

Association Rule Mining → This technique finds relationship (rules) between items in large datasets, often used in market basket analysis to discover associations between products bought together.

Itemset: A collection of one or more items.

Support count: Frequency of occurrence of an itemset.

Support: Fraction of transactions that contain an itemset.

freq. Item: An itemset whose support  $\geq$  minsup threshold.

Association Rule: An implication expression of the form  $X \rightarrow Y$ , where  $X$  and  $Y$  are itemsets, representing that if  $X$  occurs,  $Y$  will likely occur as well.

Rule evaluation metrics: Support → fraction of transactions that contain both  $X, Y = \frac{|X \cup Y|}{N}$

Confidence → How often items in  $Y$  appear in transactions that contain  $X = \frac{c}{s}$

ARM task → Given a set of transactions  $T$ , the goal of association rule mining is to find all rules having  $[S \geq \text{minsup threshold}, C \geq \text{minconf threshold}]$

• Brute-force Approach: List all possible rules, compute their support and confidence, and prune those that don't meet the thresholds. However, this is computationally prohibitive as the number of potential rules grows exponentially.

• 2-Step approach: 1. freq. itemset generation whose support  $\geq$  minsup 2. Rule Generation gen. high conf. rules from each freq. itemset. Where each rule is a binary partitioning of freq. itemset.

• freq. itemset generation is still computationally expensive

Freq. Itemset gen. Brute-force approach: each itemset in the lattice is a candidate freq. itemset, count the support of each candidate by scanning the database.

→ Prune each transaction against every candidate, complexity  $\sim O(NM^2) \rightarrow$  Expensive since  $M = 2^d$

Computational Complexity - Total number of itemsets  $2^d$ , Total number of possible association rules  $R: 3^d - 2^{d+1} + 1$

fig. strategies: Reduce the number of candidates  $O(N)$  → Complete search:  $M^d$ , use pruning tech. to reduce  $M$

• Reduce the number of transactions  $(N)$  → Reduce size of  $N$  as the size of itemset increases, use by DHP and Vertical-based mining algorithms.

• Reduce the number of comparisons  $(NM)$  → Use efficient data structures to store the candidates or transactions, No need to match every candidate against every transactions.

Apriori Algorithm: if an itemset is freq., then all of its subsets must also be freq.  $\forall X, Y: X \subseteq Y \rightarrow s(X) \geq s(Y)$  anti-monotone → Support of an itemset never exceeds the support of its subset.

• Method: 1. Let  $k=1$ , generate freq. itemsets of length 1. 2. Repeat until no new freq. itemset are identified. [gen. length  $(k+1)$  candidate itemsets from length  $k$  freq. itemsets.

• Prune Candidate itemsets containing subsets of length  $k$  that are infreq. → Count the support of each candidate by scanning the DB, → Eliminate candidates that are infreq., leaving only those that are freq.

Clustering → is the task of grouping data points such that data points within the same group (cluster) are more similar to each other than to those in other clusters. The aim is to identify natural groupings in the data without any prior labelling.

K-means: The K-means algorithm divides data into  $k$  clusters by minimizing the within cluster variance. STEPS: ① Randomly select  $k$  initial centroids.

② Assign each data point to the closest centroid. ③ Recalculate the centroids based on the assigned points. ④ Repeat the process until convergence.

→ stopping Criteria: Convergence occurs when there is no change in the membership of clusters or when the Squared error falls below a threshold.

• Pros: Low computational complexity. Cons: Sensitive to initial centroid selection and outliers. Clusters may not be spherical or of equal size.

Similarity measures: → Using difference measures for clustering can yield difference cluster, → edit distance and correlation distance are most common choice of similarity measure for microarray data

• Why validity of clusters? Given some data, any clustering algorithm generates clusters, we need to make sure the clustering results are valid and meaningful.

• Measuring the validity of clustering usually involve: Optimality of cluster, Verification of biological meaning of clusters.

• Optimal cluster should minimize distance within clusters, maximize distance between clusters.

Euclidean distance  $d(x_1, y_1) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$  Manhattan distance  $d(x_1, y_1) = \sum_{i=1}^n |x_i - y_i|$   
correlation distance  $\frac{cov(x, y)}{\sqrt{var(x) \cdot var(y)}}$  Var  $(x) = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}$  Cov  $(x, y) = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{n-1}$

**Classification:** is a supervised learning problem where the goal is to assign a label to input data.  
**Artificial Neural Networks (ANN):** are computational models inspired by human brain, consisting of layers of nodes (neurons) that transform input data into output through various activation functions and weights.

**Dense layer:** Neurons are fully connected, that is, each neuron is connected to every neuron in the previous and/or next layers.

**Sigmoid:**  $\sigma(x) = \frac{1}{1 + e^{-x}}$     **tanh:**  $\tanh(x)$     **ReLU:**  $\max(0, x)$     **Leaky ReLU:**  $\max(x, 0.1x)$

**How Networks learn:** The backpropagation algorithm: Learning from experience. A neural network adjusts the strengths, i.e. weights, of every connections to approximate a mapping from input to output.

**Loss:** function to calculate errors, Comparing Actual output with ground truth. Metric: measure current performance of the artificial neural network.

