - \mu_i)) \end{aligned}$$

For the t'th step of Newton-Raphson's algorithm, denoting $\mu_i^{(t)} := \text{sig}(x_i^T \beta^{(t)})$ and $S^{(t)} := \text{diag}(\mu_i^{(t)}(1-\mu_i^{(t)}))$. The next value of β takes the form (for simplicity, ignore the label (t))

$$\begin{aligned}
\beta^{(t+1)} &= \beta - \left(\nabla^2 J(\beta)\right)^{-1} \nabla J(\beta) \\
&= \beta + (X^T S X)^{-1} X^T (c - \mu) \\
&= (X^T S X)^{-1} X^T S X \beta + (X^T S X)^{-1} X^T (c - \mu) \\
&= (X^T S X)^{-1} X^T S \left(X\beta + S^{-1}(c-\mu)\right) \quad \text{diagonal matrix } S \text{ is invertible as } \mu_i \in (0,1) \\
&= (X^T S X)^{-1} X^T S z
\end{aligned}$$

where $z := X\beta + S^{-1}(c-\mu)$. This is the solution of weighted least square problem

$$\arg\min_{\beta} (z - X\beta)^T S (z - X\beta)$$

where $S_{ii} = \mu_i(1-\mu_i)$ are the weights. And so in this case, the Newton-Raphson algorithm aligns with the so called *iterative reweighed least square* algorithm.

Regularisation Suppose the data set $\{x_i\}$ is linearly separable, meaning there exists a linear classifier with parameter $\tilde{\beta} \in \mathbb{R}^p$ that separates all data: $x_i^T \tilde{\beta} > 0$ if $y_i = 1$ and $x_i^T \tilde{\beta} < 0$ if $y_i = -1$ (or succinctly, $y_i x_i^T \tilde{\beta} > 0$ for all i). Then,

$$\lim_{c\to\infty} J(c\tilde{\beta}) = \lim_{c\to\infty} \frac{1}{n} \sum_{i=1}^{n} \log(1 + e^{-c y_i x_i^T \tilde{\beta}}) = \sum_{i=1}^{n} \log(1) = 0$$

So, the minimisation problem (7.3) potentially yields $\widehat{\beta} = c\tilde{\beta}$ for large c. The estimated class probabilities $\widehat{\eta}(x) = \text{sig}(c\tilde{\beta}^T x)$ are close to 0 or 1 depending on the sign of $\tilde{\beta}^T x$ (referred to as *over-confident* predictions). While this value of $\widehat{\beta}$ still produces the same prediction rule (which only depends on the sign of $\widehat{\beta}^T x$), a small change in the input value could result in the opposite class, and there is no informative class probability produced. One may address this issue by turning problem 7.3 into a regularized version, for example,

$$\arg\min_{\beta \in \mathbb{R}^p} \; J(\beta) + \|\beta\|_2^2$$

so that estimates like $\widehat{\beta} = c\tilde{\beta}$ with large c are penalised.

Feature Map Suppose the data set is not linearly separable, for example, the one in figure 14, then logistic regression does not work.

Figure 14: Non-linearly-separable classification

But as is shown in figure 15, the data set becomes separable if they are squared. The map $\phi(x_i) = (x_i, x_i^2)$ adds a new dimension to the one dimension data points x_i, thereby making the problem linearly separable in the higher dimensional *feature space* $\phi(\mathcal{X})$. The map ϕ is often called *feature map*, because it extracts "features" of x_i. One could replicate all the procedures of logistic regression by replacing x by $\phi(x)$, but the parameter β has a higher

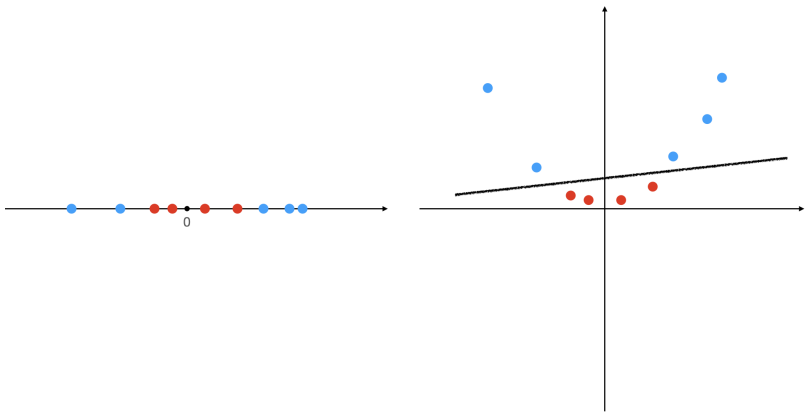


Figure 15: transforming non-linearly-separable problem to a higher dimension

dimension in this case.

When $x_i \in \mathcal{X}$ is two-dimensional, there are 6 possible (two-term) features to extract:

$$1, x_{i1}, x_{i2}, x_{i1}x_{i2}, x_{i1}^2, x_{i2}^2$$

The possible choices of the feature space can rapidly increase even when considering only two-term combinations, resulting in high computational costs. It is advisable to use a high-dimensional feature map only if the estimator depends solely on the inner product $x_i^T x_i$ (not satisfied by the logistic regression), In such cases, the kernel method can be applied (see SC4 Advanced Statistical Machine Learning). Alternatively, pick only the terms which may help classification (depends on the specific problem).

Extension to multi-class The multi-class logistic regression specifies the conditional probabilities as soft-max functions: for each $c \in \{1, \cdots, C\}$,

$$P(Y = k \mid X = x) = \frac{\exp(x^T \beta_c)}{\sum_{j=1}^{C} \exp(x^T \beta_j)}$$

equivalently, $(Y|X = x) \sim \text{Multinom}(\text{softmax}(Bx + b))$ where the rows of matrix $B \in \mathbb{R}^{C \times p}$ are coefficients β_j, the bias $b \in \mathbb{R}^C$ are added for flexibility, and

$$\text{softmax}(z) := \left(\frac{e^{z_1}}{\sum_i e^{z_i}}, \cdots, \frac{e^{z_C}}{\sum_i e^{z_i}} \right)^T$$

Then a plug-in estimator can be formed using estimators $\{\widehat{\beta}_c\}$ for $\{\beta_c\}$. The discriminant functions $f_c(x) := P(Y = c \mid X = x)$ are not linear in this case, but the decision boundaries between any two classes k, l are linear because

$$\log \left(\frac{P(Y = k \mid X = x)}{P(Y = l \mid X = x)} \right) = (\beta_k - \beta_l)^T x$$

An alternative approach to mapping the linear combination $z := Bx + b$ to class probabilities $f_c(x)$ is to use a simpler transformation:

$$f_c(x) = \frac{z_c}{\sum_{k=1}^{C} z_k}$$

However, this transformation is unstable under translations of the values of z_k. For instance, consider $z_1 = (1, 2, 3)$ and $z_2 = (101, 102, 103)$ (which happens when the intercepts differ by 100). They are mapped to class probabilities $(0.167, 0.333, 0.5)$ and $(0.330, 0.333, 0.337)$ respectively. This instability is an undesirable feature, as the outputs of linear regression could shift. In contrast, $\text{softmax}(z_1) = \text{softmax}(z_2) = (0.09, 0.245, 0.665)$. The softmax function remains unchanged by translations of z_k.

7.3 Surrogate Loss and Linear Classifiers

In the binary case, the 0-1 loss function can be expressed as a function taking only one input, utilising the discriminant function f as defined in Equation 7.2:

$$L(y, h(x)) := 1_{y \neq h(x)} = 1_{yf(x)<0} = \phi_{0\text{-}1}(yf(x))$$

Here, the function $\phi_{0\text{-}1}(z) := 1_{z<0}$ is neither differentiable nor convex. To facilitate risk minimisation (an optimisation problem), an alternative approach is to propose a convex and continuous loss function ϕ defined on the product $yf(x)$. The corresponding risk is defined as below:

Definition 7.4 — **ϕ-risk.** For a loss function $\phi : \mathbb{R} \to \mathbb{R}^+$, the ϕ-risk is

$$R_\phi(f) := E[\phi(Y(f(X))] = E[\eta(X)\phi(f(X)) + (1 - \eta(X))\phi(-f(X))]$$

where $\eta(x) := P(Y = 1 \mid X = x) = \text{sig}(f(x))$.

Note that $\psi_{0\text{-}1}$-risk is exactly the risk under 0-1 loss function. When finding a good proxy ψ for $\psi_{0\text{-}1}(z)$, one should make sure

$$\widehat{f}_\phi := \inf_f R_\phi(f) \approx \widehat{f}_{\phi_{0\text{-}1}} := \inf_f R_{\phi_{0\text{-}1}}(f)$$

. The minimum requirement is that ψ is a stronger penalisation than $\psi_{0\text{-}1}$ at any point.

Definition 7.5 — **Surrogate Loss function.** A continuous and convex loss function $\phi : \mathbb{R} \to \mathbb{R}^+$ is surrogate loss function if

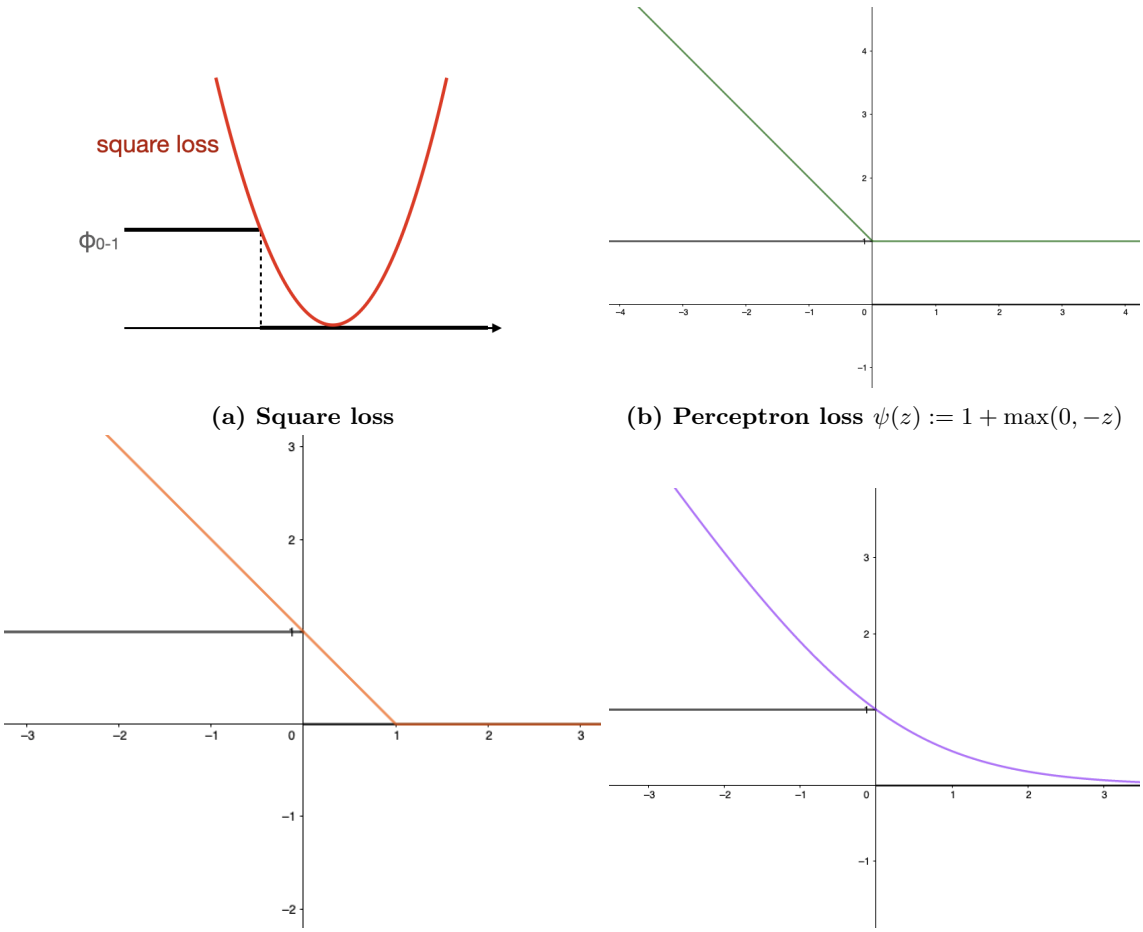
$$\psi(z) \geq \psi_{0\text{-}1}(z) \; \forall z \in \mathbb{R}$$

Not every surrogate loss function is reasonable. For instance, the squared loss $\psi(z) := (1 - z)^2$ (Figure 16a) penalizes large positive values of $yf(x)$, even if $f(x)$ correctly identifies the label. Ideally, a good surrogate loss function should decrease simultaneously. Examples in Figure 16, excluding the square loss, satisfy this property.

The perceptron loss only penalises wrong classifications ($yf(x) < 0$), the hinge loss favours a discriminant function f with $|f(x)| > 1$, and the logit loss encourages a larger $|f(x)|$ (resulting in larger class separation). In fact, the logit loss aligns with the conditional plug-in estimator (logistic regression) using MLE as defined in Equation 7.3. Suppose one explores the hypothesis class of linear classifiers, $f(x) = \beta^T \phi(x)$ for some parameter β and feature map ϕ, then the optimal classifier under the loss ψ can be found by minimizing the objective function

$$J(\beta) := \frac{1}{n} \sum_{i=1}^{n} \psi(y_i \phi(x_i)^T \beta)$$

which can be solved using (stochastic) gradient descent.



(a) Square loss **(b) Perceptron loss** $\psi(z) := 1 + \max(0, -z)$

(c) Hinge loss $\psi(z) := \max(0, 1 - z)$ **(d) Logit loss** $\psi(z) := \log(1 + e^{-z})/\log(2)$

Figure 16: Examples of surrogate loss functions. (The black segments represent the 0-1 loss as a reference)

Quality of surrogate loss The ERM estimator $\widehat{f}$ can be defined by pointwise optimisation, for a fixed observation $X = x$, $\widehat{f}_\phi(x)$ is the minimum of the following minimisation problem

$$H_\phi(x) := \inf_{\alpha \in \mathbb{R}} \{\eta(x)\phi(\alpha) + (1 - \eta(x))\phi(-\alpha)\}$$

On the other hand, a bad discriminant function f, not following the class probability $\eta(x) = P(Y = 1 \mid X = x)$, satisfies $f(x)(\eta(x) - 1/2) < 0$. When the loss is minimised over bad discriminant functions, define

$$H_\phi(x)^- := \inf_{\alpha\,:\, f(x)(\eta(x)-1/2)<0} \{\eta(x)\phi(\alpha) + (1 - \eta(x))\phi(-\alpha)\}$$

a good surrogate loss function should satisfies $H_\phi(x)^- > H_\phi(x)$ for all x with $\eta(x) \neq 1/2$, such loss function is *classification-calibrated*Jordan (2009). For example, the hinge loss and logit loss are classification calibrated (proof left as exercise), but the perceptron loss is not classification-calibrated.

7.4 Generative Classifiers

Recall that the generative plug-in approach models the full joint distribution X, Y. The joint distribution can be broken down by specifying a marginal model for Y, and then the conditional distribution $X|Y$.

Setup of generative classifier

- specify the prior class probabilities $\pi_k := P(Y = k)$
- specify the conditional PDF/PMF of $X|Y = k$, denoted as g_k

The posterior class probabilities are given by

$$\begin{aligned} P(Y=k|X=x) &= \frac{P(Y=k,X=x)}{P(X=x)} \\ &= \frac{P(Y=k,X=x)}{\sum_{i=1}^{K} P(x,Y=i)} \quad \text{by law of total probability} \\ &= \frac{P(x|Y=k)P(Y=k)}{\sum_{i=1}^{K} P(x|Y=i)P(Y=i)} \\ &= \frac{g_k(x)\pi_k}{\sum_{i=1}^{K} g_i(x)\pi_i} \propto g_k(x)\pi_k \quad \text{because the denominator is independent of } k \end{aligned}$$

Therefore, the Bayes prediction rule is

$$h^*(x) = \arg\max_k P(Y=k|X=x) = \arg\max_k g_k(x)\pi_k$$

Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1,\cdots,n}$, the prior preferences π_k can be simply estimated by the proportion of data within class k,

$$\widehat{\pi}_k = \frac{m_k}{n}$$

where $m_k = m_k(y) := |\{i : y_i = k\}|$. Indeed, this is the maximum likelihood estimator

Proposition 7.6 $\widehat{\pi}_k$ are the MLEs of $\pi_k = P(Y=k)$.

Proof. Y is a multinomial distribution on $\mathcal{Y} = \{1, \cdots, K\}$ with probabilities π_k, so

$$L(\pi|y) = f_\pi(y) = \prod_{k=1}^{K} \pi_k^{m_k(y)}$$

the log-likelihood is easier to maximise, which is

$$l(\pi|y) = \sum_{k=1}^{K} m_k \log(\pi_k)$$

However, the maximisation should be performed with constraint $\sum_k \pi_k = 1$. (because they represent probabilities of classes) The Lagrangian can be used instead,

$$\mathcal{L}(\pi, \gamma) = \sum_{k=1}^{K} m_k \log(\pi_k) - \gamma \left(\sum_{k=1}^{K} \pi_k - 1 \right)$$

solving $\partial\mathcal{L}/\partial\pi$ yields $\pi_k = m_k/\gamma$, and by the constraint, $\sum_k m_k/\gamma = n/\gamma = 1$. So $\gamma = 1$ and the MLE of π_k is indeed in the form provided.
There is a fantastic video on YouTube explaining intuitively why the Lagrangian approach works Lagrange Multipliers — Geometric Meaning and Full Example □

The most critical aspect is defining the conditional PDF/PMF g_k. This involves specifying the structure of features x within a given class k. In the scenario where the feature space $\mathcal{X}$ is continuous and classes exhibit a spherical and more concentrated structure at the centre, the Gaussian distribution (multivariate normal) is often a suitable choice.

7.4.1 Gaussian class distributions

The conditional distribution $X|Y=k$ is multi-variate normal $N(\mu_k, \Sigma_k)$ with mean $\mu_k \in \mathbb{R}^p$ and covariance matrix $\Sigma_k \in \mathbb{R}^{p\times p}$ throughout this section.

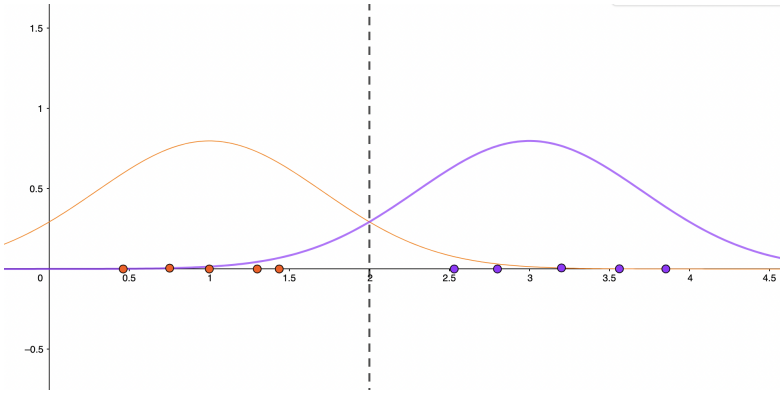


Figure 17: A simple one-dimensional example of linear discriminant analysis with Gaussian distribution

Example 7.7 The concept is easier to illustrate for one-dimensional features x. In Figure 17, $X|Y=0 \sim N(1,1/2)$ and $X|Y=1 \sim N(3,1/2)$. The corresponding PDFs are plotted in two colours. The data points x_i are marked on the x-axis, and class labels y_i are indicated by the colours. In this case, $\widehat{\pi}_1 = \widehat{\pi}_0 = 1/2$ because the two classes have an equal number of data points. The decision boundary is at $x=2$, so if $x<2$, it is labelled as class 0. In practice, the true parameters of conditional distributions $X|Y=k$ are not known, so μ_0, μ_1, Σ_0, Σ_1 have to be estimated from the dataset. For example, the MLE estimators can be used.

Remark 7.8 Both the difference in class means $|\mu_k - \mu|$ (where $\mu := E(X)$) and the covariance matrices Σ_0, Σ_1 contribute to the variation in X. By the law of total variance,

$$\mathrm{Cov}(X) = E_Y[\mathrm{Cov}_X(X|Y=k)] + \mathrm{Cov}_Y(E_X[X|Y=k]) = E_Y[\Sigma_k] + \mathrm{Cov}_Y(\mu_k)$$

where the first term corresponds to the average with-in-class variation, and the second term corresponds to the variance between cluster means. Since Y is a discrete multinomial variable, the two terms are

$$W := E_Y[\Sigma_k] = \sum_{k=1}^{K} \pi_k \Sigma_k$$

$$B := \mathrm{Cov}_Y(\mu_k) = \sum_{k=1}^{K} \pi_k (\mu_k - \mu)(\mu_k - \mu)^T$$

In the one-dimensional setting, the ratio $B/W \in \mathbb{R}^+$ is called *separability* between classes.

