A.I. notes

Cooper

1 Linear Regression

- **Hypothesis** function

$$y = h(x) = \theta_0 + \theta_1 x$$
$$y = h(x) = \boldsymbol{\theta} \cdot \boldsymbol{x}$$

- **Loss function**

$$J(\theta_0.\theta_1) = \frac{1}{2}\frac{1}{m}\sum_{i=1}^{m}\Big(h(x^{(i)}) - y_i\Big)^2 = \frac{1}{2}\frac{1}{m}\sum_{i=1}^{m}\Big(\theta_0 + \theta_1 x^{(i)} - y_i\Big)^2$$
$$J(\theta_0, \theta_1, \ldots, \theta_N) = \frac{1}{2}\frac{1}{m}\sum_{i=1}^{m}\Big(\theta_0 + \sum_{k=1}^{N}\theta_k x_k^{(i)} - y_i\Big)^2$$

- **Calculate the Gradient** gradient is the direction along which the loss function increases the most

$$\nabla_\theta J_{(0)} = \begin{pmatrix} \frac{\partial J}{\partial \theta_0} \\ \\ \frac{\partial J}{\partial \theta_1} \end{pmatrix}_{(0)} \qquad \nabla_\theta J_{(0)} = \begin{pmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \\ \\ \vdots \\ \frac{\partial J}{\partial \theta_N} \end{pmatrix}_{(0)}$$

- **Update**

$$\boldsymbol{\theta}_{(1)} = \boldsymbol{\theta}_{(0)} - \alpha \nabla_\theta J_{(0)}$$
$$\boldsymbol{\theta}_{(n+1)} = \boldsymbol{\theta}_n - \alpha \frac{1}{m}\mathbf{X}^T(\mathbf{X}\boldsymbol{\theta}_n - \boldsymbol{y})$$

- **Repeat**.

$$\boldsymbol{\theta}_{(n+1)} = \boldsymbol{\theta}_{(n)} - \alpha \nabla_\theta J_n \qquad n = 0, 1, 2, \ldots$$
$$\text{Stop if} \quad \big|J_{(n+1)} - J_n\big| \leq \text{tol} \quad \text{or} \quad n + 1 \geq \text{ max}$$

1.1 Regularisation

A good machine learning model should have a good *bias-variance trade-off*. This can be achieved by minimising the generalisation error, which is the sum of the squared bias and the variance.

$$J(\theta_0.\theta_1) = \frac{1}{2}\frac{1}{m}\sum_{i=1}^{m}\Big(h(x^{(i)}) - y_i\Big)^2 + \lambda\sum_{j=1}^{N}\theta_j^2$$

If λ is small, large values of $\sum_{j=1}^{N}\theta_j^2$ are allowed, meaning that the model contains too many parameters, thus it can **overfit** (training error small, generalization error unacceptably large, high variance). If λ is large , only small values of $\sum_{j=1}^{N}\theta_j^2$ are allowed. This means that the training will reduce the number of parameters, this it might **underfit** (training error is too large, high bias).

2 Logistic Regression

- **Hypothesis function**, models the probability that a given set of features belongs to class

$$h(\mathbf{x}) = \sigma(\theta \cdot \mathbf{x}) = \frac{1}{1 + exp(-\theta \cdot x)}$$

- **Loss function** (cross-entropy), entropy measures uncertainty on the possible values of a prediction

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left\{y_i ln\Big[h(x^{(i)})\Big] + (1-y_i) ln\Big[1-h(x^{(i)})\Big]\right\}$$

- **Gradient descent** for logistic Regression:

  Choose a starting point ⇒ Calculate the gradient ⇒ Update and repeat.

- If encounter **multi-class**, perform the same as many times as the number of categories of logistic regressions, each of which splits the data into two classes. End up with the same number of Hypothesis functions, select the highest probability.