In the example above, $W = 1/2 \times 1/2 + 1/2 \times 1/2 = 1/2$ and $B = (3-2)^2/2 + (1-2)^2/2 = 1$. So the separability is 2. If the class means are closer together, say $\mu_0 = 3/2$, $\mu_1 = 5/2$, then $B = 1/4$ and separability becomes $1/2$. The classes are more difficult to distinguish.

If $\mathcal{X} \subseteq \mathbb{R}^2$, the PDFs of two Gaussian distributions are depicted as elliptical grey contour lines in Figure 18. A slight correlation exists between the two variables X_1 and X_2 in both classes, and this correlation is captured by the covariance matrices Σ_1 and Σ_2. The decision boundary is where the two Gaussian PDFs are equal, marked by the dotted curve, which goes through the points where equal contour lines meet. The Gaussian distribution is considered *light-tailed*, leading to a rapid decrease in the likelihood of points falling in the larger contour ellipses.

Higher dimensions of feature space $\mathcal{X}$ are impossible to plot, but the concept is similar. In general, the pdf of p-dimensional multivariate Gaussian distribution $N(\mu_k, \Sigma_k)$ is

$$g_k(x) := \phi(x\,;\,\mu_k, \Sigma_k) = (2\pi)^{-p/2} |\Sigma_k|^{-1/2} \exp\left\{ -\frac{1}{2}(x-\mu)^T \Sigma_k^{-1} (x-\mu) \right\}$$

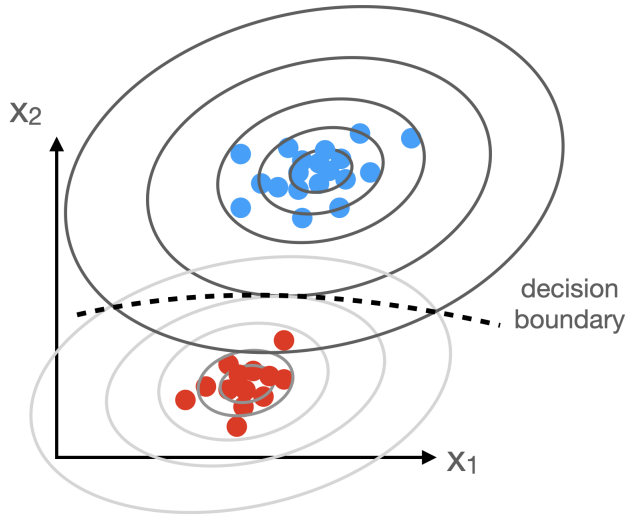


Figure 18: Illustration of Gaussian discrimination analysis with two-dimensional data.

It is easier to define the discriminant function as the log-likelihood, indeed, by the monotonicity of the log function,

$$h^*(x) = \arg\max_k P(Y=k|X=x) = \arg\max_k \pi_k g_k(x)$$
$$= \arg\max_k \log(\pi_k g_k(x))$$

Note that any constant independent of k can be ignored. Plugging the pdf of Gaussian yields

$$f_k(x) := \log(\pi_k g_k(x)) = \log(\pi_k) + \log(\phi(x\,;\,\mu_k,\Sigma_k))$$
$$= \underbrace{\log(\pi_k) - \frac{1}{2}\left(\log|\Sigma_k| + \mu_k^T\Sigma_k^{-1}\mu_k\right)}_{=:a_k} + \underbrace{\mu_k^T\Sigma_k^{-1}}_{=:b_k} x + x^T \underbrace{\left(-\frac{1}{2}\Sigma_k^{-1}\right)}_{=:c_k} x \quad \text{some constants independent of } k \text{ have been dropped}$$

which is a quadratic function. Therefore, Gaussian Discriminant Analysis (with different class covariance) is also known as *Quadratic Discriminant Analysis* (QDA). However, if the assumption of equal class covariance, $\Sigma_k \equiv \Sigma$ for all k, is made (referred to as *tied covariance*), then the quadratic term $-x^T\Sigma_k^{-1}x/2$ becomes independent of k. Consequently, the discriminant functions become linear. This analysis is then termed *Linear Discriminant Analysis* (LDA). In the case with only two classes, it involves finding a separating hyperplane. Removing all terms independent of k results in

$$f_k(x) := \left(\log(\pi_k) - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k\right) + \mu_k^T\Sigma^{-1}x \tag{7.4}$$

The log-likelihood of all parameters $\pi = \{\pi_k\}_{k=1,\cdots,K}$, $\mu = \{\mu_k\}_{k=1,\cdots,K}$, $\Sigma = \{\Sigma_k\}_{k=1,\cdots,K}$ is given below.

$$l(\pi,\mu,\Sigma|\mathcal{D}) = \log\left(\sum_{i=1}^n \pi_{y_i}\phi(x_i\,;\,\mu_k,\Sigma_k)\right)$$
$$= \sum_{k=1}^K \left(m_k(y)\log(\pi_k) + \sum_{i\,|\,y_i=k}\log\phi(x_i\,;\,\mu_k,\Sigma_k)\right) \quad \text{terms are collected by classes}$$
$$= \sum_{k=1}^K m_k(y)\log(\pi_k) - \frac{1}{2}\sum_{k=1}^K\sum_{i\,|\,y_i=k}(x_i-\mu_k)^T\Sigma_k^{-1}(x_i-\mu_k) - \sum_{k=1}^K\frac{m_k}{2}\log|\Sigma_k| + \text{constants}$$

Proposition 7.9 The MLE estimators of the Gaussian generative setting are

$$\widehat{\pi}_k = \frac{m_k(y)}{n}$$

$$\widehat{\mu}_k = \frac{1}{m_k(y)} \sum_{i \mid y_i = k} x_i$$

$$\widehat{\Sigma}_k = \frac{1}{m_k(y)} \sum_{i \mid y_i = k} (x_i - \widehat{\mu}_k)(x_i - \widehat{\mu}_k)^T$$

Note that the MLE estimators of class mean and covariance and exactly the sample mean and covariance in class k.

The general form of the Bayes prediction rule, as given in equation 7.1, involves plugging the Maximum Likelihood Estimators (MLE) into the discriminant function and taking the pointwise maximum to obtain the desired prediction rule $h^{(\mathcal{D})}$. However, a notable drawback associated with this approach is its computational cost, especially in high dimensions. The process involves numerous matrix and vector multiplications for finding the MLE estimators and conditional PDFs. Hence, a dimension reduction approach, such as projection, becomes desirable.

7.4.2 Fisher's LDA

Principal Component Analysis (PCA), discussed earlier, is a dimensional reduction technique. However, its primary focus is on capturing directions with the largest variation rather than maximizing cluster separation. Revisiting the example in Figure 18, the first principal component and the corresponding projection are shown in Figure 19. There is an overlap region between the two classes after the projection, indicating unsatisfactory separability. Additionally, PCA is an unsupervised learning technique that disregards the class labels y_i. Therefore, PCA is not suitable in this case.

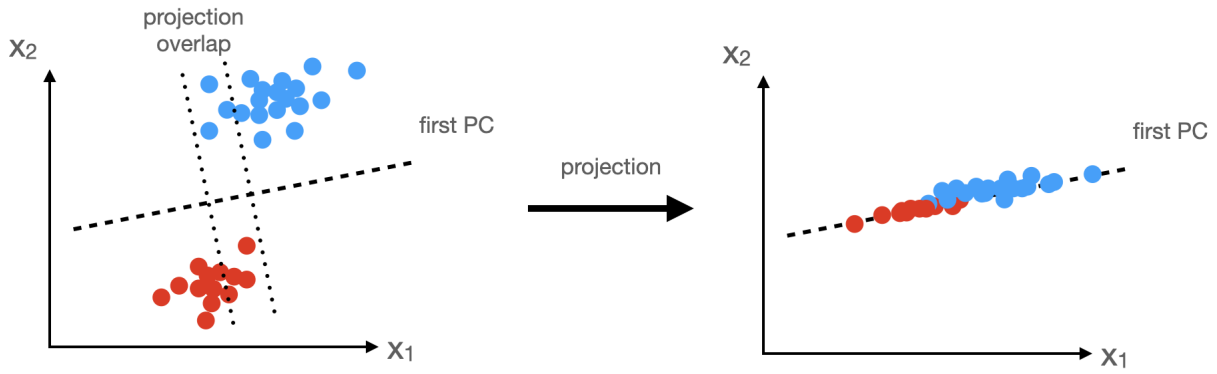


Figure 19: using PCA to reduce the dimension for LDA, ending up with non-separable projections

However, we can borrow the idea of PCA in the sense that the projected direction should maximise the separability between classes, as illustrated in Figure 20.

Let's consider the simplest case where $\Sigma = I$ (indicating independent features with unit variance) and two classes (i.e., $\mathcal{Y} = 1, 2$). In this course, we only study this projection for the tied covariance (i.e. $\Sigma_k \equiv \Sigma$ for all classes k). The discriminant functions can be packed as a single function $\eta(x) := f_2(x) - f_1(x)$ as discussed before. The corresponding prediction rule should be

$$h(x) = \begin{cases} 1, & \text{if } \eta(x) < 0 \\ 2, & \text{if } \eta(x) \geq 0 \end{cases}$$

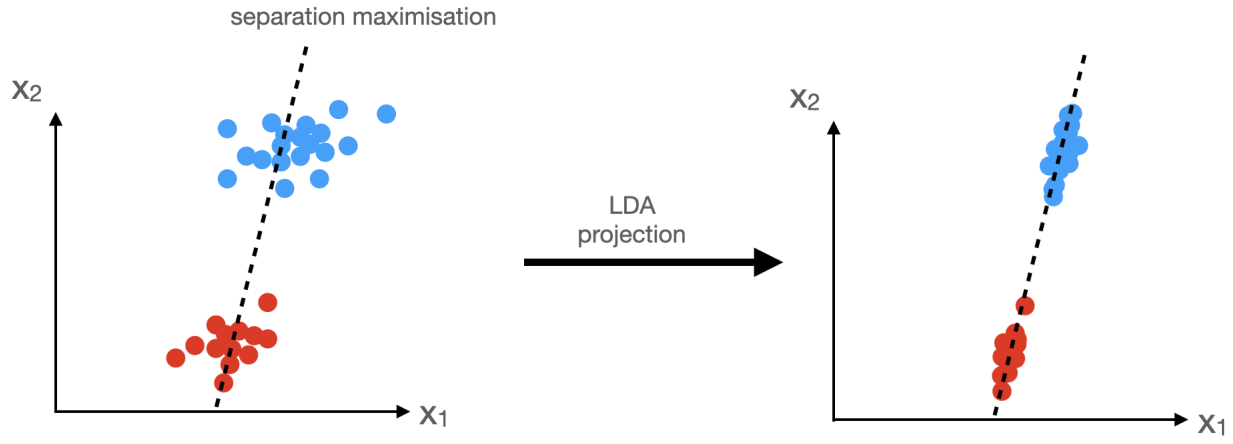


Figure 20: using Fisher's approach to reduce the dimension for LDA, ending up with non-separable projections

Plugging the LDA discriminant functions from Equation7.4 into the η function yields

$$\eta(x) = \log\left(\frac{\pi_2}{\pi_1}\right) - \frac{1}{2}(\mu_2 - \mu_1)^T(\mu_2 + \mu_1) + (\mu_2 - \mu_1)^T x = (\mu_2 - \mu_1)^T(x - x_0)$$

where the point

$$x_0 := \frac{1}{2}(\mu_1 + \mu_2) - (\mu_2 - \mu_1)\frac{\log(\pi_2/\pi_1)}{(\mu_2 - \mu_1)^T(\mu_2 - \mu_1)}$$

lies on the line through μ_1, μ_2 and acts as a decision boundary. As illustrated in Figure 21, if the projection of $x - x_0$ onto the line is above x_0 (closer to class 2), then x is assigned to class 2. x_0 is closer to μ_1 in this case because $\pi_2 > \pi_1$ (giving the class with more points more room).

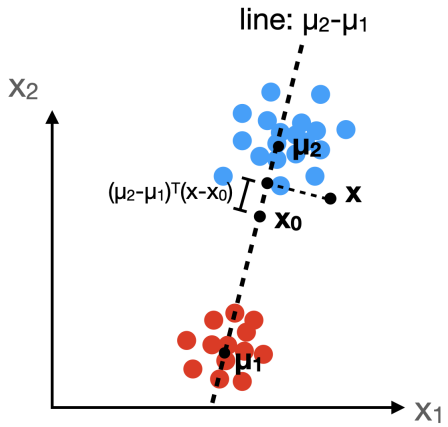


Figure 21: using the projection onto the line joining μ_2 and μ_1 as a classifier

Extension 1: non-unit covariance

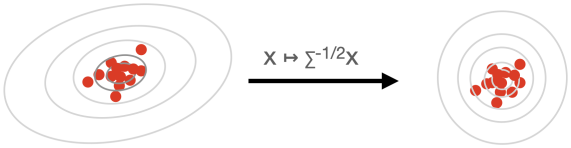


Figure 22: Standardisation for LDA

Dropping the assumption $\Sigma = I$ for now, the key idea is to remove all correlations between features and normalize their variances to 1. (illustrated in Figure 22) If $\Sigma = \text{diag}(\sigma_1^2, \cdots, \sigma_p^2)$, where σ_j^2 represents the variance of the j-th feature, the transformed data $\Sigma^{-1/2}x = (x_j/\sigma_j)$ achieves unit variance. This transformation is valid in general.

Proposition 7.10 Consider the transformation $x \mapsto \Sigma^{-1/2}x =: x^\bullet$, the covariance matrix of $X^\bullet = I$

Proof.

$$\begin{aligned}\text{Cov}(\Sigma^{-1/2}X) &= \Sigma^{-1/2}\text{Cov}(X)(\Sigma^{-1/2})^T \\ &= \Sigma^{-1/2}\Sigma\Sigma^{-1/2} \quad \text{by symmetry of } \Sigma, \Sigma^{-1/2} \text{ is also symmetric}\end{aligned}$$

□

Remark 7.11 Suppose the covariance matrix Σ(which is positive semi-definite) has full rank, it has Eigen-decomposition $\Sigma = V\Lambda V^T$ where Λ is diagonal and V is an orthogonal matrix. The *square root* of matrix $\Sigma^{-1} = V\Lambda^{-1}V^T$, denoted $\Sigma^{-1/2}$, is defined as

$$\Sigma^{-1/2} := V\Lambda^{-1/2}V^T$$

where the square root of the diagonal matrix Λ^{-1} is obtained by simply taking the square root of all its diagonal entries. With this definition, indeed $\Sigma^{-1/2}\Sigma^{-1/2} = V\Lambda^{-1/2}V^TV\Lambda^{-1/2}V^T = V\Lambda^{-1/2}\Lambda^{-1/2}V^T = V\Lambda^{-1}V^T$

It can be shown that with covariance matrix Σ, the discriminant function of a two-class problem can be simplified to

$$\eta(x) = (\mu_2^\bullet - \mu_1^\bullet)(x^\bullet - x_0^\bullet)$$

where $\mu_i^\bullet := \Sigma^{-1/2}\mu_i$. This gets us back to the simplest setting.

Extension 2: multiple classes

With $K > 2$, a similar approach can be employed. As illustrated in Figure 23, projecting x onto the line passing through μ_1 and μ_2 suggests that μ_2 is closer to x. Further projecting onto the line passing through μ_2 and μ_3 still favours μ_2, leading to the assignment of label 2 for x. The decision boundaries, acting as x_0 did previously, are denoted by red dotted lines. It might seem like we have three pairs and thus three lines $\mu_3 - \mu_1$, $\mu_2 - \mu_1$, and $\mu_2 - \mu_3$ to project on. However, the three lines lie on the same 2-dimensional plane.

Projecting the two-dimensional $\mathcal{X}$ in Figure 23 onto another two-dimensional plane spanned by the class means μ_k may not sound interesting. However, even if the dimension of $\mathcal{X}$ is increased to very large values, two projections are sufficient for classification.

More generally, K classes only require projection to $K-1$-dimensional hyperplane H_{K-1} spanned by the class

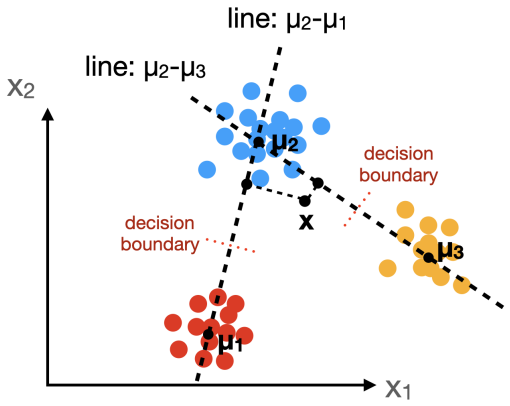


Figure 23: LDA analysis with 2-dimensions and 3 classes

means μ_k. By plugging in equation 7.4 into K-class classifier,

$$\begin{aligned}
h^*(x) &= \arg\max_k f_k(x) \\
&= \arg\max_k \log(\pi_k) - \frac{1}{2}(\mu_k^\bullet)^T\mu_k^\bullet + (\mu_k^\bullet)^T x^\bullet \quad \text{used the projection given in extension 1} \\
&= \arg\max_k \log(\pi_k) - \frac{1}{2}(\mu_k^\bullet)^T\mu_k^\bullet + (\mu_k^\bullet)^T x^\bullet - (\mu^\bullet)^T x \quad \text{where } \mu^\bullet := \sum_k \pi_k \mu_k^\bullet \\
&= \arg\max_k (\mu_k^\bullet - \mu^\bullet)^T \left(\log(\pi_k) - \frac{1}{2}(\mu_k^\bullet)^T\mu_k^\bullet\right) \frac{\mu_k^\bullet - \mu^\bullet}{\|\mu_k^\bullet - \mu^\bullet\|^2} + (\mu_k^\bullet - \mu^\bullet)^T x^\bullet \\
&= \arg\max_k (\mu_k^\bullet - \mu^\bullet)^T (x^\bullet - d_k^\bullet)
\end{aligned}$$

Here, we are projecting onto $\mu_k^\bullet - \mu^\bullet$ instead, which is also on the hyperplane H_{K-1}. The vector

$$d_k^\bullet := \frac{(\mu_k^\bullet)^T\mu_k^\bullet/2 - \log(\pi_k)}{\|\mu_k^\bullet - \mu^\bullet\|^2}(\mu_k^\bullet - \mu^\bullet)$$

lies on the line $\mu_k^\bullet - \mu^\bullet$, and encodes preference towards class k, which plays the role of x_0 as before. The class with the largest value after projection is chosen. Hence, the classifier h^* only uses the information on x from $(\mu_k^\bullet - \mu^\bullet)^T x^\bullet$, $k = 1, \cdots, K$. It can be shown that the projected coordinates onto H_{k-1} is indeed enough for the LDA prediction rule.

Proposition 7.12 The Bayes rule for the LDA classification problem only requires the (orthogonal) projection given by $x \mapsto \overline{U}^T x$ where $\overline{U}$ consists of the first $K-1$ Eigenvectors of The between-class covariance matrix B. And in the case where $\pi_k \equiv 1/K$(equal weights), this is equivalent to the rule which assigns projected x to the closest projected centroid(class mean) on the hyperplane H_{k-1}.

Proof. Recall that the between-class covariance matrix (for the standardised class means $\mu_k^\bullet$) is

$$B = \mathrm{Cov}(E[X^\bullet|Y]) = \sum_{k=1}^K \pi_k(\mu_k^\bullet - \mu^\bullet)(\mu_k^\bullet - \mu^\bullet)^T$$

this matrix has rank at most $K-1$ because $\{\mu_k^\bullet - \mu^\bullet\}_{k=1,\cdots,K-1} \subseteq H_{k-1}$, and note each summand corresponds to a projection matrix onto the line joining $\mu_k^\bullet$, $\mu^\bullet$ because for any vector $v \in \mathcal{X}$,

$$(\mu_k^\bullet - \mu^\bullet)(\mu_k^\bullet - \mu^\bullet)^T v = \left((\mu_k^\bullet - \mu^\bullet)^T v\right)(\mu_k^\bullet - \mu^\bullet) \in \mathrm{Span}(\mu_k^\bullet - \mu^\bullet)$$

Hence, $\mathrm{Im}(B) = H_{K-1}$, meaning that B serves as a projection onto the hyperplane H_{K-1}. Nevertheless, the set $\{\mu_k^\bullet - \mu^\bullet\}$ does not form an orthogonal basis for H_{K-1}. To address this, we turn to the Eigendecomposition $B = UDU^T$, where $U \in \mathbb{R}^{p\times p}$ is an orthogonal matrix, and $D = \mathrm{diag}(\lambda_1, \cdots, \lambda_p)$ is a diagonal matrix containing the eigenvalues in non-decreasing order. Notably, only the first $K-1$ eigenvalues are non-zero. Consequently, the decomposition can be simplified to

$$B = \overline{U}\,\overline{D}\,\overline{U}^T$$

where $\overline{D} = \mathrm{diag}(\lambda_1, \cdots, \lambda_{K-1})$ and $\overline{U}$ takes the first $K-1$ columns of matrix U. Now, $\{u_1, \cdots, u_{K-1}\}$ (the first $K-1$ eigenvectors) forms an orthonormal basis for H_{K-1}, and we can establish an orthogonal projection $x \mapsto \overline{U}^T x$. The coordinates of the projection are defined as $z_l := u_l^T x^\bullet$ (referred to as *discriminant coordinates*). Meanwhile, the projected point is expressed as

$$\tilde{x} = \sum_{l=1}^{K-1} (u_l^T x^\bullet) u_l \in H_{K-1}$$

We now establish that when $\pi_k \equiv 1/K$, following the projection $x^\bullet \mapsto \overline{U}^T x^\bullet$, the Bayes prediction rule is equivalent to identifying the nearest centroid within the subspace $\overline{U}^T\mathbb{R}^p = H_{K-1}$.

$$\begin{aligned}
h^*(x) &= \arg\max_k \ \log(\pi_k) - \frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k) \\
&= \arg\min_k \ \frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k) = (x^\bullet - \mu_k^\bullet)^T(x^\bullet - \mu_k^\bullet) \quad \text{by assumption, } \pi_k \text{ is independent of } k \\
&= \arg\min_k \|x^\bullet - \mu_k^\bullet\|^2 = \arg\min_k \ \left\|\overline{U}^T x^\bullet - \overline{U}^T\mu_k^\bullet\right\|^2 \quad \text{because } \overline{U} \text{ has orthonormal columns} \\
&= \arg\min_k \ \left\|\overline{U}^T x^\bullet - \mu_k^\bullet\right\|^2 \quad \text{because } \mu_k^\bullet \in H_{k-1}
\end{aligned}$$

when π is not uniform, there is an additional term $-\log(\pi_k)$, which encodes the prior preference. □

Connection with PCA

It can be proved that the first Eigenvector u_1 maximises the separability. Indeed,

$$\begin{aligned}
\arg\max_{a\in\mathbb{R}^p} \frac{\mathrm{Cov}(E(a^TX|Y))}{E[\mathrm{Cov}(a^TX|Y)]} &= \arg\max_{a\in\mathbb{R}^p} \frac{a^TBa}{a^T\Sigma a} \\
&= \arg\max_{u\in\mathbb{R}^p} \frac{u^TB^\bullet u}{u^Tu} \quad \text{set } u = \Sigma^{-1/2}a \\
&= \arg\max_{u\in\mathbb{R}^p,\,\|u\|=1} u^TB^\bullet u
\end{aligned}$$

this is the familiar PCA problem formulation. By Proposition 3.1, the first Eigenvector u_1 of $B^\bullet$ is the solution. Similarly, u_2 maximises separability among the space $\{u \in \mathbb{R}^p : u_1^Tu = 0,\ \|u\| = 1\}$. Each u_l has a similar interpretation. See more technical details in Duda et al. (2001).

Algorithm 2 QDA Gaussian discriminant analysis without dimension reduction

1: Input: data points $\mathcal{D} = \{(x_i, y_i)\}$
2: Compute the MLE estimators $\widehat{\pi}_k$, $\widehat{\mu}_k$, $\widehat{\Sigma}_k$ as given in Proposition 7.9
3: Find the Eigendecomposition of $\widehat{\Sigma}_k = V_k\Lambda_kV_k^T$, and compute $\widehat{\Sigma}_k^{-1/2} =: \Lambda_k^{-1/2}V_k^T$
4: define transformation $T(x_i) := \widehat{\Sigma}_{y_i}^{-1/2}x_i$, transform $\widehat{\mu}_k \mapsto \widehat{\Sigma}_k^{-1/2}\widehat{\mu}_k =: \widehat{\mu}_k^\bullet$
5: **return** the function

$$h^{(d)}(x) = \arg\min_{k=1,\cdots,K} \|T(x) - \widehat{\mu}_k^\bullet\|^2 - 2\log(\widehat{\pi}_k) - x^T\widehat{\Sigma}_k^{-1}x + \log|\widehat{\Sigma}_k|$$

In the case of untied covariance, the returned function is

$$h^{(d)}(x) = \arg\min_{k=1,\cdots,K} \|T(x) - \widehat{\mu}_k^\bullet\|^2 - 2\log(\widehat{\pi}_k)$$

Algorithm 3 LDA: Gaussian discriminant analysis with dimension reduction and tied covariance

1: Input: data points $\mathcal{D} = \{(x_i, y_i)\}$
2: Compute the MLE estimators $\widehat{\pi}_k$, $\widehat{\mu}_k$ as given in Proposition 7.9, the MLE of Σ in this case is

$$\widehat{\Sigma} = \frac{1}{n} \sum_{k=1}^{K} \sum_{i \mid y_i = k} (x_i - \widehat{\mu}_k)(x_i - \widehat{\mu}_k)^T$$

3: Find the Eigendecomposition of $\widehat{\Sigma} = V \Lambda V^T$, and compute $\widehat{\mu}_k^{\bullet} = \Lambda^{-1/2} V^T \widehat{\mu}_k$, $x^{\bullet} = \Lambda^{-1/2} V^T x$
4: obtain the first $K-1$ Eigenvectors $\{u_l\}_{l=1,\cdots,K-1}$ of matrix $\widehat{B}^{\bullet}$, and define transformation $T(x) := U^T x^{\bullet}$ where columns of U are u_l.
5: **return** the function

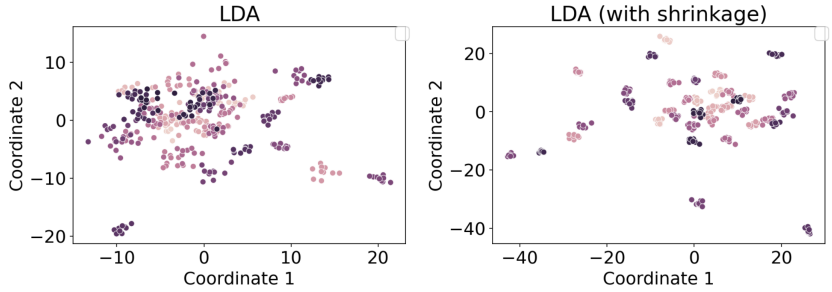
$$h^{(d)}(x) = \arg\min_{k=1,\cdots,K} \|T(x) - \widehat{\mu}_k^{\bullet}\|^2 - 2\log(\widehat{\pi}_k)$$

If you execute only steps 1-4 of Algorithm 3, the resulting transformation T can serve as a tool for dimension reduction. This transformation succinctly captures the information in $\mathcal{D}$ essential for classification purposes.

Regularised Discriminant Analysis LDA offers the advantage of lower computational cost, while QDA provides greater flexibility in the model. These two models can be combined by replacing the estimator for Σ_k in Algorithm 2 with the following estimator:

$$\widehat{\Sigma}_k(\alpha) = \alpha \widehat{\Sigma}_k + (1 - \alpha)\widehat{\Sigma}$$

where the hyperparameter $\alpha \in [0, 1]$ could be found by cross-validation.



(a) LDA (b) LDA with shrinkage

Figure 24: taken from Lecture notes (Figure 2.5) illustrating the effect of shrinkage

LDA with shrinkage In order to improve the separability of clusters after dimensional reduction in Algorithm 3, the estimator of Σ in step 2 could be replaced by the shrinkage estimator

$$\widehat{\Sigma}(\delta) = \delta I_p + (1 - \delta)\widehat{\Sigma} \quad \delta \in (0, 1)$$

which pulls the variance in each dimension closer to 1. See an example of shrinkage in Figure 24.

7.4.3 Naive Bayes Methods

If the features are binary or categorical variables rather than real variables, i.e. $\mathcal{X} = \{1, \cdots, C\}$ where C is the number of categories, the multinomial distribution is appropriate. Specifically, assume $X|Y = k \sim \text{Multi}(\theta_{1k}, \cdots, \theta_{Ck})$. Each parameter θ_{ik} represents the probability of belonging to category i in class k, with MLE simply given by proportion of data points in category i within class k:

$$\widehat{\theta}_{kj,c} = \frac{\sum_{i|y_i=k} 1_{x_{ij}=c}}{m_k} = \frac{|\{(i,j) \,:\, y_i = k, x_{ij} = c\}|}{|\{i \,:\, y_i = k\}|}$$

However, when each feature has c categories, there are C^p combinations of categories, necessitating $O(C^p)$ parameters. This poses a clear risk of over-fitting.

The naive Bayes estimator avoids over-fitting by assuming that different features are mutually independent within class k. The conditional PMF/PDF of class k, denoted as $g_k(x)$, is expressed as

$$g_k(x) = \prod_{i=1}^{p} g_{ki}(x; \theta_{ki}) \tag{7.5}$$

, where g_{ki} is the distribution of $X_i|Y = k$ with parameter θ_{ki}. See pseudo-code of naive Bayes in Algorithm 4

Algorithm 4 Naive Bayes Method

Input: data points $\mathcal{D} = \{(x_i, y_i)\}_{i=1,\cdots,n}$, parametric models $g_{kj}(x_i; \theta_{kj})$ for $k = 1, \cdots, K$, $j = 1, \cdots, p$
for $k = 1, \cdots, K$ **do**
    Compute the MLE estimator

$$\widehat{\pi}_k := \frac{m_k}{n}$$

    for $j = 1, \cdots, p$ **do**
        Compute the MLE estimator for $\widehat{\theta}_{kj}$ for g_{kj} using the non-missing data points

$$\{(x_{ij}, y_i) \,:\, i = \{1, \cdots, n\}, x_{ij} \text{ not NaN}\}$$

    end for
end for
return the function

$$h^{(d)}(x) = \underset{k=1,\cdots,K}{\arg\max} \left\{ \log(\widehat{\pi}_k) + \sum_{j=1}^{p} \log\left(g_{kj}(x_j; \widehat{\theta}_{kj})\right) \right\}$$

Over-fitting By modeling each feature separately, the naive Bayes estimator requires only $O(Cp) = O(p)$ parameters, effectively avoiding over-fitting. However, the strong independence assumption is often unrealistic ("naive"), yet surprisingly, the estimator performs well in many cases with apparent dependencies Domingos and Pazzani (1997).

Mixed Data This assumption proves particularly beneficial for mixed-type datasets containing both continuous and categorical features. It eliminates the need to determine the joint distribution of a categorical variable X_i and a real-valued variable X_j, as each can be independently modeled by g_{ki} and g_{kj}.

Missing Data If in the prediction, feature $\tilde{x}_i$ of new data $\tilde{x}$ is missing, marginalisation is enough for generative classifiers. Computing the following marginal distribution instead

$$g_k(X_j = \tilde{x}_j, j \neq i) = \int_{x_i} \prod_{j=1}^{p} g_{kj}(x_j; \theta_{kj})\, dx_i$$

For Naive Bayes estimators, by equation 7.5,

$$g_k(X_j = \tilde{x}_j, j \neq i) = \prod_{j \neq i} g_{kj}(x_j; \theta_{kj})$$

then

$$P(\tilde{Y} = k \,|\, X_j = \tilde{x}_j, j \neq i) \propto \pi_k \prod_{j \neq i} g_{kj}(x_j; \theta_{kj})$$

by Bayes rule.

During the training stage, handling missing values often involves discarding data entries with incomplete information. Without the independence assumption, missing x_{ij} would result in discarding the entire row x_i. However, for the naive Bayes estimator, which assumes independence between features, this process can be performed for each feature separately, minimising information loss.

7.4.4 Conclusion

Suitable choices of conditional distribution $X|Y = k$ (g_k) for generative classifiers:

- Real variable: use Gaussian (LDA/QDA) or other absolutely continuous distributions
- Categorical variable: use multinomial distribution, or other discrete distributions with finite support.
- Counting variables (support is $\mathbb{N}$): Poisson or other discrete distributions with infinite support.
- mixed-type variables: requires independence assumptions like naive Bayes.

Using generative classifiers has the following advantages

- modelling assumptions can be statistically tested. For example, the independence assumption of naive Bayes methods
- provides ways to handle missing data
- has interpretable predictions.

whereas generative classifiers suffer from a lack of flexibility due to strong modelling assumptions. And they are strongly influenced by outliers because each point plays a role in predicting the parameters of $X|Y = k$ and outliers could have higher leverages.

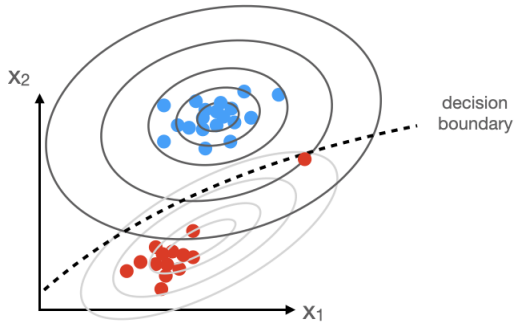


Figure 25: Effect of outlier on the QDA (generative classifiers using Gaussian distributions)

For instance, in the Gaussian case, an outlier pulls the mean towards it and disrupts the covariance, thereby altering the decision boundary (refer to Figure 25). Conversely, linear classifiers like logistic regression and perceptron are primarily affected by data points close to the decision boundary, rendering outliers nearly negligible in their impact.

7.5 Nearest Neighbours

In this section, we introduce a classifier that is model-free, i.e. makes no assumptions about the distributions of X, Y. The idea of nearest-neighbour (NN) methods involves aggregating information from the closest neighbors. For example, given a new data point x, the 1-NN classification finds the nearest neighbor $\text{nn}(x) := \arg\min_{i=1,\cdots,n} \|x - x_i\|^2$ (see an example on the left of Figure 26), and assigns the class label of the nearest neighbor $y_{\text{nn}(x)}$ to x. This classification method induces a partition of the sample space called *Voronoi tessellation* (as shown on the right of Figure 26).

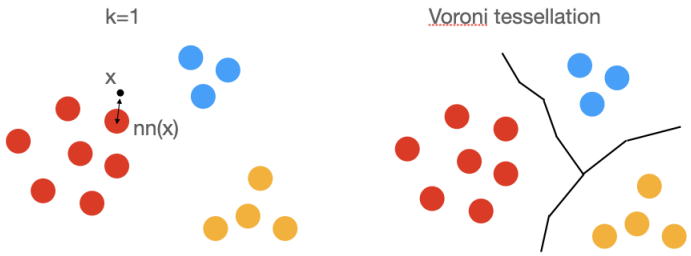


Figure 26: kNN method with $k = 1$. (left) illustration of prediction rule (right) the induced tessellation of the space

However, the process above is strongly affected by the training sample and may over-fit. More generally, the kNN classifier picks the k nearest neighbours (define $\text{knn}(x)$ to be the labels of k nearest neighbours), and approximate the class probability for each class c as

$$P(Y = c \mid X = x) \approx \frac{\sum_{i \in \text{knn}(x)} 1_{y_i = c}}{k}$$

, and one selects c with highest class probability. From this point of view, the KNN method is a non-parametric conditional plug-in method. For example, in Figure 27, the data point x is in the red class with probability $2/3$ and in the blue class with probability $1/3$. The most suitable value of k can be determined by T-fold cross-validation introduced in Section 5.2.

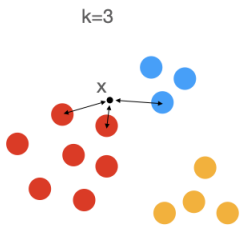


Figure 27: example of kNN classification with $k = 3$.

kNN for regression In fact, the kNN method can be used for regression task, by approximating the conditional CDF $P(Y \leq y \mid X = x)$ by

$$\frac{\sum_{i \in \text{knn}(x)} 1_{y_i \leq y}}{k}$$

The Bayes estimator under this approximation is

$$\widehat{h}(x) = \frac{\sum_{i \in \text{knn}(x)} y_i}{k}$$

Standardisation It is advisable to normalise the data points so that they have a zero sample mean and a standard deviation of 1. This normalisation is recommended because each dimension (feature) may not be on the same scale, but they have equal weights in the Euclidean distance $\|x - x'\|_2^2$.

Distance-metric In the examples above, the distance (dissimilarity) measure is chosen to be the Euclidean distance $\rho(x, x') = \|x - x'\|_2^2$. More generally, the Mahalanobis distance measure is $\rho_M(x, x') := x^T M x'$ for some positive definite matrix M, with the Euclidean distance using $M = I$. The choice of an appropriate distance measure can be learned from data, which is crucial for many machine learning algorithms, including k-means, KNN, and SVM. Refer to Moutafis et al. (2017) for an overview of distance metric learning algorithms.

Drawbacks Despite the simplicity of this algorithm, it has several drawbacks

- The computational cost of the prediction rule is high: each distance measure requires $O(p)$ operations and there are n data points to compare. So the cost for a single prediction is $O(np)$. Further, the prediction rule stores all the training data which takes a lot of space.
- learns on the relationship between attributes/features
- curse of dimensionality: the volume of a hypercube grows exponentially with the dimension, causing the distance measure to grow quickly. For instance, consider distances to the origin. If $x_1 = (1, 1, 1, 1, \cdots, 1) \in \mathbb{R}^{100}$, then $\|x_1 - 0\|_2 = 10$. However, for $x_2 = (10, 0, \cdots, 0)$, the distance $\|x_2 - 0\|_2 = 10$ as well. It is evident that the difference in the first attribute between x_2 and 0 is very large, but x_2 will be treated as a "neighbor" of 0 along with x_1.
- The choice of a distance metric is not straightforward and a poor choice can result in a sub-optimal classifier.

8 Neural Networks

8.1 Introduction and Motivations

Another view on logstic regression The logistic regression introduced in section 7.2 predicts an binary outcome(class label) $y \in \{\pm 1\}$ via estimating the probability $p_1 := P(y = 1|X = x)$, which is modelled by a linear relation through the logit link function:

$$\text{logit}\,(p_1) \approx \boldsymbol{x}^T\beta, \qquad \text{where sigmoid function logit}\,(p_1) = \sigma^{-1}(p) = \log\left(\frac{p}{1-p}\right)$$

the vector β consists of the coefficients on the predictors x_i. The Bayes estimator is defined by

$$h^*(x) = \text{step}_{1/2}(p) := \begin{cases} 1, & \text{if } p > 1/2 \\ -1, & \text{if } p \leq 1/2 \end{cases}$$

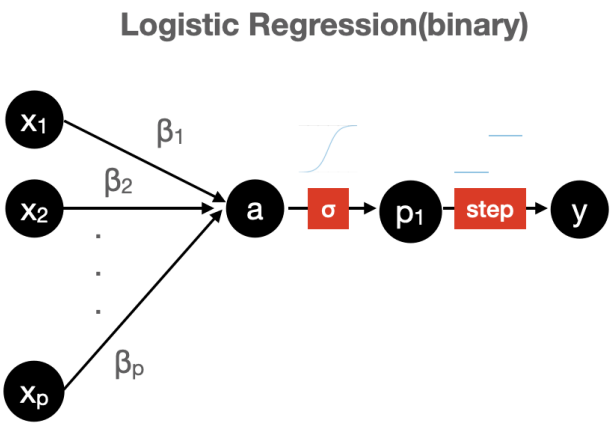


Figure 28: Binary logistic regression represented by a computational graph. The nodes x_i are the ith attribute/dimension, not the ith data point. Each node represents a random variable, not a specific value(realisation).

Suppose the values of β are known. Given new data $x = (x_1, \cdots, x_p)$, the label y is estimated by finding $p_1 = \sigma(\mathbf{x}^T\beta)$, then letting $y = \text{step}_{1/2}(p_1)$. This procedure can be represented as a computational graph (see Figure 28), where each node (circle) represents a variable, and the edges represent operations or parameters. The quantity p_1 is a one-dimensional non-linear feature of the dataset x that suits the binary classification task.

For multi-class logistic regression, the features $p_c := P(Y = c \mid X = x)$ are modeled as

$$p_c = \text{sig}(a_c) = \text{sig}(\beta_c^T x)$$

where the vector $\beta_c = (\beta_{c1}, \cdots, \beta_{cp})^T$ represents the parameters for the effect of each x_i on $a_c := \beta_c^T x$. The activation function for each a_c is σ, this is often the case in neural networks. The output (class label) y is determined by the softmax of all class probabilities p_c. One may alternatively select the value of c with the highest p_c, i.e., $y = h(x) := \arg\max_{c=1,\cdots,C} p_c$. However, this arg-max function is not differentiable. As we will introduce later, neural networks are trained by gradient descent, which requires differentiability for each operation in the network. The computational graph for multi-class logistic regression is shown in Figure 29.