- Metrics for classification

  - Precision = $\frac{\text{True Positives}}{\text{True Positives+False positives}}$; Recall = $\frac{\text{True Positives}}{\text{True Positives+False negatives}}$

		Ground Truth		
		1	**0**	
Model Prediction	**1**	True Positive	False Positive	**Precision**
	0	False Negative	True Negative	
		Recall		

NOTE:

For a classification model, high variance → overfitting, high bias → underfitting.

- If the model classifies only yellow helicopters as a helicopter.

  - It is overfitting (high variance). This is because the model may be too complex and, thus, it fits features that are not relevant to the classification (the colour).

  - To help with high variance model, we can (1) add more data points to the training set, (2) simplify the hypothesis function, (3) regularize the optimization algorithm, (4) remove some features from the dataset.

- If the model classifies any round object as a helicopter.

  - It is underfitting (high bias). This is because the model is unable to learn more complex feature, such as the bladed, the helicopter tail, etc.

2.1 Supporting Vector Machines (SVMs)

In SVMs, we want to compute the **optimal linear decision boundary**, which is the **hyperplane** with the largest margin between the two classes.

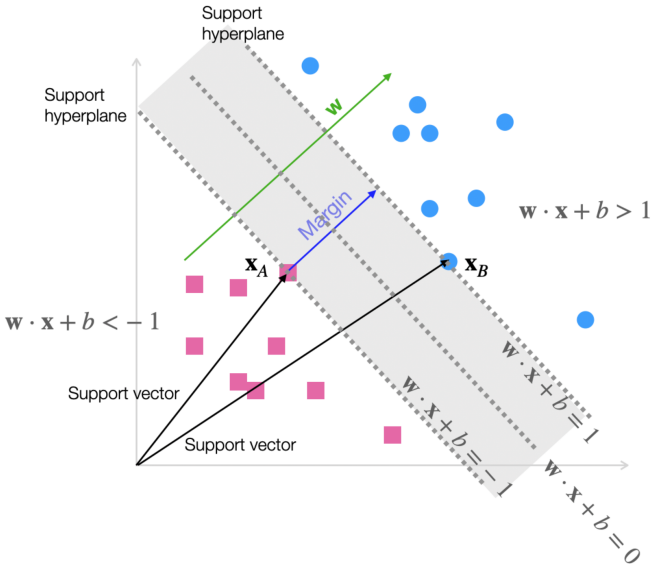
- Support hyperplanes (optimal hyperplane between two points)

$$\mathbf{w}\cdot\mathbf{x}+b=1$$
$$\mathbf{w}\cdot\mathbf{x}+b=-1$$

  How is the hyperplane with largest margin oriented can be found by solving an optimisation problem

$$\text{Find } \mathbf{w}, b \text{ such that } \frac{2}{\|w\|} \text{ is maximum}$$
$$\text{or } \min_{(w,b)} \frac{1}{2}\|w\|^2 \quad \text{subject to} \quad y_i(\mathbf{w}\cdot\mathbf{x}^{(i)}+b)\geq 1 \quad i=1,2,\ldots,m$$

- Finding the hyperplane that maximizes the margin between the classes is done by minimizing a cost function, which is a convex optimization problem. Convex optimization problems have a unique global minimum, which means that the solution found by SVMs is a **global minimum**.



2.1.1 Regularisation

if the data is not linearly separable, we need to introduce regularisation to allow some misclassified data points by introducing a slack variable $\varepsilon_i > 0$ such that: (i) $\varepsilon_i = 0$ for correctly classified (ii) $\varepsilon_i = \left|y_i - f(\mathbf{x}^{(i)})\right|$ otherwise.

$$\min_{(w,b,\varepsilon)} \frac{1}{2}\|w\|^2 + C\sum_i^m \varepsilon_i \quad \text{subject to} \quad y_i(\mathbf{w}\cdot\mathbf{x}^{(i)} + b) \geq 1 - \varepsilon_i \quad \varepsilon_i \geq 0$$

small C - soft (wide margin) $\quad C \to \infty$ - hard margin

NOTE:

If an SVM model is overfitting (high variance), you can (1) use regularisation - decreasing C, (2) adding more data, (3) reduce number of features, (4) removing outliers.