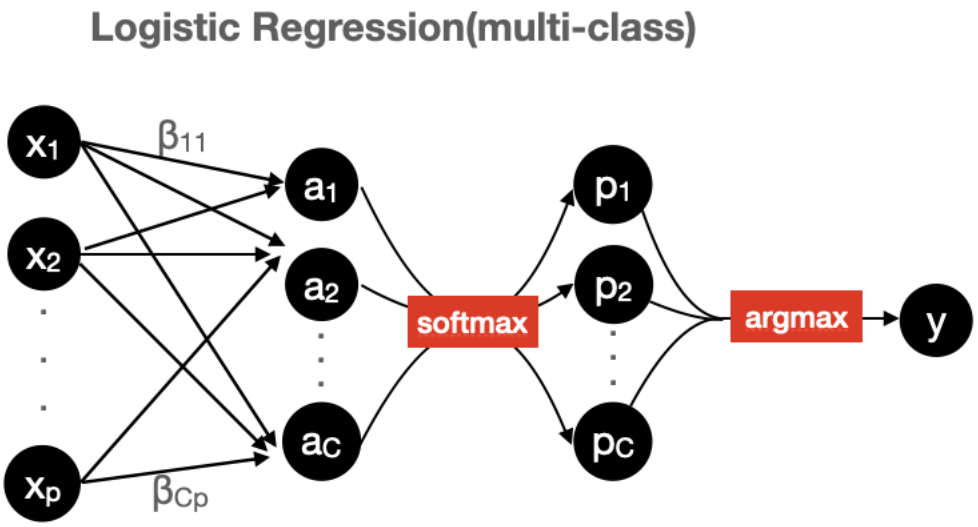


Figure 29: Multi-class Logistic regression represented by a computational graph. The edges between first two layers represent the coefficients/weights β_{ic}.

Single-Layer Perceptron The idea of a neural network is to learn features from data instead of specifying them, as in logistic regression where the specified features are the class probabilities. Specifically, each feature $z_k = \phi_{\theta_k}(x)$ is represented by an adaptive basis function for some parameter θ_k. The number of features, m, and the functional form ϕ of the features are specified. The functions $\{\phi_{\theta_k}\}_{k=1,\cdots,m}$ are termed *adaptive basis functions*.

In neural networks, each feature, represented by a node (called a *hidden unit*) z_k, is specified to be a non-linear activation of some linear combination of x, i.e., $z_k := s(w_k^T x + b_i)$. Here, $w_k = (w_{k1}, w_{k2}, \cdots, w_{kp})^T \in \mathbb{R}^p$ is the weight vector, $b_i \in \mathbb{R}$ is the bias, and $s : \mathbb{R} \to \mathbb{R}$ is a pre-specified differentiable activation function. In logistic regression, the sigmoid function is commonly used. The unknown parameters to be learned are denoted by $\theta_k = (w_{k1}, w_{k2}, \cdots, w_{kp}, b_i)$. This form is chosen for its ease of differentiation, facilitating the training of neural networks (i.e., fitting parameters θ_k) by gradient descent. The appropriate choice of activation function s will be discussed in Section 8.3.

The vector $z = (z_1, \cdots, z_m)$ is referred to as the *hidden layer*. Each feature z_i is flexible and extracts different features for different datasets.

The single-layer perceptron, a computational process as described above, is compactly defined by

$$a := Wx + b \in \mathbb{R}^m \quad \text{linear combination (pre-activation layer)}$$

$$z := (s(a_1), \cdots, s(a_m))^T \in \mathbb{R}^m \quad \text{post-activation (hidden) layer}$$

$$y := s^{(o)}(z) \in \mathbb{R} \quad \text{output layer}$$

where $W = (w_{ki}) \in \mathbb{R}^{m \times p}$ are the weights on x (in logistic regression, the weights are the parameters β_{Ci}), and the vector $b = (b_1, \cdots, b_m) \in \mathbb{R}^m$ are the bias(intercepts) for each variable a_j. The function $s : \mathbb{R} \to \mathbb{R}$ is a pre-specified non-linear activation function, and $s^{(o)} : \mathbb{R}^m \to \mathcal{Y} \subseteq \mathbb{R}$ is the output layer activation function (which is softmax for logistic regression). The computational graph is shown in Figure 30.

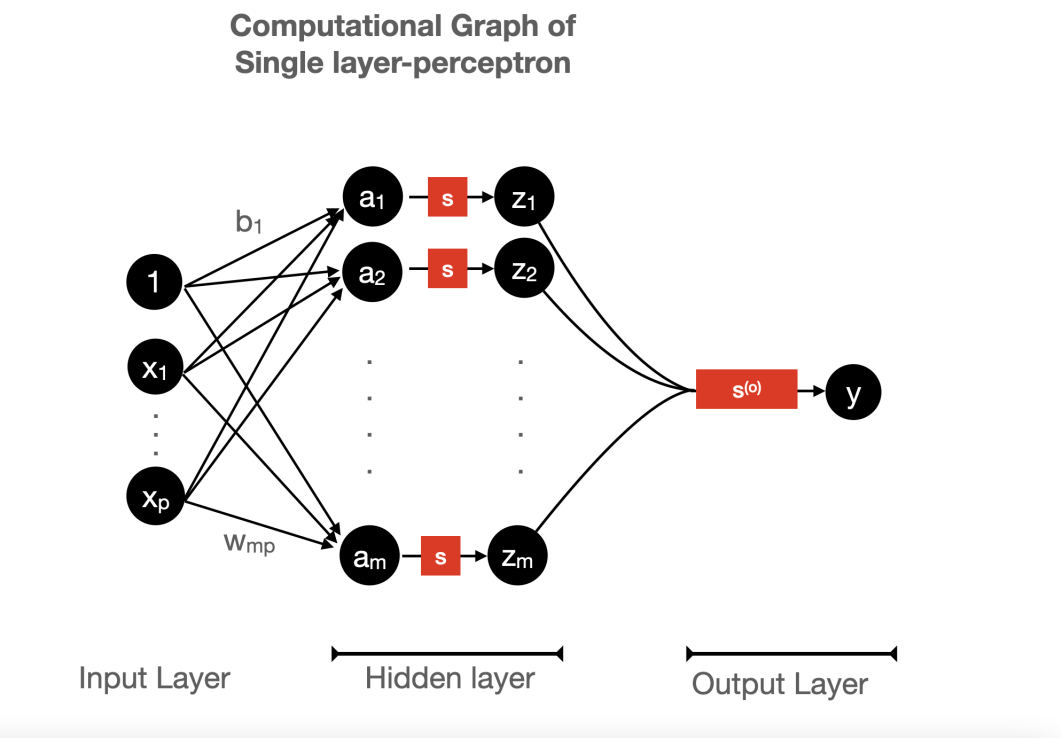


Figure 30: computational graph of a single-layer perceptron, the node 1 is added to facilitate the biases b and $b^{(o)}$

Choice of output activation Suppose only linear predictors based on the features z are considered; then, $y = s^{(o)}(z) := z^T w^{(o)} + b^{(o)}$, where $w^{(o)} \in \mathbb{R}^m$ are the (output layer) weights and $b^{(o)} \in \mathbb{R}$ is the bias. This is often enough for regression task. As for binary classification, the neural network should produce $p_1 = P(Y = 1 \mid X = x) \in [0, 1]$, so a sigmoidal function σ is applied to the linear combination $f := z^T w^{(o)} + b^{(o)}$ as output activation, i.e. $s^{(o)}(z) := \sigma(z^T w^{(o)} + b^{(o)})$.

Definition 8.1 — **Sigmoidal function.** The function $\sigma : \mathbb{R} \to [0, 1]$ is sigmoidal if

$$\lim_{z \to \infty} \sigma(z) = 1, \quad \lim_{z \to -\infty} \sigma(z) = 0$$

In some occasions, the codomain is replaced by $[-1, 1]$ and second limit is replaced by $\lim_{z \to -\infty} \sigma(z) = -1$, but scaling and translation will move functions defined on $[-1, 1]$ to $[0, 1]$ without changing the shape.

an example of sigmoidal function is $\sigma(z) = 1/(1 + e^{-z})$ discussed in logistic regression. The computational graph of the single-layer perceptron for binary classification is illustrated in figure 31.

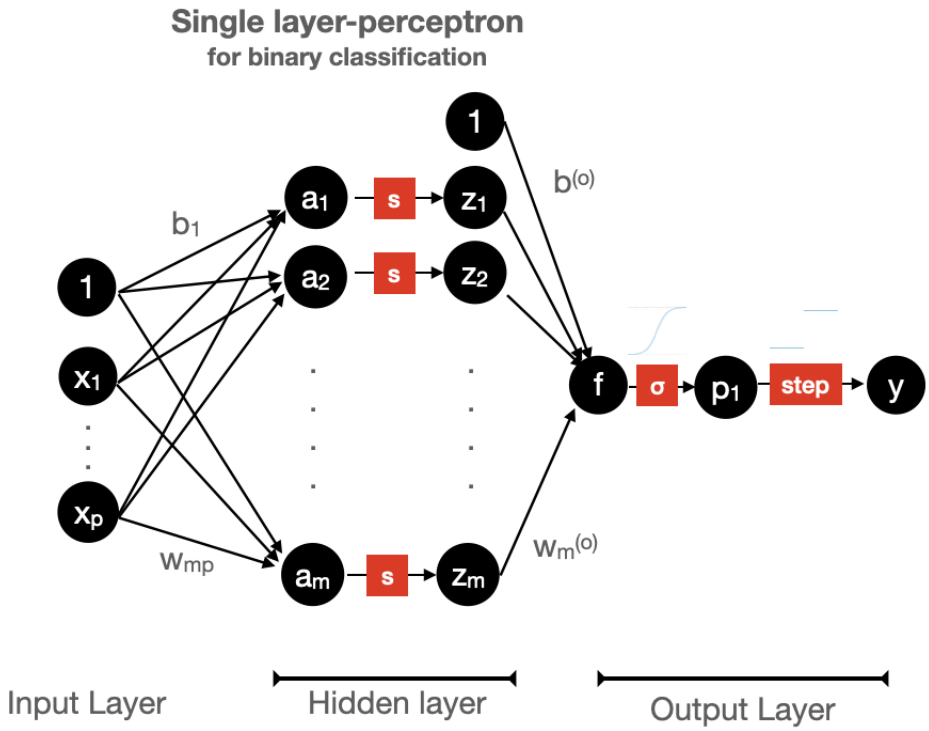


Figure 31: computational graph of a single-layer perceptron for binary classification

Alternatively, a decision boundary τ can be specified for $f(x) = z(x)^T w^{(o)} + b^{(o)}$ (where $z(x) := Wx + b$), and the output (prediction rule) is defined by

$$h(x) := \begin{cases} 1, & \text{if } f(x) \geq \tau \\ 1, & \text{if } f(x) < \tau \end{cases}$$

in section 7.2, we used $\tau = 0$. This is a much simpler output activation.

As for multi-class task $y \in \{1, \cdots, C\}$, the neural network should produce C probabilities $p_c = P(Y = c \mid X = x)$. The hidden layer z is first mapped to $\mathbb{R}^C$ by a linear transformation

$$f = W^{(o)} z + b^{(o)} \quad \text{where } W^{(o)} := (w_{ck}^{(o)}) \in \mathbb{R}^{C \times m},\ b^{(o)} = (b_1^{(o)}, \cdots, b_C^{(o)}) \in \mathbb{R}^C$$

and then the softmax activation can be applied to f, producing $p := \text{softmax}(f)$. The computational graph for multi-class classification is illustrated in figure 32.

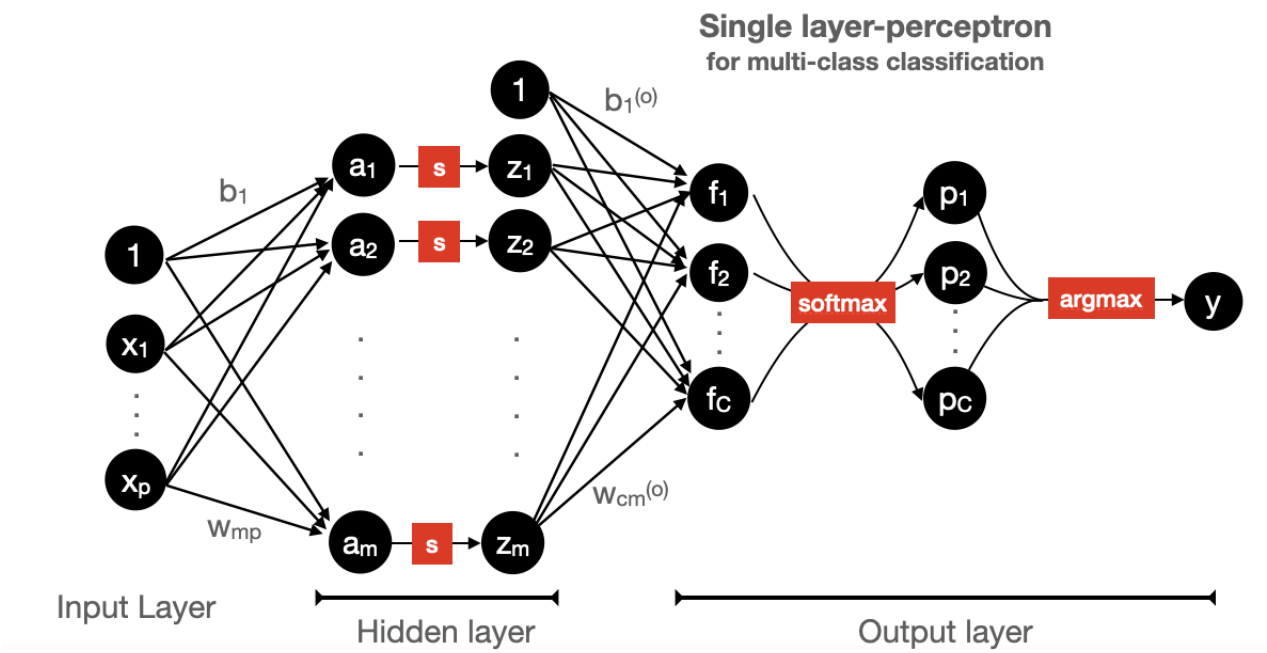


Figure 32: computational graph of a single-layer perceptron for multi-class classification

Over-parametrisation If there are many features (large m), the number of parameters to estimate becomes massive. In the single-layer (one hidden layer) case discussed above, there are mp parameters for the hidden layer and m parameters for the output layer. However, as discussed in Section 5.2, this over-parametrisation can actually reduce the risk of the prediction rule obtained from a neural network. In some cases, the risk may even be lower than in the under-parametrisation scenario (few features).

When there are infinite number of features, it can be shown that the single layer perceptron estimates any continuous function on bounded interval. This is the universal approximation theorem for arbitrary width, first developed by Cybenko (1989).

Theorem 8.2 — **Universal Approximation Theorem(arbitrary width).** Given a continuous discriminatory activation function $s : \mathbb{R} \to \mathbb{R}$ (roughly speaking, s is discriminatory if it does not map an interval to a point), and any continuous function $h^*(x) : I_p \to \mathbb{R}$ defined on the hyper-cube $I_p := [0,1]^p$, for any precision $\epsilon > 0$, there exists $m \in \mathbb{N}$ and weights $W \in \mathbb{R}^{m \times p}, w^{(o)} \in \mathbb{R}^m$ and biases $b \in \mathbb{R}^m, b^{(o)} \in \mathbb{R}$ such that the function $h(x) := \sum_{k=1}^{m} w_k^{(o)} s(w_k^T x + b_k) + b^{(o)}$ satisfies

$$|h(x) - h^*(x)| < \epsilon \; \forall x \in I_p$$

Remark 8.3 For any finite data set, it is reasonable to assume each attribute/dimension is bounded, so that the sample space $\mathcal{X} \subseteq \mathbb{R}^p$ can be normalised to $I_p = [0,1]^p$.

8.2 Network Training

In the previous section, we deliberately ensured that every operation in the single-layer perceptron (SLP) is differentiable. Consequently, the output of the SLP, denoted as $h(x;\theta)$, is differentiable with respect to x and any of the parameters. Here, θ encompasses all parameters involved in the SLP, such as W, b, $w^{(o)}$, and $b^{(o)}$ for an SLP with linear output activation.

When considering the *training* of an SLP, we aim to minimize the empirical risk function:

$$J(\theta) := \frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i;\theta))$$

Here, L is a chosen loss function, and $\{(x_i, y_i)\}_{i=1,\dots,n}$ forms the training set. The training process involves finding the set of parameters θ that minimizes $J(\theta)$ through gradient descent. To facilitate this, the loss function L is selected to ensure that $J(\theta)$ is differentiable in every dimension. For regression tasks where $\mathscr{Y} \subseteq \mathbb{R}$ and Y is a continuous variable, a common choice for the loss function is $L(y, \widehat{y}) = (y - \widehat{y})^2$.

The gradients are calculated by chain rule, which can be written down by tracing along the computational graph (in Figure 30). For example, for the weight w_{kj},

$$\frac{\partial L(y, h(x;\theta))}{\partial w_{kj}} = \frac{\partial L(y, h(x;\theta))}{\partial h(x;\theta)} \frac{\partial h(x;\theta)}{\partial z_k(x)} \frac{\partial z_k(x)}{\partial w_{kj}}$$

because the edge w_{kj} leads to $z_k(x) = s(w_k^T x + b_k)$, and then connects to the output $h(x;\theta)$ via output activation $s^{(o)}$.

Implementation of gradient descent It's important to note that J often exhibits a complex structure, is non-convex, and may have numerous local minima. Hence, the choice of the initial point of gradient descent significantly influences performance. The initialisation of the numerous parameters in neural networks remains a subject of ongoing research. Further, the Stochastic Gradient Descent (SGD) is typically employed when dealing with large, high-dimensional datasets to mitigate computational costs. More implementation details can be found in section 13.4 of Murphy (2022) and chapter 3 and 4 of Dive into deep learning provides many python codes.

Binary Classification The neural network produces a discriminant function $f(x;\theta) = f(x; W, b, w^{(o)}, b^{(o)}) = z(x)^T w^{(o)} + b^{(o)}$ (where $z(x) := Wx + b$) and the prediction rule is $h(x;\theta) = 1$ if $f(x;\theta) \geq 0$, $h(x;\theta) = -1$ otherwise. Recall from Section 7.3 that the loss function for binary classification can all be defined on the quantity $yf(x;\theta)$, but the 0-1 loss $\psi_{0\text{-}1}(z) := 1_{z\geq 0}$ is not differentiable. Therefore, the objective

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} 1_{y_i \neq h(x;\theta)} = \frac{1}{n}\sum_{i=1}^{n} \psi_{0\text{-}1}(y_i f(x;\theta))$$

is not differentiable. The solution given in Section 7.3 is to use a convex differentiable function ψ (surrogate loss function) as a proxy. In this case, minimise

$$\tilde{J}(\theta) := \frac{1}{n}\sum_{i=1}^{n} \psi(y_i f(x_i;\theta))$$

instead, which approximates $J(\theta)$.

If a sigmoid output activation is applied, producing $\eta(x;\theta) := \mathrm{sig}(f(x;\theta))$ that approximates $P(Y = 1 \mid X = x)$ (similar to the setup of logistic regression). When the surrogate loss $\psi(z) = \log(1 + e^{-z})/\log(2)$ is used, the loss can be expressed with respect to η as follows:

$$\begin{aligned}
\psi(yf(x;\theta)) &= \frac{1}{\log(2)} \left(1_{y=1} \log(\mathrm{sig}(f(x;\theta))) + 1_{y=-1} \log(\mathrm{sig}(-f(x;\theta)))\right) \\
&\propto 1_{y=1} \log(\mathrm{sig}(f(x;\theta))) + 1_{y=-1} \log(1 - \mathrm{sig}(f(x;\theta))) \\
&= 1_{y=1} \log(\eta(x;\theta)) + 1_{y=-1} \log(1 - \eta(x;\theta)) =: \tilde{L}(y, \eta(x;\theta))
\end{aligned}$$

where $\tilde{L}(y, \eta(x;\theta))$ is called *log loss*. Therefore, the objective function becomes

$$\tilde{J}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \tilde{L}(y_i, \eta(x_i;\theta)) = \frac{1}{n}\sum_{i=1}^{n} 1_{y_i=1} \log(\eta(x_i;\theta)) + 1_{y_i=-1} \log(1 - \eta(x_i;\theta))$$

Mathematically, the two objective functions are the same.

We will skip the training for multi-class classification as it is out of the scope.

Regularisation A single-layer perceptron described above uses $\theta = (W, b, w^{(o)}, b^{(o)})$, and there are in total $(p+1)m + (m+1) = mp + 2m + 1$ parameters. In some cases, the width m should be sufficiently large to obtain a good estimate. When this is the case, regularization is required. For example, add the L-2 norm penaliser

$$\text{pen}(\theta) := \sum_{i,j} w_{i,j}^2 + \sum_{k=1}^{m} (w_m^{(o)})^2$$

to the objective function $J(\theta)$. Alternatively, apply early stopping (stop when the loss on the validation set rises) to gradient descent as an implicit regularisation. Other regularisation methods are introduced in Section 13.5 of Murphy (2022).

8.3 Multi-layer Perceptron

Multiple hidden layer can be used, starting with $\boldsymbol{h}^{(0)} := x$ (input layer), define the first hidden layer (with $m^{(1)}$ nodes) as in the first section:

$$z^{(1)} := s^{(1)}\left(W^{(1)}x + b^{(1)}\right) = s\left(W^{(1)}\boldsymbol{h}^{(0)} + b^{(1)}\right)$$

where $W^{(1)} \in \mathbb{R}^{m^{(1)} \times p}$ contains the weights, $b^{(1)} \in \mathbb{R}^{m^{(1)}}$ are the biases and $s^{(1)} : \mathbb{R}^m \to \mathbb{R}^m$ is the activation (for the first layer). Suppose the second layer has $m^{(2)}$ nodes, the second hidden nodes $z^{(2)}$ learns the features of the first hidden layer $z^{(1)}$, with the same functional form $z^{(2)} := s^{(2)}\left(W^{(2)}z^{(1)} + b^{(2)}\right)$. Generally, for layer l,

$$z^{(l)} := s^{(l)}\left(W^{(l)}z^{(l-1)} + b^{(l)}\right)$$

If there are L hidden layers in total, the output is

$$y = s^{(o)}(z^{(L)})$$

. A simple example of the output activation is the linear transformation $y = (w^{(o)})^T z^{(L)} + b^{(o)}$, that is, y is essentially the $(L+1)$th layer with an identity activation function.

The activation functions $\{s^{(l)}\}_{l=1,\cdots,L}$ can be different or the same for all layers. This neural network has a high flexibility. The computational graph of a neural network with two hidden layers is shown in Figure 33.

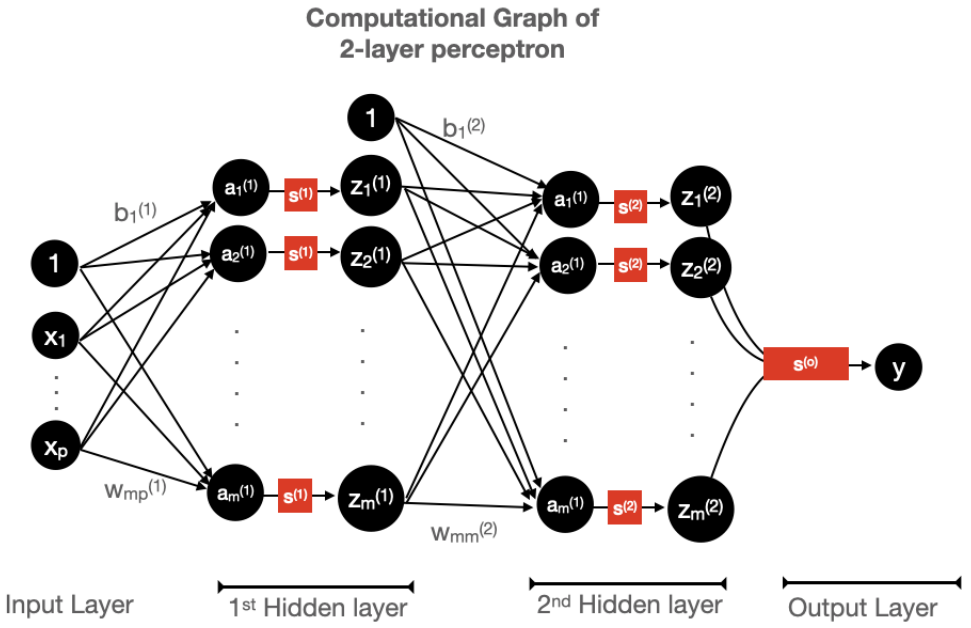


Figure 33: MLP of two hidden layers. For simplicity, the number of hidden nodes are both m in the two layers

Definition 8.4 — Feedforward Neural Network. A L-layer feedforward neural network (also called multi-layer perceptron, in short, MLP) is defined by the following procedure:

$$z^{(0)} := x$$
$$z^{(l)} := s^{(l)}\left(W^{(l)} z^{(l-1)} + b^{(l)}\right)$$

where $W^{(l)} \in \mathbb{R}^{m^{(l)} \times m^{(l-1)}}$ and $b^{(l)} \in \mathbb{R}^{m^{(l)}}$ are the unknown parameters. The output layer is $z^{(L)}$.

Remark 8.5 In the *recurrent neural network*, the output layer is allowed to feedback to the hidden layers. There are also other architectures of the neural network. But in this note, we focus on feedforward neural networks.

8.3.1 Choice of activation function

Identity Suppose activation function $s^{(l)}$ is chosen to be identity function, the layer l is called a *linear layer*. The functional form of the two layers $s^{(l-1)}, s^{(l)}$ would be the same as that of single layer. For example, if $s^{(1)} = \mathrm{id}$,

$$\begin{aligned} z^{(2)} &= s^{(2)}\left(W^{(2)} z^{(1)} + b^{(1)}\right) \\ &= s^{(2)}\left(W^{(2)}(W^{(1)} z^{(0)} + b^{(1)})) + b^{(1)}\right) \quad \text{identity activation} \\ &= s^{(2)}\left(W^{(2)} W^{(1)} z^{(0)} + (W^{(2)} + I) b^{(1)}\right) \end{aligned}$$

which is equivalent to a single-layer perceptron with weights $W := W^{(2)}W^{(1)}$ and bias $b := (W^{(2)} + I)b^{(1)}$. Multiple linear layers make a difference when regularisation is applied. For example, the L-2 regularisation for the two linear layer case above is

$$\sum_{i,j} (w_{ij}^{(1)})^2 + \sum_{k,l} (w_{kl}^{(2)})^2$$

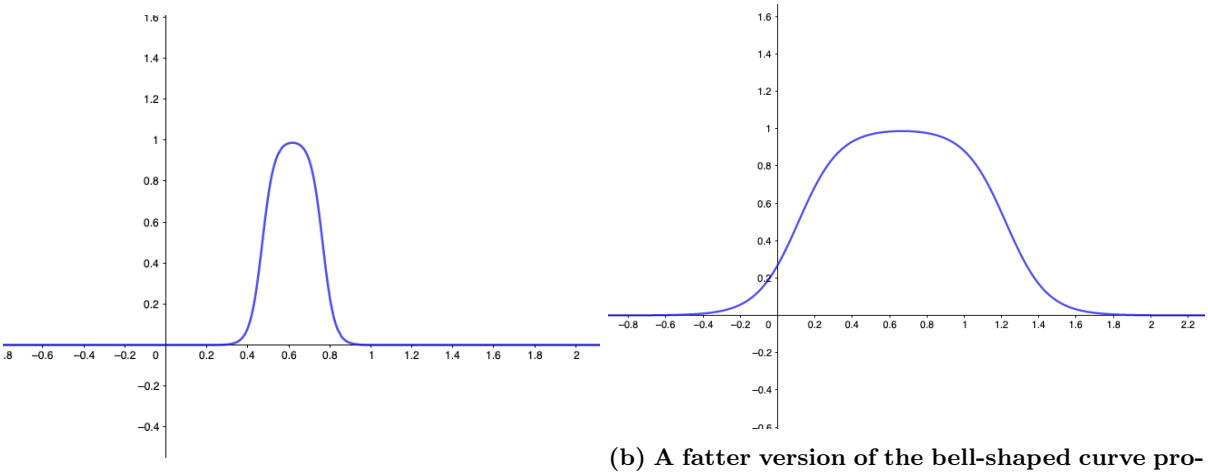
whereas the regularisation for a single linear layer is $\sum_{i,j} w_{ij}^2$ where $w_{ij} = \sum_k w_{ik}^{(2)} w_{kj}^{(1)}$. It has been illustrated in Golubeva et al. (2021) and related works that introducing a *linear bottleneck* to a layer l by breaking the weight matrix $W^{(l)}$ into $W^{(l1)}W^{(l2)}$ and applying regularization on the decomposition improves training speed and reduces the generalisation gap. In Parkinson et al. (2023), the connection of multiple linear bottlenecks to encouraging a low-rank prediction rule h (straight and parallel contour lines), which avoids overfitting.

Sigmoid and tanh We have already encountered this function several times—the logistic (sigmoid) activation function defined as $s(a) := \sigma(a) = 1/(1 + e^{-a})$, which is one of the earlist activation functions. It can be seen as a smooth version of the Heaviside step function $H(a) := 1_{a \geq 0}$, where $H(w^T x + b) = 1_{w^T x + b \geq 0}$. To understand why this activation is used, let's consider the one-dimensional setting where $w, x, b \in \mathbb{R}$. The function $x \mapsto z(x) := H(wx + b) = 1_{wx \geq -b}$ can represent any step function with a step-size of 1. If we consider a linear combination of several such Heaviside functions, it can produce flexible step functions, as illustrated on the left side of Figure 34. Intuitively, adding more Heaviside step functions (increasing the width of the hidden layer) allows the linear combination to approximate any continuous function, which is in line with the universal approximation theorem (Theorem 8.2).

Figure 34: (left) example of linear combination of Heaviside step functions; (right) using many Heaviside step functions to approximate a parabola

However, the Heaviside function is not differentiable, which prohibits the use of gradient descent to train a network with Heaviside step activation. Therefore, one may opt for the sigmoid function instead. In fact, for large w, $\sigma(wx+b) = 1/(1+e^{-wx-b})$ closely resembles a Heaviside step function. The bias b adjusts the position of the step function. For smaller values of the weight w, the linear combination of sigmoid functions actually produces quite smooth functions, as illustrated in Figure 35. Feel free to experiment with the linear combination of more sigmoid functions to discover various interesting shapes that can be constructed.

Another choice is $s(a) = \tanh(a)$, which has similar shape to the sigmoid function but takes values in $[-1, 1]$.



(a) A bell-shaped curve produced by $\sigma(wx+b) - \sigma(wx+b-10)$ where $w=-34$, $b=26$

(b) A fatter version of the bell-shaped curve produced by $\sigma(wx+b)-\sigma(wx+b-10)$ with $w=-9$, $b=11$

Figure 35: The linear combination of the sigmoid activations of two linear combinations of x

Rectifiers Both the sigmoid and tanh activations suffer from the *vanishing gradient problem*, where the gradient approaches zero as $|w^T x + b|$ becomes large. This causes the parameters w, b to stop updating in gradient descent. To address this issue, non-saturating activations like rectifiers Glorot et al. (2011) are preferred. The rectified linear unit (ReLU), defined as $s(a) = \max(a, 0)$, is one such activation that discards negative values. The linear combination of ReLUs forms line segments, as shown on the left of Figure 36. Combining many ReLU units in a linear combination should enable the approximation of any continuous function, as illustrated on the right of Figure 36. Once again, this aligns with the universal approximation theorem.

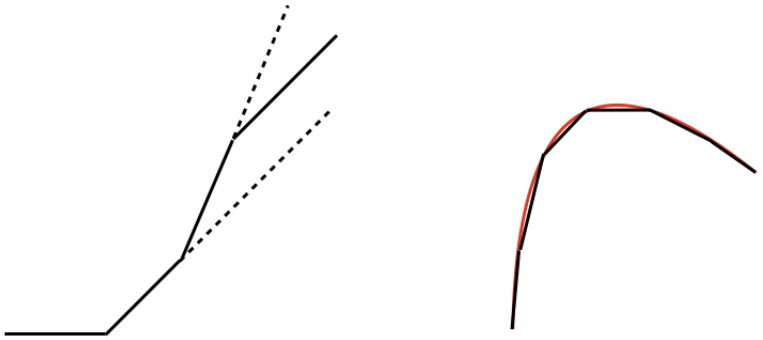


Figure 36: (left) example of linear combination of ReLUs; (right) using ReLUs to approximate a curve

8.3.2 Deep vs shallow

When the number of layers L of a feedforward neural network is relatively small ($L = 1, 2$), the network is considered *shallow*. Conversely, for a large L, the network is termed *deep*, and machine learning based on deep neural networks is referred to as *deep learning*. While the universal approximation theorem guarantees the approximation of any continuous function by a single-layer perceptron, it does not specify the required width. In fact, for complex functions, the number of hidden units needed can be enormous. Suppose the target function h^* has partial derivatives up to order D. The paper Poggio et al. (2017) proves that the parameters required for a shallow neural network to approximate h^* up to precision ϵ are $O(\epsilon^{-p/D})$, which becomes exceedingly large for high-dimensional data (large p); this phenomenon is known as *the curse of dimensionality*. In contrast, the parameters required for a deep neural network (equipped with a smooth activation function s that is not a polynomial) are $O(\epsilon^{-2/D})$.

A recent paper Parkinson et al. (2024) demonstrates that the function space that can be approximated to arbitrary precision by a 3-layer ReLU neural network is larger than that of a 2-layer ReLU neural network. In fact, an early paper Lippmann (1987) already showed graphically that using a 3-layer neural network for a classification task produces complicated decision regions that are suitable for complex data structures, whereas 2-layer network can only produce simple and convex decision regions (see Figure 14 in that paper).

There also exists a universal approximation theorem for deep neural networks Geuchen et al. (2023). This theorem states that a deep neural network with arbitrary depth (number of hidden layers) and bounded width (number of hidden units in each layer) can approximate any continuous function.

As a simple example to explain the necessity of multiple layers, let's use the sigmoid activation function to approximate a function with two bell-shaped modes, as shown on the right of Figure 37. The two-layer neural network on the left has two hidden units in the second layer, each capable of learning one mode of the function. As illustrated in Figure 35, two hidden units are sufficient to fit a bell shape, where each sigmoid activation corresponds to half of the bell. Therefore, we use 4 hidden units in the first hidden layer, allocating 2 units for each $z_i^{(2)}$. This function can be approximated by a single-layer neural network, but more hidden units are required and the interpretability is lost.

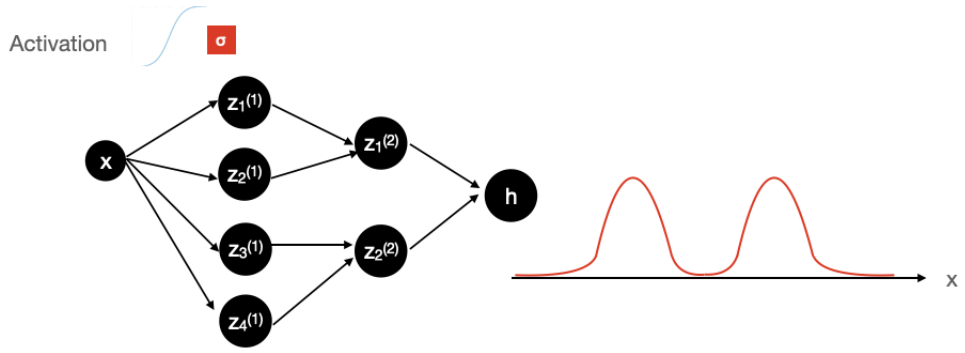


Figure 37: (right) a function with two modes. (left) the architecture of the two-layer perceptron used to estimate the function on the right. The pre-activation layers (linear transformations of x) and biases are ignored for simplicity.

In the field of image processing and image classification, convolutional neural networks (CNNs) are commonly employed, and they will be introduced in Section 8.4. Consider human faces, which have eyes, noses, ears, and each facial feature consists of basic shapes such as circles, curves, and lines. A deep CNN is capable of extracting low-level features (basic geometric shapes) first, and then extracting high-level features (like faces). Visualizations of the hidden layers trained from deep CNNs can be found in Zeiler and Fergus (2013).

8.3.3 Back-Propagation

In this section, we will discuss the training procedure of a Multi-Layer Perceptron (MLP). The parameters are still estimated by performing gradient descent on an objective function (empirical risk function with respect to some loss function L). However, there are more layers of derivatives in the chain rule, particularly for earlier layers, which increases the computational cost.

Multivariate Chain rule Suppose $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, $z \in \mathbb{R}$, and $y = f(x)$, $z = g(y)$, where f is differentiable and g has all partial derivatives, then the partial derivative of z w.r.t. x_i is

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

collecting all the partial derivatives, the gradient of z w.r.t x is

$$\nabla_x z = \left(\frac{\partial y}{\partial x}\right)^T \nabla_y z$$

where $\partial y / \partial x \in \mathbb{R}^{n \times m}$ is the Jacobin of y w.r.t. x.

In order to find the gradient of each loss term $L(y_i, h(x_i))$, we use the simplified computational graph of a K-layer feedforward neural network shown in Figure 38 where each node is multivariate (except the loss L). The output of the neural network is $h(x;\theta) = z^{(K)}$, and $L = L(y, h(x;\theta)) = L(y, z^{(K)})$.

The derivative of L w.r.t. a parameter can read from the computational graph by tracing the parent of L until reaching the parameter. For example, the bias $b^{(K-1)}$ leads to $z^{(K-1)}$, and so

$$\frac{\partial L}{b^{(K-1)}} = \frac{\partial L}{\partial z^{(K)}} \frac{\partial z^{(K)}}{\partial z^{(K-1)}} \frac{\partial z^{(K-1)}}{\partial b^{(K-1)}}$$

One could also apply chain rule between two layers of hidden units to include the nodes $a^{(l)}$, but $a^{(l)}$ is simply a linear transformation of $z^{(l-1)}$.

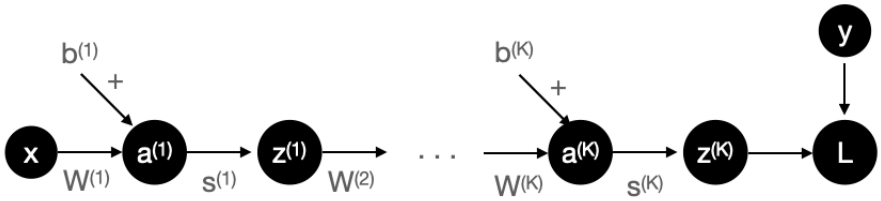


Figure 38: A simplified computational graph for K-layer feedforward neural network. Each node is a variable, and each edge represents an operation. For example, $W^{(1)}$ is the transformation $x \mapsto W^{(1)}x$.

In general, for the parameters of layer l, denoted $\theta^{(l)} := (W^{(l)}, b^{(l)})$,

$$\frac{\partial L}{\theta^{(l)}} = \frac{\partial L}{\partial z^{(K)}} \frac{\partial z^{(K)}}{\partial z^{(K-1)}} \cdots \frac{\partial z^{(l+1)}}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial \theta^{(l)}}$$

Note that each derivative in the equation above is a matrix except the first term $\partial L/\partial z^{(K)}$, which is a vector in $\mathbb{R}^{m^K}$. Computing the derivative from $\partial L/\partial z^{(K)}$ only involves matrix-vector multiplications, whereas starting from $\frac{\partial z^{(l+1)}}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial \theta^{(l)}}$ involves matrix-matrix multiplications. Therefore, the most efficient way is to start from L, and compute gradients backwards along the computational graph.

Recursive Operation Ultimately, we have to find $\partial L/\partial \theta^{(l)}$ for each layer l. Consider the following two derivatives:

$$\frac{\partial L}{\partial \theta^{(K-1)}} = \frac{\partial L}{\partial z^{(K)}} \frac{\partial z^{(K)}}{\partial z^{(K-1)}} \frac{\partial z^{(K-1)}}{\partial \theta^{(K-1)}}$$

$$\frac{\partial L}{\partial \theta^{(K-2)}} = \underbrace{\frac{\partial L}{\partial z^{(K)}} \frac{\partial z^{(K)}}{\partial z^{(K-1)}}}_{\text{redundant}} \frac{\partial z^{(K-1)}}{\partial z^{(K-2)}} \cdots \frac{\partial z^{(K-2)}}{\partial \theta^{(K-2)}}$$

there is a redundant multiplication that is already calculated when finding the first derivative. Note that for each layer $l = 1, \cdots, K$

$$\begin{aligned} \frac{\partial L}{\partial \theta^{(l)}} &= \frac{\partial L}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial \theta^{(l)}} \\ &= \frac{\partial L}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial \theta^{(l)}} \end{aligned}$$

Therefore, the derivatives of all $\theta^{(l)}$ can be calculated in a recursive fashion, which is quite efficient. This can be viewed as storing the derivatives $\partial L/\partial z^{(l)}$ in each node $z^{(l)}$ and propagating this information backward along the computational graph. Hence, the procedure is called *back-propagation* (pseudo-code provided in Algorithm 5).