2.1.2 Kernel Trick

For nonlinearly separable data, one option is to enrich the data with nonlinearities. This can be achieved by mapping the data into a nonlinear higher-dimensional space, $\mathbf{x} \to \phi(\mathbf{x})$, where ϕ is a nonlinear map, for example, a polynomial $(x_1^{(i)}, x_2^{(i)}) \to (x_1^{(i)}, x_2^{(i)}, x_1^{(i)2} + x_2^{(i)2})$. Intuitively, by adding dimensions in the feature space, it is more likely that the data becomes linearly separated.

Kernel Trick decomposes the weight vector in the transformed feature space

$$\mathbf{w} = \sum_{j=1}^m \beta_j \boldsymbol{\phi}(\boldsymbol{x}^{(j)})$$

where β_j become the new weights to find. In other words, we express the weight vector with coordinates defined in the transformed feature space.

$$f(\boldsymbol{x}^{(i)}) = \sum_{j=1}^m \beta_j \boldsymbol{\phi}(\boldsymbol{x}^{(j)}) \cdot \boldsymbol{\phi}(\boldsymbol{x}^{(i)}) + b$$

we resort to a theorem of functional analysis, which ensures that there exists a function, K, such that

$$K(\boldsymbol{x}^{(j)} \cdot \boldsymbol{x}^{(i)}) = \boldsymbol{\phi}(\boldsymbol{x}^{(j)}) \cdot \boldsymbol{\phi}(\boldsymbol{x}^(i))$$

where K is the *kernel*. K suggested that we DO NOT need to increase the dimension of the feature space to include nonlinearities, we only need to apply the kernel to inner products in the original feature space. essentially finding optimal separating hyperplane without calculating anything about $\boldsymbol{\phi(x^{(i)})}$.

Kernel can also be explained as (i) A single-channel convolution filter, (ii) A transformation of the dataset to a higher dimension.

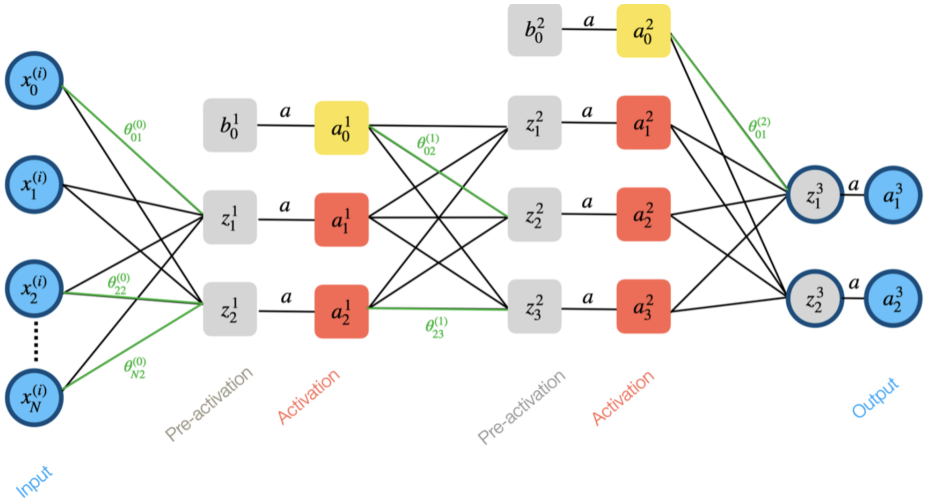
3 Neural Network

3.1 Feedforward Neural Network

Neural Network generalise the principle of linearly combining simple nonlinear functions

$$y \approx h(x : \theta)$$

function that maps input data "x" to output "y", with the function parameters represented by "θ".