Algorithm 5 Back-Propagation (for a K-layer feedforward neural network)

1: Input:

- Loss function $L : \mathcal{Y} \times \mathbb{R}^{m^{(K)}} \to \mathbb{R}^+$ where $m^{(K)}$ is the dimension of the output layer
- parameters $\theta^{(l)}$ for each layer $z^{(l)}$ and the derivative $\partial z^{(l)}/\partial \theta^{(l)}$

2: compute $\partial L/\partial z^{(K)}$

3: **for** $l = K-1, \cdots, 1$ **do**

4: Compute

$$\frac{\partial L}{\partial z^{(l)}} = \frac{\partial L}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial z^{(l)}}$$

5: Compute

$$\frac{\partial L}{\partial \theta^{(l)}} = \frac{\partial L}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial \theta^{(l)}}$$

6: **end forreturn** the gradients $\{\frac{\partial L}{\partial \theta^{(l)}}\}_{l=1,\cdots,K}$

For stochastic gradient descent, Algorithm 5 should be run for each selected data point (x_i, y_i), and the derivative of the empirical risk w.r.t. $\theta^{(l)}$ is given by

$$\frac{\partial J(\theta)}{\partial \theta^{(l)}} = \frac{1}{n} \sum_i \frac{\partial L(h(x_i; \theta), y_i)}{\partial \theta^{(l)}}$$

With the presence of regularization, the derivative of the penaliser should be added to the above expression.

Note that for other architectures, back-propagation also works. In each recursive step, the gradients are passed to the parents of a node in the computational graph. Python packages like TensorFlow and PyTorch are designed for performing back-propagation on arbitrary computational graphs.

8.4 Convolutional Neural Network

Neural networks can be applied in image processing, where each image contains three 2-dimensional grids representing the RGB channels. Therefore, for an image with a resolution of $n \times n$, the input vector (a vectorised version of the grids) has a dimension of $3n^2$, which can be extremely high. For a feedforward neural network described in the previous section, this results in too many parameters in the first layer.

The feedforward neural network contains the linear processing unit, i.e., $a = w^T + b$, which are *saturated* in the sense that each node in the layer is connected to all nodes in the previous layer and the next layer. Alternatively, one may use the convolution processing unit $a = w * x + b$, where $(w * x)$ is the convolution of w and x. The idea of convolution comes from a simple probability problem: finding the distribution of $X + Y$ where X, Y are discrete/continuous random variables. Suppose for simplicity $X, Y \in \{1, 2, \ldots, 10\}$, then $X + Y \in \{1, 2, \ldots, 20\}$, and

$$P(X + Y = n) = \sum_{i+j=n} P(X = i)P(Y = j) = \sum_{i=1}^{9} P(X = i)P(Y = n - i) = \sum_{i=-\infty}^{\infty} P(X = i)P(Y = n - i)$$

For the sake of generalisation to arbitrary discrete random variables, the upper and lower bounds of the sum are replaced by $\pm\infty$. In this case, the value of the sum is unchanged because at least one of $P(X = i)$, $P(Y = n - i)$ is 0 for $i \notin \{1, 2, \ldots, 9\}$.

Definition 8.6 — (discrete) Convolution. The convolution between two discrete random variables X and Y (with support on $\mathbb{N}$), denoted $X * Y$, is

$$X * Y := \sum_{i=-\infty}^{\infty} P(X = i)P(Y = n - i)$$

Suppose $X = \{X_i\}_{i \in I_X}, \{Y\}_{j \in I_Y}$ are lists of real numbers, the convolution between X, Y is defined as:

$$X * Y := \sum_{i=-\infty}^{\infty} X_i Y_{n-i}$$

where $x_i = 0$ if $i \notin I_X$ and similarly for Y.

Intuitively, the convolution between two lists X and Y is akin to using a scanner Y and scanning through the list X, as shown in Figure 39 for an example. The list $Y = (1/3, 1/3, 1/3)$ produces moving 3-averages of the list X. Note that for the first term (top left of the figure), I implicitly added two 0's in front of X so that this cross product can be properly defined. This is called *padding*, a technique of adding 0's outside the boundary to avoid loss of information on the boundaries of the image. The number of 0's beyond each boundary, denoted p, can be adjusted. In Figure 39, $p = 2$. If $p = 0$ (no padding), we start straight away with the cross-product on the bottom left involving x_1, x_2, x_3.

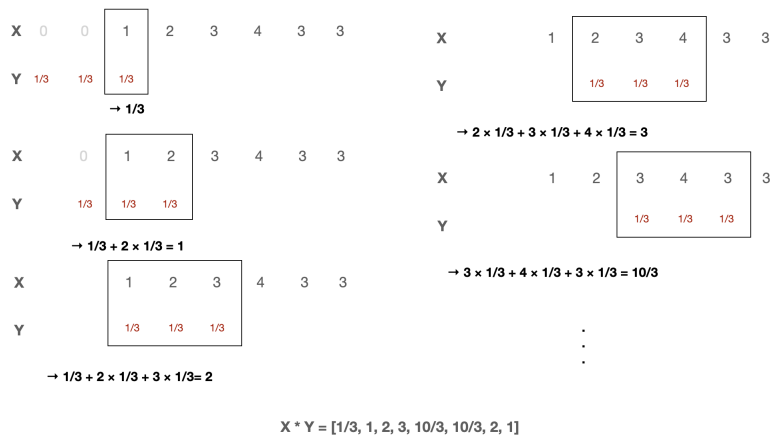


Figure 39: Convolution of 1-dimensional lists

Continuous convolution Convolution can also be taken between two functions h, k, replacing the summation by integration:

$$(h * k)(t) := \sum_{\tau=-\infty}^{\infty} h(\tau)k(t-\tau)$$

Given a function h, the function k here is called a *kernel* for h, which can be chosen according to the task. For example, if k is chosen to be the PDF of the standard normal distribution, then it makes h smoother (think of why using the idea in Figure 39).

convolution for image The 2-D convolution, defined below, can be understood as using a square scanner to traverse through a 2-D grid, representing an image. This operation is highly effective for feature extraction in image processing. For instance, Figure 40 illustrates a hand-written digit "1" on a 6×6 grid, where h denotes the pixel values, and the scanner (i.e., kernel) k is a 3×3 array.

Definition 8.7 — (2-D discrete) Convolution. The convolution between two dimensional tensors (matrices) $A \in \mathbb{R}^{n\times n}, B \in \mathbb{R}^{d\times d}$ (WOLG assume $d < n$), denoted as $A * B$, is a $n - d + 1$ by $n - d + 1$ matrix where the i, j component is

$$[A * B]_{i,j} = \sum_{k=i}^{i+d} \sum_{l=j}^{j+d} A_{kl} B_{kl}$$

Remark 8.8 Technically speaking, this operation is a cross-product. In a strict sense, convolution involves flipping the matrix A before taking the cross-product. However, in image processing, flipping the image does not alter its content. Therefore, we do not distinguish between convolution and cross-product in this section.

Remark 8.9 The idea of using 2-D convolution for image processing is inspired by the mechanism of a cat's vision, which scans through the image using a focus region called a *receptive field*. This concept was formalized into a neural network called *Neocognitron* Fukushima (1980). Although the algorithm is not used nowadays due to the lack of backward-propagation, it has inspired many future CNN designs (e.g., Alsheikhy et al. (2020)) and has been cited thousands of times.

Using the following kernel:

$$k = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

The value of the cross-product between a 3x3 region on the image $h \in \mathbb{R}^{6\times 6}$ and k will be maximized when there is a straight line in the middle of the scanner, capturing the feature of a "vertical line" in the image. Of course,

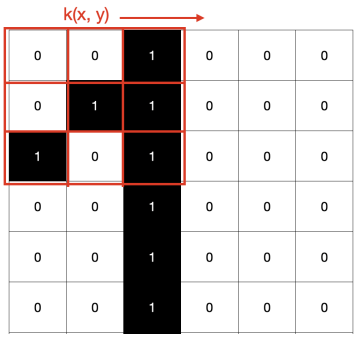


Figure 40: Convolution(cross-product) of two matrices: using a kernel k to scan through the image of a hand-written digit 1

the kernel k can be adjusted to capture other features. The convolution of h and k in this example is given by:

$$h * k = \begin{pmatrix} 1 & 3 & 0 & 0 \\ 1 & 3 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{pmatrix} \tag{8.1}$$

You may substitute the components of h and k to verify this convolution operation. Note that without padding, any edge on the boundary of the grid will not be recognised.
Examples of other kernels for image processing are given below

- Margin detector:
$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$
when all pixel values in a 3 by 3 region are the same, the cross-product is 0. However, if there is a sharp change in the value, the cross-product is quite large.

- Sharpening:
$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$
it makes the central pixel value in the 3 by 3 region differs more from the four neighbours (left, right, above, below).

- Gaussian blur
$$\frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$
This kernel has a bell-shape like a 2-dimensional Gaussian distribution. As opposed to sharpening, it pulls the value of the central node closer to all its neighbours.

Pooling Convolution does reduce the dimension by $d - 1$, but only by a small amount because the kernels are usually not very large. Note that the second column of the matrix in Equation 8.1 describes the same feature, the pooling technique takes the average or maximum of the values in a block. For example, the 2×2 max-pooling of the matrix in Equation 8.1 is:

$$\text{pool}(h * k) = \begin{pmatrix} 3 & 0 \\ 3 & 0 \end{pmatrix}$$

This pooled input only has 4 dimensions, whereas the original image has 36 dimensions.

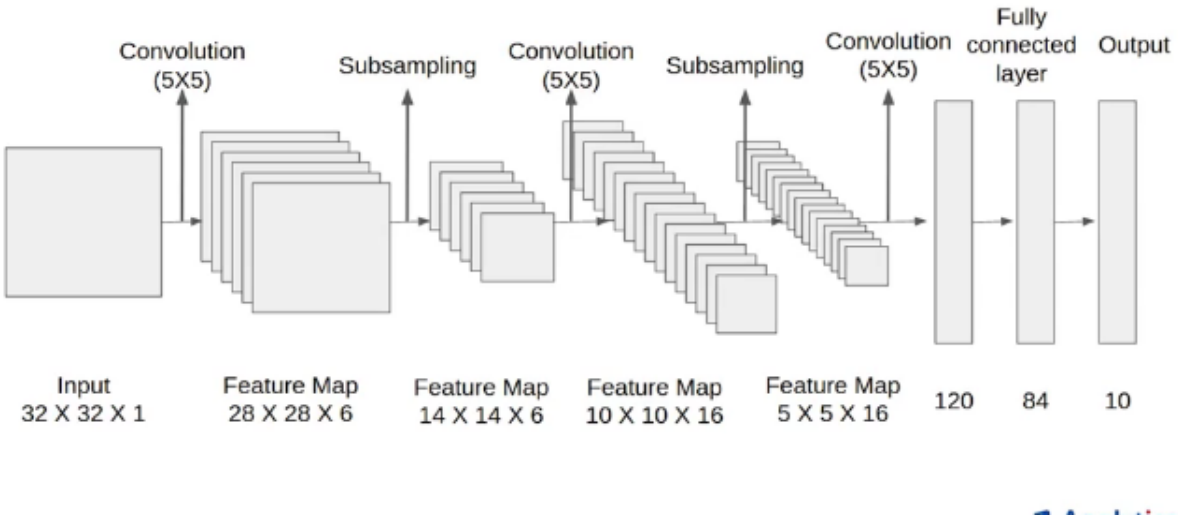


Figure 41: The architecture of LeNet-5 for processing a 32 by 32 image taken from Shipra Saxena - The Architecture of Lenet-5. Sub-sampling means pooling

An early convolutional neural network known as *LeNet-5* by Lecun Lecun et al. (1998) utilises two convolutional layers along with two max-pooling layers to generate a low-dimensional vector. This vector is then used as an input for a 2-layer feedforward neural network. The overall architecture is illustrated in Figure 41.

Suppose the image size is larger; scanning through the entire image with a 3×3 kernel k may be slow. To speed up the process, a stride, indicating the spacing between each point of the scanner k, can be added. In real-life scenarios, where image sizes may be 1800×1200 or even larger, larger strides and kernels are often considered. For example, the hand written digit with higher resolution in 42 is scanned using a 3 by 3 kernel with a stride of 2.

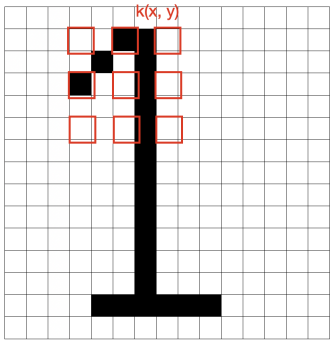


Figure 42: Convolution with stride $s = 2$

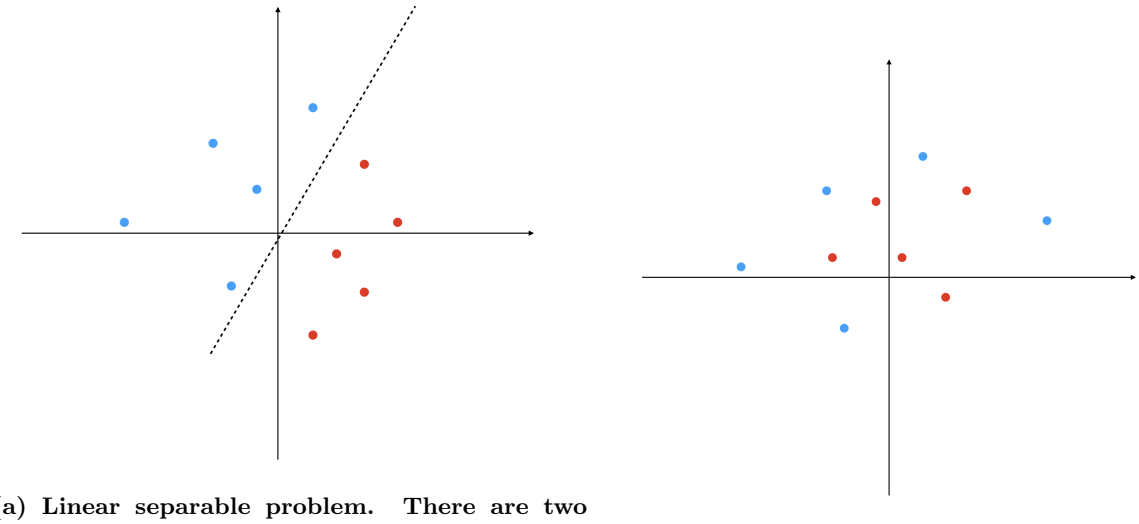
The idea of convolution can be generalized to any dimension, such as 3D for video processing. Chapter 7 of "Dive into deep learning" introduces some Python code implementations of the convolution networks discussed above, and Chapter 8 introduces modern CNN architectures, including GoogLeNet, ResNet, and DenseNet.

9 Decision Tree and Random Forest

In this chapter, x_j is considered the jth attribute (predictor), and $x^{(i)}$ denotes the ith data point.

9.1 Decision Tree

For a linearly separable dataset, as illustrated in Figure 43a, it is straightforward to draw a separating line represented by $w^T x + b = 0$, where $w \in \mathbb{R}^2$. This line effectively distinguishes the data points $x^{(i)}$ into their respective classes. However, real-world datasets are often more intricate, as depicted in Figure 43b. In such cases, finding a line or curve with a simple functional form (e.g., polynomial or circles) to separate the two classes becomes challenging.

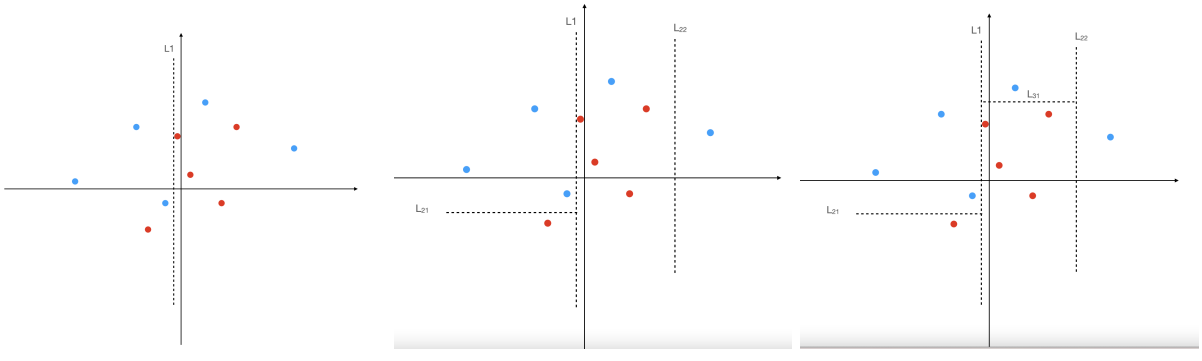


(a) Linear separable problem. There are two classes: blue and red

(b) Not linearly separable problem

Figure 43: Linear Separation

Instead of finding a complicated decision boundary to separate the two classes at once, we partition the space into several regions $\mathcal{R}_{ik=1,\cdots,R}$ using only straight lines. (See figure 44) More precisely, we only use decision boundaries in the form $x_j = v$ for $j \in 1, \cdots, p$ and $v \in \mathbb{R}$.



(a) First layer **(b) Second layer** **(c) Third layer**

Figure 44: Decision tree demonstration on a data set

First, line L_1 is drawn to roughly separate the two groups(Figure 44a). There is only one red point to the left of L_1, and it can be separated from the blue points with a horizontal line L_{21}. The left of L_1 is already fully separated (each region is *pure*). As for the right of L_1, there is no one-step solution to separate the points, so first

draw L_{22} to isolate the blue point at the far right(Figure 44b), and then the other blue point left can be separated using L_{31}(Figure 44c).

A good partition of the sample space for classification should have more pure regions (all points are from one class), or regions close to being a pure region (e.g. 10 blue and 1 red). For the partition in Figure 44, all regions are pure.

Definition 9.1 — Impurity vector. For classification task with K classes, the impurity vector of a region $\mathcal{R} \subseteq \mathcal{X}$, denoted $\boldsymbol{\eta}(\mathcal{R})$, is defined by

$$\boldsymbol{\eta}(\mathcal{R}) := \begin{pmatrix} \eta_1(\mathcal{R}) \\ \eta_2(\mathcal{R}) \\ \vdots \\ \eta_K(\mathcal{R}) \end{pmatrix} \quad \text{where } \eta_k(R_\alpha) := \frac{\sum_{i=1}^n I(x^{(i)} \in \mathcal{R},\, y^{(i)} = k)}{\sum_{i=1}^n I(x^{(i)} \in \mathcal{R})}$$

where $I(A)$ is the indicator function of an event A.

Each term $\eta_k(\mathcal{R})$ represents the empirical proportion of class k in the region $\mathcal{R}$. For a pure region, $\boldsymbol{\eta}(\mathcal{R})$ has only one entry with a value of 1, and all other entries are 0.

The recursive process of partitioning the space using binary partitions can be demonstrated using a *decision tree*, as shown in Figure 45. Each node represents a region in the sample space, where the top node, referred to as the *root node*, represents the entire sample space. Every node either has no children or two children (*left child* and *right child*), with the children representing a binary partition of the region. The node above each layer is the *parent node* for the connected nodes below it. The region of a child is always a subset of the region represented by its parent. A node without any children is termed a *leaf node*, and a color is assigned to it, representing the class label for that region. The regions represented by all the leaf nodes form a partition of the sample space.

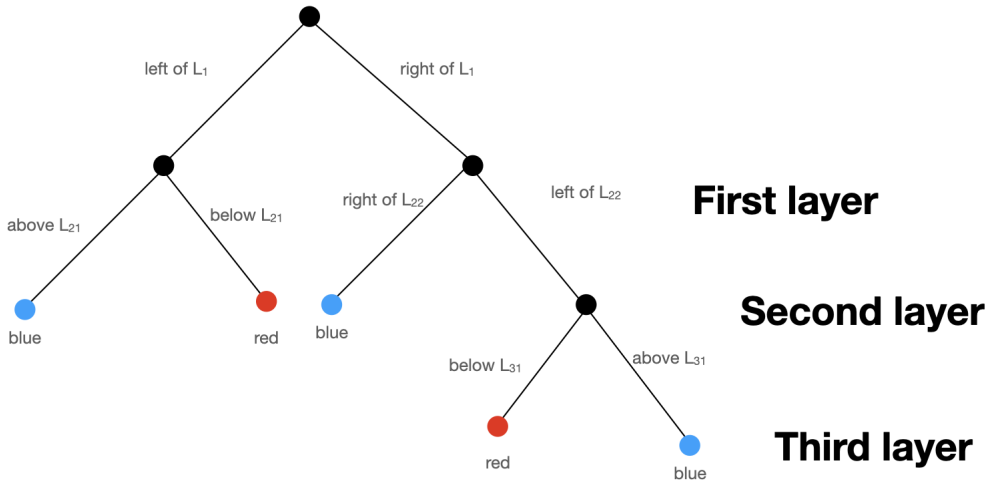


Figure 45: The tree, Decision Tree

The class label for each leaf label(representing region $\mathcal{R}$) is chosen by *majority vote*: assign the class that appears the most to the region:

$$\widehat{\beta}(\mathcal{R}) \underset{k=1,\cdots,K}{\arg\max} \frac{\sum_{i=1}^n I(x^{(i)} \in \mathcal{R}) I(y^{(i)} = k)}{\sum_{i=1}^n I(x^{(i)} \in \mathcal{R})}$$

Suppose for $j = 1, \cdots, R$, $\widehat{\beta}_j$ is the class assignment to region of leaf node j that represents region $\mathcal{R}_j$, then the prediction rule represented by this decision tree is

$$h(x) = \sum_{i=1}^{R} \beta_i \mathcal{R}_i$$

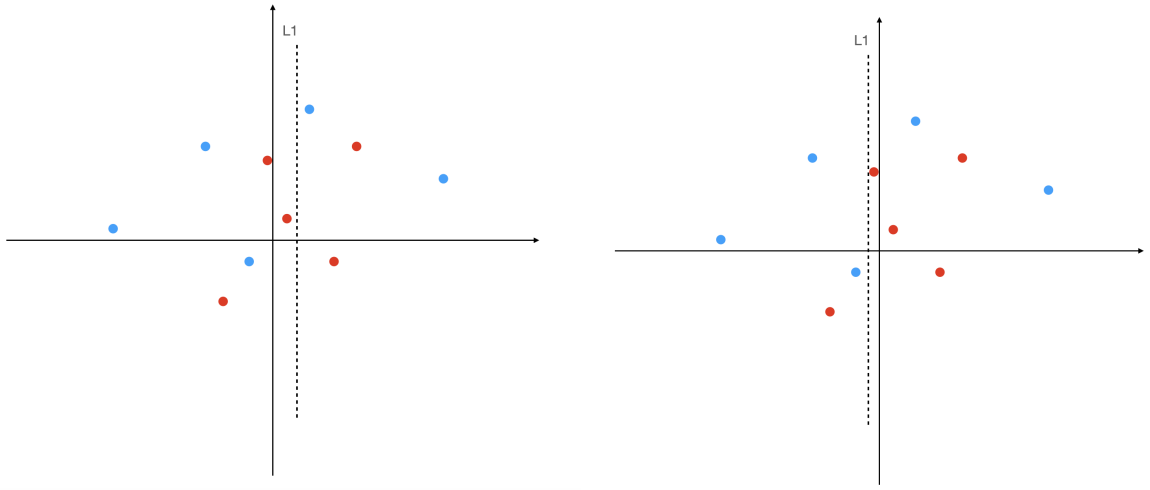
it can be shown that the majority vote estimator for β_i is the Bayes rule under 0-1 loss.

Choice of Separation line At each step, there are infinitely many choices of separation lines (if the attributes are continuous). For example, the first separation line L_1 on $\mathbb{R}^2$ can be any of the following:

$$\{(x_1, x_2) : x_1 = v\}, \quad \{(x_1, x_2) : x_2 = s\}$$

where $v, s \in \mathbb{R}$.

However, some choices are better than others. Figure 46 presents two choices of L_1 for the dataset in Figure 43b. For choice 1, there are 3 blue and 3 red points on the left side, and 2 blue and 2 red points on the right side. If we stop here and estimate the class labels using this separation, there is only 50% accuracy, which is not different from that before separation. (In information theory, we say the entropy did not change, or information gain is 0.)



(a) Choice 1 **(b) Choice 2**

Figure 46: Two choices of the first separation on $\mathbb{R}^2$

As for choice 2 (Figure 46b), 3 points out of 4 to the left of L_1 are blue, and 4 points out of 6 to the right of L_1 are red. So, we assign blue to the left region and red to the right region. Let B represent blue and R represent red. The accuracy of estimation based on this separation is:

$$P(\text{left})P(B \,|\, \text{left}) + P(\text{right})P(R \,|\, \text{right}) \approx \frac{1}{2}\frac{3}{4} + \frac{1}{2}\frac{4}{6} = \frac{17}{24} > \frac{1}{2}$$

so choice 2 is better than choice 1 in terms of accuracy.

Estimation of the partition Given a impurity measure $i(\mathcal{R}) \in \mathbb{R}^+$ (summarises the impurity vector $\boldsymbol{\eta}(\mathcal{R})$ with a single positive number, will be introduced later), the *information gain* for a partition $\mathcal{R}_l \cup \mathcal{R}_r = \mathcal{R}$ (given separation line $x_j = v, \mathcal{R}_l = \{x \in \mathcal{X} : x_j < v\}$ and $\mathcal{R}_r = \{x \in \mathcal{X} : x_j \geq v\}$), is defined as

$$\text{IG}(\mathcal{R}_l, \mathcal{R}_r) = i(\mathcal{R}) - q_l i(\mathcal{R}_l) - q_r i(\mathcal{R}_r)$$

where $q_l := |\{i = 1, \cdots, n : x^{(i)} \in \mathcal{R}_l\}|$ is the number of data points in region $\mathcal{R}_l$.

Clearly, a good separation line should reduce the impurity measures in $\mathcal{R}_l$ and $\mathcal{R}_r$, or equivalent, maximises information gain. If any of the sub-regions $\mathcal{R}_l$, $\mathcal{R}_r$ is not pure, a new separation line is searched by maximising information gain over that sub-region. It is repeated until a reasonable purity (the impurity vector $\boldsymbol{\eta}(\mathcal{R})$ close to e_k for any class k) is achieved for all leaf nodes, or when the information gain no longer increases. This method is called *Greedy search*.

Note that for each attribute x_j considered, there is no need to search through lines $x_j = v$ for all $v \in \mathbb{R}$. Denote the set of indices to the left of line as $I^- := \{i \in \{1, \cdots, n\} : x_j^{(i)} < v\}$, and set of indices to the right of the line as $I^+ := \{i \in \{1, \cdots, n\} : x_j^{(i)} \geq v\}$. For the data set with three points in Figure 47, there are effectively only four possible outcomes of the binary partition:

- $v \leq x_j^{(1)}$: $I^- = \emptyset$, $I^+ = \{1, 2, 3\}$
- $x_j^{(1)} < v \leq x_j^{(2)}$: $I^- = \{1\}$, $I^+ = \{2, 3\}$
- $x_j^{(2)} < v \leq x_j^{(3)}$: $I^- = \{1, 2\}$, $I^+ = \{3\}$
- $v > x_j^{(3)}$: $I^- = \{1, 2, 3\}$, $I^+ = \emptyset$

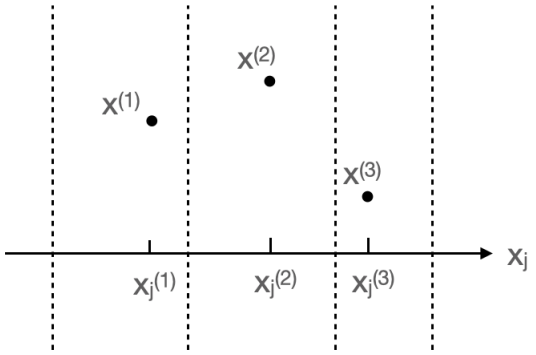


Figure 47: Illustration of the choice of partition line for a data set with only three points

In general, for a p-dimensional data set with n points, there are only $(n+1)p$ effective partitions for the data set. The one with highest value of q_l is selected.

9.1.1 Impurity Measures

A first instinct of the impurity measure would be the probability of misspecification(after majority vote): $i(\mathcal{R}) = 1 - \max_k \eta_k$ where η_k are the components of the impurity vector $\boldsymbol{\eta}(\mathcal{R})$. However, the impurity function has a triangle shape along each η_k (see the plot for binary classification in 49), which makes the information gain 0 for any separation that keeps η_k on one side of 0.5. Entropy and Gini index introduced below solves this problem.

Entropy A commonly used impurity measure is *entropy*. It quantifies how much information would be given on average by an event. For events with lower probabilities, such as the moon being bombarded, you will be surprised and gain a lot of information from this event. In contrast, if you see the sun rising from the east, there is no extra information for you because this happens every day, and it has a high probability. In general, the entropy of events with a high probability should be close to 0, and the entropy of events with low probabilities should be high, i.e. it is a decreasing function in the probability of the event. In addition, we require the information function $h : \mathscr{F} \to \mathbb{R}^+$ (where $\mathscr{F}$ is the space of all events) to satisfy additivity:

$$h(E \cap F) = h(E) + h(F) \quad \text{where } E, F \in \mathscr{F} \text{ are independent events}$$

In other words, the information of the events E and F occurring together is exactly the sum of the information given by E and the information given by F. It is important to note that $P(E \cap F) = P(E) \cdot P(F)$, so clearly,

the function h should involve logarithm. An example is $h(E) := \log_2(1/P(E)) = -\log_2(P(E))$. (Other logarithm bases also work; 2 is the conventional choice in information theory.) Note that $1/P(E) \in [1, \infty)$, so $h(E) \geq 0$ and $h(E) = 0$ if and only if $P(E) = 1$.

The entropy of a random variable is defined as the expected value of the information function h for a discrete variable X:

$$\text{entropy}(X) = E_X(h(X)) = -\sum_x P(x) \log_2(P(x))$$

If the number of possible values of X is fixed, this quantity is maximised when all probabilities are equal. For example, if $X \in \{0, 1\}$, the entropy is maximised when $P(X = 0) = P(X = 1) = 0.5$. Figure 48 shows a 3D plot of the entropy function against $P(X = 0)$ and $P(X = 1)$.

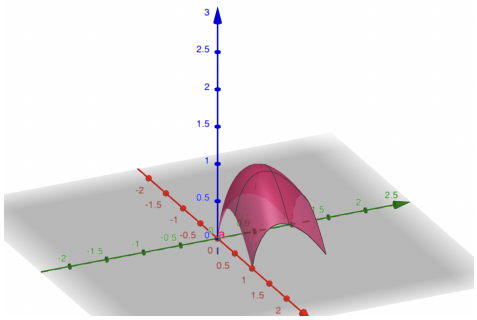


Figure 48: Demonstration of entropy on two values

In the case of a decision tree, the entropy(impurity measure) of region $\mathcal{R}$ is defined as follows:

$$\text{entropy}[\boldsymbol{\eta}(\mathcal{R})] := -\sum_{k=1}^{K} \eta_k(\mathcal{R}) \log\left(\eta_k(\mathcal{R})\right)$$

Suppose an estimate on the class probabilities(impurity vector) in region $\mathcal{R}$, $\hat{\boldsymbol{\eta}}(\mathcal{R})$, is already obtained using other methods, the *cross-entropy* between the collected data and the model distribution defined as:

$$\text{CE}[\boldsymbol{\eta}(\mathcal{R}), \hat{\boldsymbol{\eta}}(\mathcal{R})] := -\sum_{k=1}^{K} \eta_k(\mathcal{R}) \log\left(\hat{\eta}_k(\mathcal{R})\right)$$

can be used. Cross-entropy also measures the quality of the estimator $\hat{\boldsymbol{\eta}}(\mathcal{R})$. Note that $\text{CE}[\cdot, \cdot]$ is not commutative, so $\text{CE}[P, Q] \neq \text{CE}[Q, P]$.

Gini index One drawback of using entropy as a loss function is that log is computationally expensive. There is an alternative: simply remove the logarithm in $-\sum_x P(x) \log(P(x))$, obtaining $-\sum_x P(x)^2$. But if the probability is concentrated, i.e. $P(x) = 1$ for only one value of x, then $-\sum_x P(x)^2 = -1$. In order to obtain positive values, 1 is added to the sum. The *Gini index* of a discrete random variable x

$$GI(X) = 1 - \sum_x P(X = x)^2$$

Gini index and entropy are actually very similar(Figure 49) and are both maximised when X follows a uniform distribution.

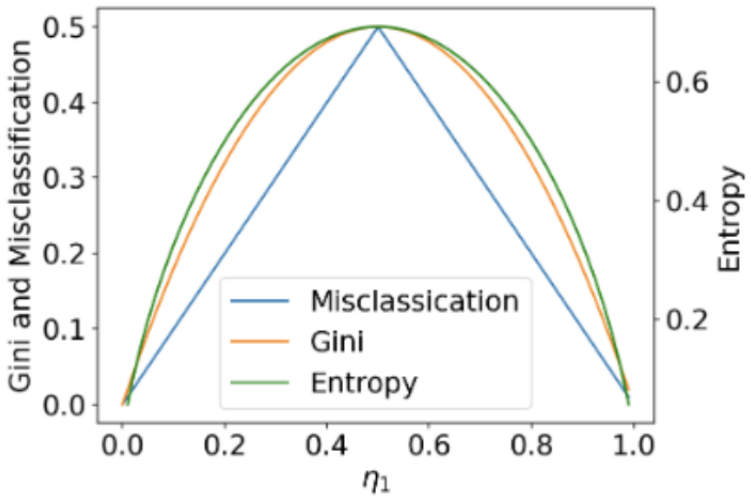


Figure 49: Comparison of the three quality measures on binary classification ($Y \in \{\pm 1\}$): sliced along $\eta_1 = P(Y = 1)$. The slice along η_{-1} is not plotted because $\eta_{-1} = 1 - \eta_1$, the curves will be the same.

Note that $\eta_k(\mathcal{R}_l)$ and $\eta_k(\mathcal{R}_r)$ must be on opposite sides of $\eta_k(\mathcal{R})$. For example, the two red nodes in Figure 50 represent the sub-regions. The weighted average of impurity on the two sub-regions, $q_l i(\mathcal{R}_l) + q_r i(\mathcal{R}_r)$, lies on the dotted line connecting the red nodes. Thus, the information gain is the difference between the height of the black node (representing $\mathcal{R}$) and the orange node (the weighted average of $\mathcal{R}_l$ and $\mathcal{R}_r$). Information gain is always positive due to concavity, which ensures the continuous progress of the greedy search algorithm. On the other hand, using misspecification error stops the algorithm straightaway.

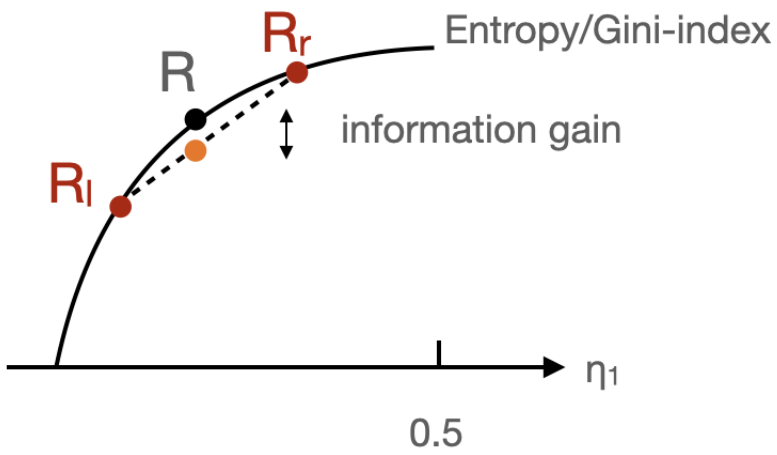


Figure 50: Using Gini index/entropy creates positive information gain.

Regularisation a deep enough decision tree is capable of drawing a box around each data point and achieve 0 training loss. However, this is clearly overfitting. One could resolve this by early stopping: use train-validation-test separation, and stop when the validation loss starts increasing. Setting a maximum depth for the tree or minimum leaf size (number of data points in the region represented by the leaf node) also act as regularisation. There is a technique called *pruning*Mansour (1997) that runs the greedy search until convergence, and prune the tree by merging some leaf nodes.

9.2 Regression tree

If the output variable y is continuous instead of categorical, we can still apply the idea of the decision tree. The estimator for each region $\mathcal{R}$ is simply the sample mean:

$$\widehat{\beta}(\mathcal{R}) = \frac{\sum_{i=1}^{n} y^{(i)} I(x^{(i)} \in \mathcal{R})}{\sum_{i=1}^{n} I(x^{(i)} \in \mathcal{R})}$$

The key problem, once again, is finding the best split. This involves minimising the squared error (impurity of a region) in the partition $\mathcal{R}_l \cup \mathcal{R}_r = \mathcal{R}$. Define the squared error of region $\mathcal{R}$ as $L(\mathcal{R}) := \sum_{i=1}^{n} I(x^{(i)} \in \mathcal{R}_l) \left(y^{(i)} - \widehat{\beta}(\mathcal{R}_l)\right)^2$. Then, the squared error of the partition is $L(\mathcal{R}_l) + L(\mathcal{R}_r)$. Minimising this squared error is equivalent to maximising the *variance reduction* $L(\mathcal{R}) - L(\mathcal{R}_l) - L(\mathcal{R}_r)$.

The greedy search can be employed to find the best partition of the parameter space. At each step (each leaf node), the best split is found by minimising the squared loss, and the process is repeated on the sub-regions. Stop until the squared error is low enough in each leaf node or until the variance reduction becomes 0. Since the decision tree and the greedy search algorithm used to train the tree can be applied to both regression and classification tasks, it is also called *Classification and Regression Tree*(CART).

Remark 9.2 Although this decision tree is called a regression tree, no linearity is present. The resulting prediction will consist of many discrete values, resembling stairs. An alternative approach is to use a *linear tree*, where the prediction in each region (leaf node) is determined by linear regression (see Figure 51). For one-dimensional input x, the prediction rule produced by a linear tree is akin to a broken-stick (segmented) regression estimator, but the breakpoints are learned from the data, representing an *adaptive approach*.

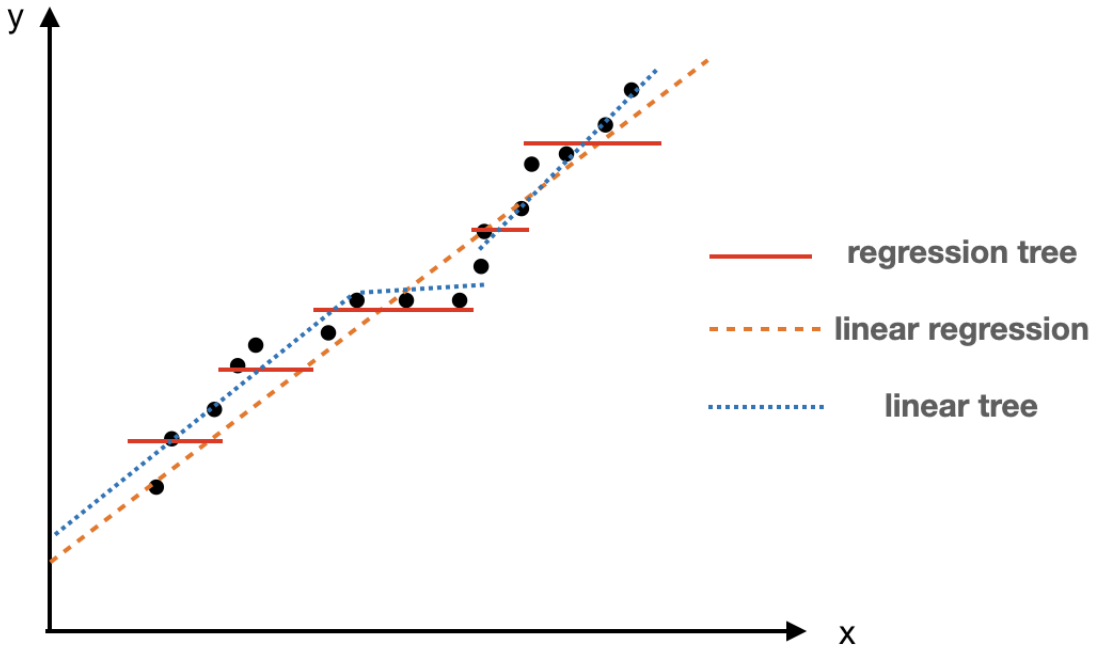


Figure 51: comparison of the estimators obtained from linear regression, regression tree and linear tree

9.3 Ensemble methods

9.3.1 Trees to Forests

There are several reasons that decision trees (CART) are widely used:

- Works for complicated data: Simply increasing the maximum depth allowance of the tree can handle complex datasets.
- Works for mixed data (with both discrete and continuous attributes).
- **Automatic variable selection**: If attribute x_i is not relevant to y, it will not be chosen as a splitting criterion at any step. You don't have to identify and remove irrelevant variables beforehand.
- **Interpretable**: It is not a black box; each layer of the tree has a meaningful interpretation related to the predictor/attribute.
- **Outliers**: Like linear classifiers, it is robust to outliers.
- Low computational cost to train the tree.

However, the drawbacks include the potential to overfit and instability (the variance of prediction rule, $\text{Var}_{D\sim P_0}[\widehat{h}^{(D)}(x)]$, is high). A small change in the training set may result in completely different trees, as shown in Figure 52.