3.1.1 Mathematical expression

- Input layer → First hidden layer

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} \Rightarrow \quad \theta^{(0)} = \begin{pmatrix} \theta^{(0)}_{0,1} & \theta^{(0)}_{0,2} \\ \theta^{(0)}_{1,1} & \theta^{(0)}_{1,2} \\ \vdots & \\ \theta^{(0)}_{N,1} & \theta^{(0)}_{N,2} \end{pmatrix}, \quad z^1 = \begin{pmatrix} b^1_0 \\ \theta^{(0)T} x \end{pmatrix}, \quad a^1 = a\{z^1\}$$

  where z is the *pre-activation*

- Second hidden layer

$$\theta^{(1)} = \begin{pmatrix} \theta^{(1)}_{0,1} & \theta^{(1)}_{0,2} & \theta^{(1)}_{0,3} \\ \theta^{(1)}_{1,1} & \theta^{(1)}_{1,2} & \theta^{(1)}_{1,3} \\ \theta^{(1)}_{2,1} & \theta^{(1)}_{2,2} & \theta^{(1)}_{2,3} \end{pmatrix}, \quad z^2 = \begin{pmatrix} b^2_0 \\ \theta^{(2)T} a^1 \end{pmatrix}, \quad a^2 = a\{z^2\}$$

- Output layer

$$\theta^{(1)} = \begin{pmatrix} \theta^{(2)}_{0,1} & \theta^{(2)}_{0,2} \\ \theta^{(2)}_{1,1} & \theta^{(2)}_{1,2} \\ \theta^{(2)}_{2,1} & \theta^{(2)}_{2,2} \\ \theta^{(2)}_{3,1} & \theta^{(2)}_{3,2} \end{pmatrix}, \quad z^3 = \theta^{(2)T} a^2, \quad h = a^3 = a\{z^3\}$$

  Or in generalised from

$$a^{L+1} = a\{\theta^{(L)T} a^L\}$$

3.1.2 Number of parameters

$$\text{n. of parameters} = \underbrace{(s_0+1)\times s_1}_{(N+1)\times 2} + \underbrace{(s_1+1)\times s_2}_{3\times 3} + \underbrace{(s_2+1)\times s_3}_{4\times 2} = (N+1)\times 2 + 17$$

$$= \sum_{i=0}^{L} (s_i+1)\times s_{i+1}$$

3.1.3 Forward propagation

forward propagation is the evaluation of the loss function from left to right during each iteration.

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\ln\left(h_k(\boldsymbol{x}^{(i)}) + (1-y_k^{(i)})\ln\left(1-h_k(\boldsymbol{x}^{(i)})\right)\right) + \frac{\lambda}{2}\sum_{l=0}^{L}\sum_{a=0}^{s_L}\sum_{b=1}^{s_{L+1}}(\theta_{a,b}^{(l)})^2\right]$$

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K}\left(h_k(\boldsymbol{x}^{(i)}) - y_k^{(i)}\right)^2 + \lambda\sum_{l=0}^{L}\sum_{a=0}^{s_L}\sum_{b=1}^{s_{L+1}}(\theta_{a,b}^{(l)})^2\right]$$

Because neural networks contains nonlinear functions, the loss functions are typically **non-convex**, a local minimum is therefore **not necessarily a global minimum**.

3.1.4 Training

Training dataset (60-80%), Validation (10-20%), and test (10-20%)

Initialise ⇒ Forward propagation ⇒ Backpropagation ⇒ Update ⇒ Repeat until convergence

3.2 Convolution Neural Network

Drawbacks from feed-forward neural networks

- textitfully connected layers not suitable for images: contains too many parameters
- need to *flatten* 2D data into 1D array — losing spatial information.

Generic process

- input → filter (convolutional layer)→ pooling layer → (repeat) → flatten → fully-connected layer
- Numbers of neurons in a convolutional layer

  $$N_d = \frac{W_d - F_d + 2P_d}{S_d} + 1$$

  where N_d is the number of neurons, W is the width of the input channel, F is the width of the filter, P is the width of the pooling, and S is the Stride number.
- Number of parameter in convolution layer l

  $$l = (F_x \times F_y \times \text{n. channels} + 1 \text{ bias}) \times K$$

  where K is the number of filters (also =number of feature maps)

3.3 Sequence Modelling

When both the data and its order contain information to perform the task, the points in the dataset are dependent on the position of other points in the dataset, e.g., speech recognition, DNA, time series. We lose key information if we shuffle the data.