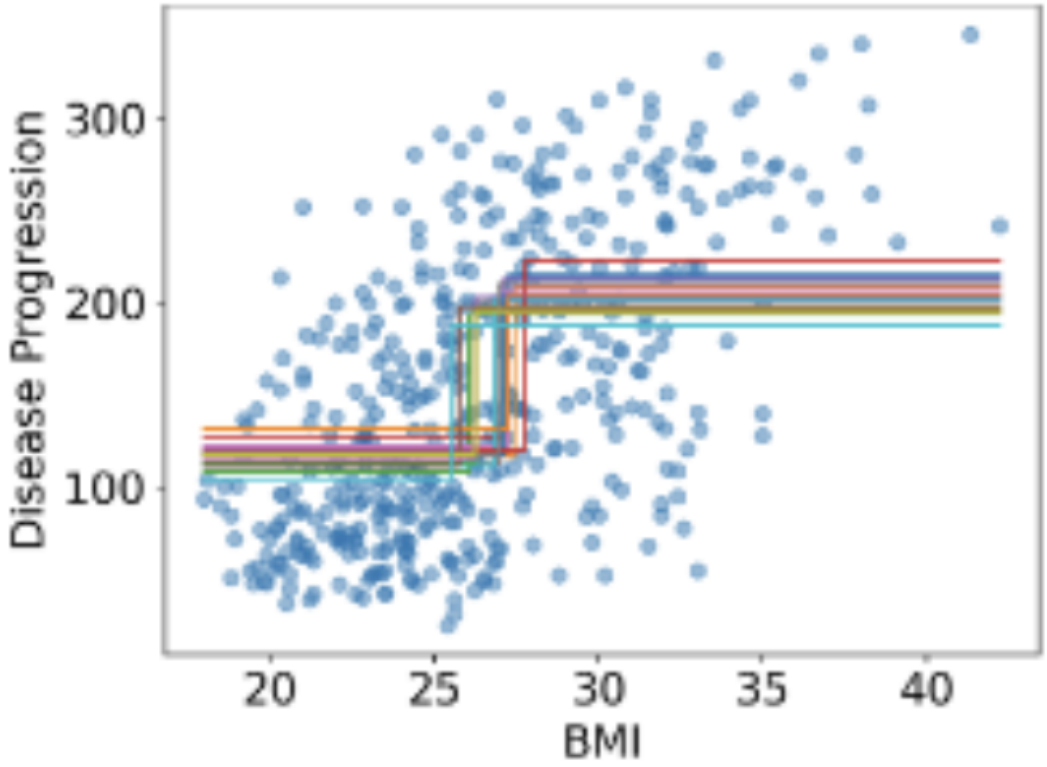


Figure 52: estimators from one-split regression trees trained on bootstrap samples: even the first split can vary drastically

Bagging To resolve instability, the *ensemble method* is used. A *forest* of B decision trees is built, where each tree $b = 1, \cdots, B$ is trained using a bootstrap sample $D_b \subseteq D$ of the data set $D = \{(x^{(i)}, y^{(i)})\}_{i=1,\cdots,n}$ (sampled WITH replacement) The prediction rule is defined by the *bagging*(bootstrap and aggregating) estimator:

$$\widehat{h}^{(D_1,\cdots.D_B)}(x) = \frac{1}{B}\sum_{b=1}^{B}\widehat{h}^{(D_b)}(x) \quad \text{for regression tree}$$

$$\widehat{h}^{(D_1,\cdots.D_B)}(x) = \underset{y=1,\cdots,K}{\arg\max}\sum_{b=1}^{B} I\left(y = \widehat{h}^{(D_b)}(x)\right) \quad \text{for classification tree}$$

where $h^{(D_b)}$ is the prediction rule obtained from a decision tree trained on the data set D_b.

Effectiveness of bagging Given independent data sets $\{D_b\}_{b=1,\cdots,B}$,

$$E_{D_1,\cdots,D_B\sim P_0}[\widehat{h}^{(D_1,\cdots,D_B)}(x)] = \frac{1}{B}\sum_{b=1}^{B} E_{D_b\sim P_0}[\widehat{h}^{(D_b)}(x)] = E_{D\sim P_0}[\widehat{h}^{(D)}(x)]$$

that means: the bias of the aggregated estimator will be the same. However, the variance significantly decreases (for large B):

$$\mathrm{Var}_{D_1,\cdots,D_B\sim P_0}[\widehat{h}^{(D_1,\cdots,D_B)}(x)] = \frac{\mathrm{Var}_{D\sim P_0}[\widehat{h}^{(D)}(x)]}{B}$$

As for the bagging estimator, the data sets D_b are not independent due to bootstrapping, but the raise in bias is minor compared to the reduction in variance. This is an example of bias-variance trade-off, indeed, bagging reduces generalisation error Grandvalet (2004).

In figure 53 (a), (b), (c), the generalisability of the prediction rule increases as B increases.

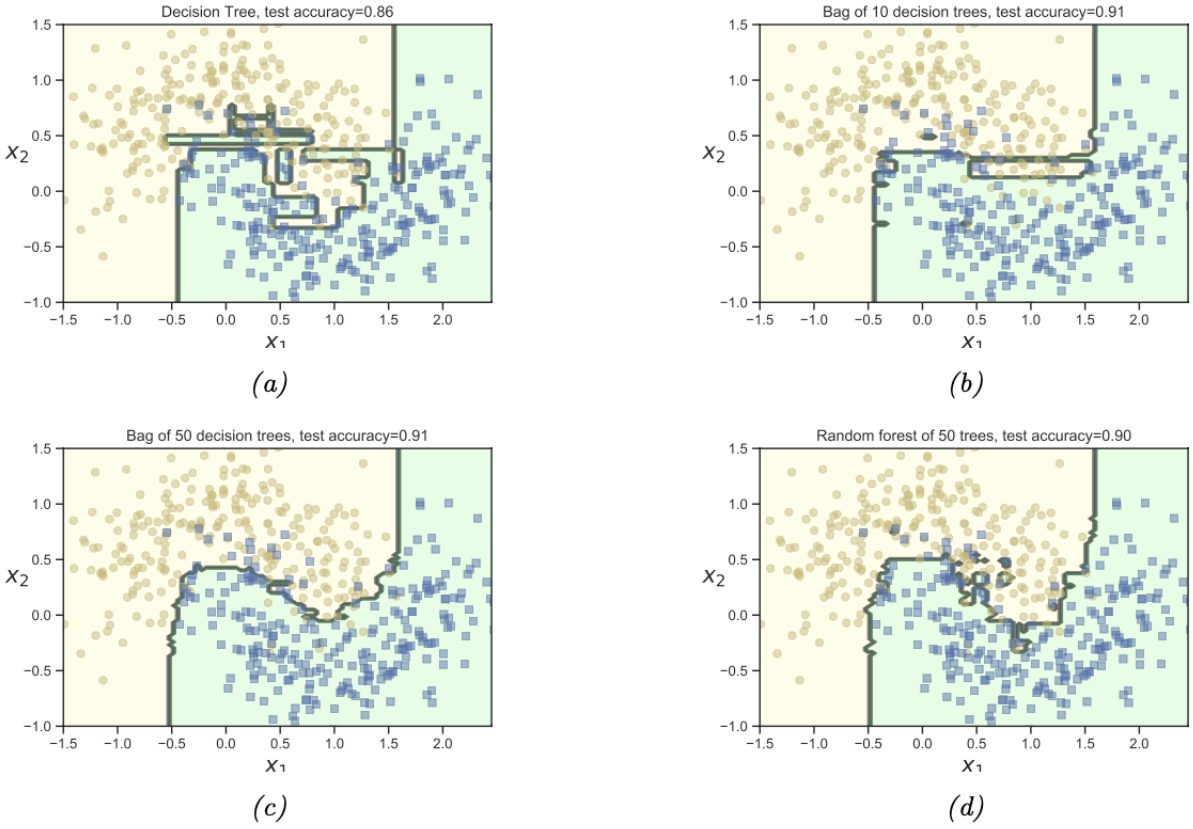


Figure 53: Comparison of the decision boundaries of a decision tree, bagged forest and random forest (taken from Murphy (2022))

Out-of-bag(OOB) error The generalisation error can be estimated by T-fold cross-validation as discussed in Section 5.2, but for each fold, a forest needs to be trained. Alternatively, the out-of-bag error does not involve additional forest training. Note that each data point $x^{(i)}$ is omitted in at least some of the bootstrap samples D_b. Define the set of *out-of-bag* samples as $\tilde{B}_i := \{b : (x^{(i)}, y^{(i)}) \notin D_b\}$; then $x^{(i)}$ automatically becomes a validation set for the trees $b \in \tilde{B}_i$. Let

$$\widehat{h}_{\mathrm{OOB}}(x^{(i)}) := \frac{1}{|\tilde{B}_i|}\sum_{b\in\tilde{B}_i} h^{(D_b)}(x^{(i)})$$

Then, the OOB error (empirical risk) is defined as

$$\widehat{R}_{\text{OOB}} := \sum_{i=1}^{n} L(y^{(i)}, \widehat{h}_{\text{OOB}}(x^{(i)}))$$

The effectiveness of the OOB error in estimating the generalization error depends on the size of out-of-bag samples $|\tilde{B}_i|$. For each bootstrap sample b, the probability of $x^{(i)}$ not being selected is $1 - 1/n$, and so the proportion of trees not using $x^{(i)}$ is roughly $\pi_{\text{OOB}}(n) := (1 - 1/n)^n$. As the sample size increases ($n \to \infty$), this proportion approaches $1/e \approx 0.367$. Therefore, $E[|\tilde{B}_i|] \to 0.367B$. This necessitates a large value of B (typically between 200 and 1000).

Random Forest First proposed by Breiman (2001), the random forest is akin to the bagging estimator, but with a key difference: each decision tree in the forest is trained using only a subset of $p_{\max}$ randomly chosen attributes (features) from the total p attributes. It has proven to be more effective than bagging.

Ensemble methods, such as building random forests, remain effective even if each estimator is individually ineffective (shallow, simple decision trees). Some decision trees may tend to overfit, while others may underfit. The practice of selecting random features helps prevent highly influential attributes from appearing in all decision trees, allowing other attributes to contribute to the forest. Consequently, the aggregated estimator in the forest captures various aspects of the dataset.

The number of trees B, minimum leaf size (or maximum tree depth), and the number of features $p_{\max}$ chosen for each tree are all hyperparameters that should be tuned before training. Fortunately, tuning is quite fast using the OOB error.

The drawbacks of random forests are:

- Computationally heavy: especially for large and high-dimensional datasets.
- Lack of interpretability: Instead of a single tree, we have many decision trees, and it is not clear what each tree represents and how they work together.

Importance of an attribute Despite the difficulty in interpretability, there is a way to identify the importance of an attribute(feature) x_j of interest: train the random forest in the usual way and calculate the loss on the test set. Then, "mess up" x_j by randomly permuting the attribute in the training set while keeping other attributes unchanged. More precisely, choose a permutation $\tau : \{1, \cdots, n\} \to \{1, \cdots, n\}$ and train the random forest on $\tilde{D} = \{(\tilde{x}^{(i)}, y^{(i)})\}_{i=1,\cdots,n}$ (where $\tilde{x}_j^{(i)} = x_j^{(\tau(i))}$ and $\tilde{x}_{-j}^{(i)} = x_{-j}^{(i)}$) instead. If x_j is critical for prediction, then the test error of the random forest trained on the perturbed dataset $\tilde{D}$ should be much higher.

9.3.2 Boosting

In general, ensemble methods use weak (in the sense of few training parameters and simple structure) prediction rules $\{h_t\}_{t=1,\cdots,T}$ and produce an aggregated prediction rule:

$$f(x) := \sum_{t=1}^{T} \beta_t h_t(x; \theta_t)$$

where β_t represents the weights and θ_t is the parameter of h_t. For random forests, $\beta_t = 1/T$, θ_t is the partition of the space.

The idea of boosting is that instead of training h_t independently, train in a sequential fashion: compute the training error of h_t for each data point $x^{(i)}$, and force the next predictor h_{t+1} to focus more on the points with poor performance, so that the next predictor learns something new from the data Robert E. Schapire (2012).

Forward Stage-wise additive boosting Define the aggregated prediction rule up to step t as

$$\widehat{f}_t(x) := \sum_{i=1}^{t} \widehat{\beta}_i h_i(x; \widehat{\theta}_i)$$

Given loss L, the *forward Stage-wise additive boosting* estimate the next prediction rule h_{t+1} and the next weight β_{t+1} by minimising the empirical risk on residuals $y^{(i)} - \widehat{f}_t(x^{(i)})$:

$$\widehat{\beta}_i, \widehat{\theta}_{t+1} = \arg\min_{\beta,\theta} \sum_{i=1}^{n} L\left(y^{(i)}, \widehat{f}_t(x^{(i)}) + \beta h_{t+1}(x^{(i)}; \theta)\right) \tag{9.1}$$

In the case of square loss, this method is called *L2-boosting* Bühlmann and Yu (2003).

AdaBoost For binary classification, recall from Section 7.3 that the 0-1 loss function for prediction rule $f(x)$ can be defined on $yf(x)$. AdaBoost is a forward Stage-wise additive boosting using the exponential surrogate loss function $\phi(z) = e^{-z}$. Substituting to equation 9.1 yields

$$\begin{aligned}
\widehat{\beta}_i, \widehat{h}_{t+1} &= \arg\min_{\beta,h} \sum_{i=1}^{n} \exp\left(-y^{(i)}(\widehat{f}_t(x^{(i)}) + \beta h(x^{(i)})\right) \\
&= \arg\min_{\beta,h} \sum_{i=1}^{n} w_{i,t} \exp\left(-y^{(i)} \beta h(x^{(i)})\right) \quad \text{where } w_{i,t} := \exp(-y^{(i)} \widehat{f}_t(x^{(i)})) \\
&= \arg\min_{\beta,h} \sum_{i=1}^{n} w_{i,t} e^{-\beta} I(y^{(i)} = h(x^{(i)})) + \sum_{i=1}^{n} w_{i,t} e^{\beta} I(y^{(i)} \neq h(x^{(i)})) \\
&= \arg\min_{\beta,h} e^{-\beta} \sum_{i=1}^{n} w_{i,t} \left(1 - I(y^{(i)} \neq h(x^{(i)}))\right) + e^{\beta} \sum_{i=1}^{n} w_{i,t} I(y^{(i)} \neq h(x^{(i)})) \\
&= \arg\min_{\beta,h} (e^{\beta} - e^{-\beta}) \sum_{i=1}^{n} w_{i,t} I(y^{(i)} \neq h(x^{(i)})) + e^{-\beta} \sum_{i=1}^{n} w_{i,t}
\end{aligned}$$

Therefore, the next prediction rule is $\widehat{h}_{t+1} = \arg\min_h \sum_{i=1}^{n} w_{i,t} I(y^{(i)} \neq h(x^{(i)}))$, which can be found by a weighted version of the weak learner. The weight $w_{i,t}$ is higher when $y^{(i)} \widehat{f}_t(x^{(i)})$ is negative and has a large absolute value (indicating strong misspecification for $x^{(i)}$), so this data point receives more attention in the next learner h_{t+1}.

As for β, for simplicity denote the error of predictor $\widehat{h}_{t+1}$ as $\widehat{\epsilon}_{t+1} := \sum_{i=1}^{n} w_{i,t} I(y^{(i)} \neq \widehat{h}_{t+1}(x^{(i)}))$, then

$$\widehat{\beta}_{t+1} = \arg\min_{\beta \in \mathbb{R}} (e^{\beta} - e^{-\beta}) \widehat{\epsilon}_t + e^{-\beta} = \frac{1}{2} \log\left(\frac{1 - \widehat{\epsilon}_{t+1}}{\widehat{\epsilon}_{t+1}}\right)$$

The full process of AdaBoost.M1 is shown in Algorithm 6. This algorithm generalizes to multi-class classification and has been implemented in the Python package Scikit-learn [https://scikit-learn.org/stable/auto_examples/ensemble/plot_adaboost_multiclass.html](Multi-class AdaBoosted Decision Trees).

Any other surrogate loss function can be used for boosting. For example, the logit surrogate loss is used in Friedman et al. (2000).

Algorithm 6 AdaBoost.M1 for binary classification

Input:

- Dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1,\cdots,n}$
- weak learner(binary classifier)
- total number of weak learners T

Initialise uniform weights $\overline{w}_{i,0} = 1/n$ for $i = 1, \cdots, n$

for $t = 1, \cdots, T$ **do**

find the weak learning by minimising the weighted empirical risk

$$\widehat{h}_t = \arg\min_h \sum_{i=1}^{n} w_{i,t-1} I(y^{(i)} \neq h(x^{(i)}))$$

compute

$$\widehat{\beta}_t = \log\left(\frac{1-\widehat{\epsilon}_t}{\widehat{\epsilon}_t}\right)$$

where $\widehat{\epsilon}_t := \sum_{i=1}^{n} w_{i,t-1} I(y^{(i)} \neq \widehat{h}_t(x^{(i)}))$

update the weights

$$w_{i,t} := \exp(-y^{(i)} \widehat{f}_t(x^{(i)})) = \overline{w}_{i,t-1} \exp(-y^{(i)} \widehat{\beta}_t \widehat{h}_t(x^{(i)}))$$

and normalise: $\overline{w}_{i,t} := w_{i,t} / \sum_j w_{j,t}$.

end for

return The aggregated predictor

$$h(x) = \text{sign}(\widehat{f}_T(x)) = \text{sign}(\sum_{t=1}^{T} \widehat{\beta}_t \widehat{h}_t(x^{(i)}))$$

Gradient Boost Friedman (2000) This algorithm performs gradient descent on the function space to find the prediction rule h that solves Eq. (9.1) instead of specifying a parametric form. The gradient is estimated using the dataset:

$$\frac{\partial L(y,f)}{\partial f} \approx g(f) := (g_1(f), \cdots, g_n(f))^T$$

where $g_i(f) := \frac{\partial L(y^{(i)}, f(x^{(i)}))}{\partial f(x^{(i)})}$. Since $g(f)$ is a finite-dimensional estimate, at each step of gradient boosting, a weak learner h_t is fitted to the negative gradient $-g(f)$:

$$h_t := \arg\min_h \sum_{i=1}^{n} \left(-g_i(\widehat{f}_{t-1}) - h(x^{(i)})\right)^2$$

It turns out that boosting methods produce strong learners as long as the accuracy of the individual weak learners is above 0.5. For a comprehensive introduction to various boosting methods and learning theory support, refer to the book by Robert E. Schapire (2012).

10 Ending

The auxiliary notes on Statistical Machine Learning finish here. Thank you very much for reading.
Please email daniel.kansaki@outlook.com or yuhang.lin@new.ox.ac.uk if you find any typo or have suggestions for improvements.

References

Alsheikhy, A. A., Said, Y. F., and Barr, M. (2020). Logo recognition with the use of deep convolutional neural networks. *Engineering, Technology & Applied Science Research*.

Breiman, L. (2001). Random forests. *Machine Learning*, 45:5–32.

Bühlmann, P. and Yu, B. (2003). Boosting with the l2 loss: Regression and classification. *Journal of the American Statistical Association*, 98:324–339.

Chavent, M. (1998). A monothetic clustering method. *Pattern Recognition Letters*, 19(11):989–996.

Cybenko, G. V. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314.

Domingos, P. M. and Pazzani, M. J. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29:103–130.

Duda, R., Hart, P., and G.Stork, D. (2001). *Pattern Classification*. Wiley Interscience, 2nd edition.

Friedman, J. (2000). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29.

Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors). *The Annals of Statistics*, 28(2):337 – 407.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202.

Geuchen, P., Jahn, T., and Matt, H. (2023). Universal approximation with complex-valued deep narrow neural networks.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA. PMLR.

Golubeva, A., Neyshabur, B., and Gur-Ari, G. (2021). Are wider nets better given the same number of parameters?

Grandvalet, Y. (2004). Bagging equalizes influence. *Machine Learning*, 55:251–270.

Jordan, M. I. (2009). Lecture 11 — february 25 11.1 surrogate loss functions.

Kingma, D. P. and Welling, M. (2019). An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392.

Kramer, M. (1992). Autoassociative neural networks. *Computers and Chemical Engineering*, 16(4):313–328. Neutral network applications in chemical engineering.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lippmann, R. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4(2):4–22.

Mansour, Y. (1997). Pessimistic decision tree pruning based on tree size. In *International Conference on Machine Learning*.

Moutafis, P., Leng, M., and Kakadiaris, I. A. (2017). An overview and empirical comparison of distance metric learning methods. *IEEE Transactions on Cybernetics*, 47(3):612–625.

Murphy, K. P. (2022). *Probabilistic Machine Learning: An introduction*. MIT Press.

Parkinson, S., Ongie, G., and Willett, R. (2023). Linear neural network layers promote learning single- and multiple-index models.

Parkinson, S., Ongie, G., Willett, R., Shamir, O., and Srebro, N. (2024). Depth separation in norm-bounded infinite-width neural networks.

Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., and Liao, Q. (2017). Why and when can deep – but not shallow – networks avoid the curse of dimensionality: a review.

Robert E. Schapire, Y. F. (2012). *Boosting: Foundations and Algorithms*. The MIT Press.

Sa-Couto, L., Ramos, J. M., Almeida, M., and Wichert, A. (2022). Understanding the double descent curve in machine learning.

Shaukat, S., Rao, T., and Khan, M. (2016). Impact of sample size on principal component analysis ordination of an environmental data set: Effects on eigenstructure. *Ekológia (Bratislava)*, 35.

Zeiler, M. D. and Fergus, R. (2013). Visualizing and understanding convolutional networks.