3.3.1 Sequence modelling with feedforward neural network

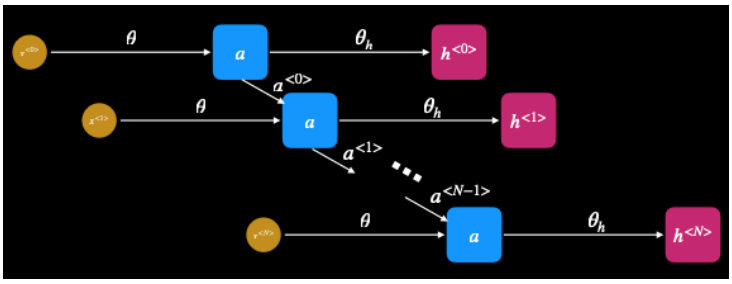
Consider time as **feature (inputs)**, and the physical variable as **label**, may be OK for short time series, but generally not a good approach due to:

- do not learn correlation as opposed to dynamical system having temporal correlations, and patterns.
- large number of parameters
- needs the inputs of a fixed size.

3.4 Recurrent Neural Network

Store the previous time-step and feed it to the next time step as the hidden state.

$$a^{<N>} = a(\theta_a^{<N-1>} + \theta x^{<N>} + b)$$
$$h^{<N>} = \theta_h a^{<N>} + b_h$$



- The weights and biases are shared across the time-step
- **time is not a feature**
- the loss function is a sum of the loss functions of every time step by applying chain rule across different time steps.

$$J = \sum_{i=0}^{N} J^{<N>}$$
$$\frac{\partial J}{\partial a^{<i>}} = \frac{\partial J}{\partial a^{<N>}} \frac{\partial a^{<N>}}{\partial a^{<N-1>}} \cdots \frac{\partial a^{<i+1>}}{\partial a^{<i>}} = \frac{\partial J}{\partial a^{<N>}} \underbrace{\theta_a \theta_a \dots \theta_a}_{N-i \text{ times}} = \frac{\partial J}{\partial a^{<N>}} \theta_a^{N-1}$$

- Unstable gradient occurs from above equation, when θ_a is small, the gradient will become smaller and leads to **vanishing gradient** problem, make it difficult for the network to remember information from previous time steps. This can be avoid by skipping some connections between layers an units, or adding a cell state to each recurrent cell.
- If θ_a is large, this leads to exploding gradient, which can be solved by clip (rescale) the gradient
- Recurrent neural network are not ideal for learning temporal correlations in long time series due to short-term memory.

NOTE:

- 3D tensor with shape **(batch_size, sequence_length, input_dim)**.
- **example** refers to a sequence of input data, where each element in the sequence is a time step, e.g., audio clip, where each time step is a small segment of the audio waveform.
  - In recurrent networks examples have to be input one at a time so that data in each forward propagation is still, effectively, one-dimensional
- **feature** refer to the characteristics of each time step or input element in the sequence, e.g. spectral representations of the audio waveform.
- **Windowing:** windowing refers to a technique of processing input data by using a sliding window over a sequence of data points. The window size determines how many data points are included in each window, and the sliding step determines how many data points the window moves forward for the next iteration. The purpose being:
  - Effectively increasing the number of examples.
  - Allowing to make predictions at any point during a sequence.
  - Reducing the amount of data analysed at each iteration.

3.4.1 Long-short time memory units LSTMs

Introducing a *cell state* C to convey (store) information from the far past – long term memory, with the combination of the same short-memory past.

- Consists of three gates to control the status of the states (forget, input, output). Each gate is controlled by a **sigmoid** activation function, due to the extreme nature of being either 1 or 0.
- **Forget gate.** This determines what percentage of the Long-term memory is remembered.
- **Candidate state** combines the shor-term memory and the input to create a **potential long-term memory** g. By using **tanh** to restrain the data in the range of -1 to 1.
- **Input gate** determines what percentage of the potential long-term memory is added to the long term memory C.
- **Output gate** calculates the potential long term memory from the cell state using a tanh function. Then, a sigmoid function is also used to determine what percentage of this potential long term memory is pass on as an output of the LSTM unit.

3.4.2 Gated recurrent units GRUs

- got rid of the cell state, and stored both long and short term memory in the **hidden state** and consists of 2 gates only.
- **Update gate** controls both the *input* and *forget* gate in determining what to forget and what to add on.
- **Reset gate** decides how much past information to forget.
- **Candidate state** contains past information.
- GRUs are simpler and with less parameters, making them easier to train than LSTMs

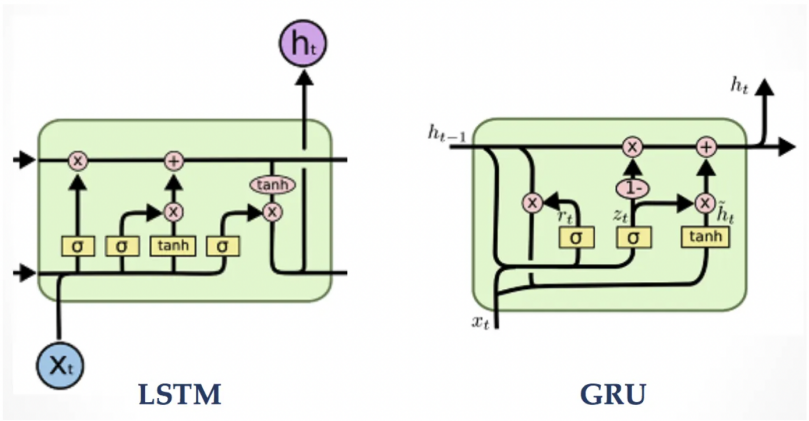


image by www.cc.gatech.edu

NOTE:

A GRU layer takes as input a 3D tensor of shape **(B, T, F)**, where B is the batch size, T is the number of timestep in each sequence, and F is the number of features in each timestep. It processes each timestep in the input sequence independently and generates an output for each timestep. Therefore, the output of the GRU layer will also be a 3D tensor of shape **(B, T, N)**, where N is the number of neurons in the layer.

Types of label and its output size:

- **Binary label:** For binary classification tasks, the output size of a neural network is usually 1, representing the probability of belonging to one of the two classes
- **Categorical label:** For multi-class classification tasks, the output size of a neural network is

usually equal to the number of classes.

- **One-Hot encoding label:** When using one-hot encoding for multi-class classification tasks, the output size of a neural network is also equal to the number of classes, each output is binary (0 1)

4 Unsupervised Learning

Methods that do not require labelled dataset. **Clustering** groups the instances based on their similarity without class labels.

4.1 k-Means clustering (Lloyd-Forgy algorithm)

Given a set of vector-valued data, $\mathbf{x}^{(i)}$, with the goal of grouping m observation into k clusters. k-means algorithm is iterative and is described as following:

- **Choose a metric (distance function)**. This quantitatively defines the notion of *similarity* by measuring the distance between points in the *feature space*. The distance are usually calculated using the squared Euclidean distance $\|\cdot\|^2$.
- **Choose number of clusters,** k
- **Choose the centroids** These are the baricentres (means), μ_j, of the clusters, representing the positions of the clusters.
- **Compute the distance** of each instance with respect to each centroid, $\|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_j\|^2$ for $i = 1, 2, \ldots, k$. **Assign** each instance $\boldsymbol{x}^{(i)}$ to the closest centroid j.
- **Compute the baricentre** of each centroid, $\mu_{j,new} = 1/N_j \sum_{j=1}^{N_j} \boldsymbol{x}^{(j)}$, where N_j is the number of points, $\boldsymbol{x}^{(j)}$, that belong in cluster j. Therefore, $\sum_{i=1}^{k} N_i = m$.
- **Update centroid**. The new centroid is the baricentre from the previous new mean.
- Compute the new distance, until the mean distance from the centroid stops changing.

Although k-means algorithm presents a fast and effective method to cluster data points, it has some limits: (i) though the algorithm is guaranteed to converge, it might not be the globally optimal solution, mainly because it depends on the initial centroid initialisation and numbers of cluster chosen. (ii) it noes not perform well when clusters have varying sizes.

There are several approaches to be taken to optimise the solution: (i) different centroid initialisation, (ii) number of clusters, a simple method is to plot the variance and the number of cluster, and choose the number of clusters at the inflection point (elbow). This is called the *elbow method*. (iii) Feature scaling, scale the input features before running k-means.

4.2 Density-based spatial clustering of applications with noise (DBSCAN)

A clustering algorithm that can handle nested clusters, and identifying nested algorithms at higher dimensions where we can't perceive by eye. DBSCAN groups points that are closely-packed, identifies the points that are isolated as outliers.

- **Choose a metric (distance function)** — Euclidean distance, $\|\cdot\|^2$
- **Choose the vicinity**, ε , to other points, i.e., the neighbourhood $\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|^2 \leq \varepsilon$. Choose N to define the number of points that are needed to classify a neighbourhood as high density.
- **Select a instance** and **Count** how many instances are located with distance ε from it.
  - If an instance has at least N in its ε, then it is considered as a *core instance*. All instances in ε of a core instance belong to the same cluster. This may also include other core instances.
  - If fewer that N belong in ε, it is identify as *Non-core* instance.
- **Classify** anomalies that are not core instances.

DBSCAN provides the advantage of not needing to specify the number of clusters, can find clusters with complicated shapes and can also identifies outliers. However, it does not cluster effectively when there are large differences in densities.

4.3 Hierarchical clustering

Hierarchical clustering does not required user input cluster numbers, it does it by inspecting the result. This section will be focusing on using **Bottom-up**, in which each instance defines a cluster a iteration 0. The algorithm then paris and merge clusters.

- **Choose a metric (distance function)** — Euclidean distance, $\|\cdot|^2$.
- **Compute** the distance between all the instances.
- **Merge** the closest pair into a new cluster, which is located midway its original instance.
- **Repeat** Step 1-2 with $m-1$ instances.
- **Stop** when left with one cluster.

4.4 Autoencoder

Autoencoders are a type of neural network used for unsupervised learning, **dimensionality reduction**, and **data compression**. Composed of two main parts:

- **Encoder:** takes the input data and maps it to a lower-dimensional representation. The lowest level is called the bottleneck.
- **Decoder:** takes this representation and maps it back to the original input data
- it can perform (i)**feature reduction** and (ii) **image correction** (e.g., removal of creases in scanned documents).

4.5 Elliptic envelope for anomaly detection

Using Gaussian density distribution as an approach to identify outliers.

4.5.1 Multivariate Gaussian distribution

A column vector of random variable (feature), $\boldsymbol{X} = [X_1, X_2, \ldots, X_N]^T$ is Gaussian distributed.

$$f_{\boldsymbol{x}}(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^N \det(\boldsymbol{\Sigma})}} \exp\Big(-\frac{\|\boldsymbol{x}-\boldsymbol{\mu}\|^2_{\boldsymbol{\Sigma}}}{2} \Big) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \qquad \boldsymbol{\mu} = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{x}^{(i)}$$

where $\boldsymbol{\mu}$ is the column-vector if the means and Σ is the **covariance matrix**, which can be computed from a sample, m (usually presented as matrix). Covariance matrix is **(1) symmetric** and **(2) positive**, which also represents the eigenvalues.

If the covariance matrix is **diagonal** (only entries on the diagonal), then both variables are **uncorrelated**. (different entries in the diagonal causes the circle to skew to ellipse). If the matrix is **non-diagonal** (full) and the principal axis of the ellipse is not parallel to the frame axis, then both variables are **correlated**

$$\boldsymbol{\Sigma} = \frac{1}{m-1}\sum_{i=1}^{m} (\boldsymbol{x}^{(i)} - \boldsymbol{\mu})(\boldsymbol{x}^{(i)} - \boldsymbol{\mu})^T$$

And finally the norm that measures the distance between the mean and a feature is

$$\|\boldsymbol{x} - \boldsymbol{\mu}|^2_{\Sigma} = (\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})$$

The inverse of the covariance matrix, Σ^{-1}, is also known as the *precision* matrix.

4.5.2 Elliptic envelope algorithm

elliptic envelope algorithm is a distance-based anomaly detection method that can be summarised as

1. **Assume** the datapoints to be Gaussian distributed
2. **Compute** mean and covariance (covariance matrix also defines the shape of the ellipse)
3. **Compute** the distance, D, between testpoints and the mean
4. **Choose** the decision boundary, c (the range of the elliptic)
5. **If** $D > c$, the test point is an outlier (anomaly).

5 Decision Tree

Decision tree contains three main components: (i) **Internal nodes** that provides classification criteria, (ii) **Branches** that listed out the features that satisfies the previous criteria, and (iii) **End nodes (leaves)** that is the final result for classification.

Noteworthy that when classifying between binary classes, the criteria can be set for e.g., $x_1 = 3$ where equal sign could be used. But when dealing with numerical values, **inequalities** must be used, $x \leq 3$.

5.1 Impurity Measures

Decision tree is constructed in the learning process node by node. The following nodes, or *children nodes*, for the partition are constructed with minimum impurity. Two impurity measure will be discussed here — *information gain, Gini diversity index*

- Information gain
  - information gain tries to find a suitable separation line in the features that can classify data in a more effective way. e.g. by shuffling the order of the feature: In this case, we can place

x_1	3	7	3	4	3	6	5	6
x_2	5	6	3	8	9	5	8	4
label	pass	fail	pass	fail	fail	pass	fail	fail

x_1	3	6	3	6	7	4	5	3
x_2	3	4	5	5	6	8	8	9
label	pass	fail	pass	pass	fail	fail	fail	fail

    a separation line at $x_2 \geq 6$ to better separate the fail by one node.

  - **Entropy** can be used to evaluate the difficulty of predicting category.

$$H(\text{label}) = -\sum_{i=1}^{n}(p_i log_2 p_i) = -\frac{3}{8}\log_2\frac{3}{8} - \frac{5}{8}\log_2\frac{5}{8} = 0.954$$

    for the above example, the entropy for the separate cases are:

$$H(\text{label}, x2 < 6) = H_{<6} = -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} = 0.81, \quad H(\text{label}, x2 \geq 6) = H_{\geq 6} = -\log_2 1 = 0$$

$$\therefore \quad H(\text{label}, x2) = 0.5H_{<6} + 0.5H_{\geq 6} = 0.405$$

    The aim is to find features with **highest information gain** (this feature will be split into children nodes to further). **IG** is **highest** when splitting the feature makes the biggest reduction in entropy.

$$IG(feature) = H(\text{label}) - H(\text{label}, feature) = 0.954 - 0.405 = 0.549$$

- Gini Impurity Criterion
  - How often a randomly chosen element from the set is incorrectly labelled. Higher GI, higher chance of misclassification

$$GI = 1 - \sum_{i=0}^{J} P_i^2$$

  - Split with the lowest weighted GI for the child trees to achieve largest reduction in GI.
- Shallow trees has less variance but higher bias. Deep trees has less bias but more prone to overfitting.

6 Principal Component Analysis (PCA)

finds an optimal direction (principle axis of the ellipsoid in gaussian distribution) of maximum variance. The maximum variance in the direction $\hat{w}_1$ is given by the maximum eigenvalue, λ_1, of the covariance matrix. Geometrically, this direction is the principal axis (eigenvector) of the ellipsoid, which is the direction along which the ellipsoid is most elongated.

The difference between PCA and linear regression is:

- Linear regression is sensitive on how you define input and output is large ($y = ax + b$ or $x = ay + b$), provides very different result. PCA treat variables as mathematically independent
- linear regression is sensitive to outliers
- The errors in PCA are orthogonal to the principal axis.