**Hyperparameters:** Activation function (Hidden nodes)  $\rightarrow$  ReLU, Output Nodes Binary (Sigmoid) Multi-class softmax, Loss func. cross entropy, Optimizer SGD, RMSprop, Adam, Learning Rate (0.001, 0.1, ...), Epochs, Batch size, Validation set (0.1), Evaluation metrics accuracy.

**Entropy (S):**  $S = -\sum_{i=1}^n p_i \log p_i$  How it all fits together  $\rightarrow$  Neuron network library written in Python. A high-level interface that focuses on user-friendliness, modularity, and extensibility.

**Neural architecture of connecting neurons:** keras\_model\_sequential  $\rightarrow$  Inside a neuron: how it transforms input  $\rightarrow$  output.

- Keras::evaluate  $\rightarrow$  Parallel  $\rightarrow$  Labeling the unknown  $\rightarrow$  Keras::compile  $\rightarrow$  How neural network learn. - Keras::fit  $\rightarrow$  Training without passing the entire set to a network at once.

**Fitting parameters:** A network processes data in batches. Each batch contains batch\_size examples. Weights are adjusted not after every sample, but once after each batch (epoch). One epoch is when the entire training set is passed forward and backward through the network once. Given batch\_size, steps\_per\_epoch, use of training computer batch size.

**Why there is no ordinal relationship in this categorical variable:** we don't want our ml algorithm to understand and harness essentially non-existing ordinal relationship, which naturally exists in integer values.

**Flatten type:** Remove all dimensions of a tensor, transforming it into one long 1D tensor.

**CNN:** Convolutional Neural Network  $\rightarrow$  specialized neural networks that apply a convolution operation in place of general matrix multiplication.

Layers: Convolution, ReLU, Pooling, Fully Connected.

**Cross-correlation:** kernel slide size  $\rightarrow$   $2 \times 2$   $\rightarrow$   $G(\text{output image}) = G(i, j) = \sum_{u=0}^{K-1} \sum_{v=0}^{K-1} K(u, v) \cdot I(i-u, j-v)$   $\rightarrow$  not Repeat. ① Sum the individual product.

Steps: ① Slide the center element of the kernel so that it lies on top of the  $(u, v)$  element of  $I$ . ② Multiply each weight in the kernel by the pixel of  $I$  underneath.

**Convolution:** same cross-correlation, except that the kernel is flipped.  $\rightarrow$  ① Relate to the kernel flip. ② Relate to the kernel flip. ③ Relate to the kernel flip. ④ Relate to the kernel flip.

⑤ Slide the center element of the convolution kernel. ⑥ Multiply each weight in the rotated convolution. ⑦ Sum the individual product.

**Zero padding:** the process of symmetrically adding zeroes to the input matrix  $\rightarrow$  output size  $(0) = \frac{W - K + 1}{2} + 1$   $\rightarrow$   $W$ : input size,  $K$ : kernel size,  $P$ : padding,  $S$ : stride.  $\rightarrow$  Reduces computational cost and helps prevent overfitting.

**ReLU Layer:** Applies the ReLU activation func. which replaces all negative values in the feature map with zero  $\rightarrow$   $\max(0, x)$ .

**Pooling Layer:** used to reduce the spatial dimensions of the feature maps, effectively performing down-sampling; Max-Pooling, Avg-Pooling.  $\rightarrow$  Reduces computational cost and helps prevent overfitting.

**Fully Connected Layer:** After feature extraction through convolution, ReLU, and pooling, this layer is used to map the extracted features to the final output, which could be a class label.

**Dropout:** Dropping out units  $\rightarrow$  Removing from the network, along with all its incoming and outgoing connections.

**Can training backpropagation with gradient descent - batch full in training - stochastic (1 training) - Mini-batch (1-100)**

**Autoencoders:** specific type of feed-forward neural network where the input is the same as the output, they compress the input into a lower-dimensional code (latent space representation).

Encoder: the part of the network that compresses the input into a latent space representation. Any parameters not in training autoencoder.

Code: Represents the compressed input that is fed to the decoder. Code size, No. of layers, No. of nodes per layer, Loss function. Disables.

Decoder: Attempts to reconstruct the input from the latent space representation. Application: Dimensionality reduction, Information retrieval, Image reconstruction.

**GANs: Generative Adversarial Networks**  $\rightarrow$  class of machine learning models that consist of a module (generator) generates fake data, Discriminator distinguishes the real from the fake data.

The generator creates fake examples that are indistinguishable from real data, while the Discriminator attempts to identify whether the data is real or fake. Both models are trained simultaneously in a process that improves the performance of both.

**Training the Generator**

1. Random noise vector  $z$  is again pass through the G to produce a fake example.
2. The Discriminator classifies the generated example.
3. The classification error is computed, and the generator's parameters are updated using backpropagation to minimize the Discriminator's error.

**Training the Discriminator**

1. A random real example  $x$  is selected from training dataset.
2. A random noise vector  $z$  is generated and pass through the G to produce a fake example.
3. Discriminator is used to classify both the real and fake examples.
4. The errors in classification are computed, and the discriminator's parameters are updated using backpropagation to minimize classification errors.

**Reinforcement Learning**: A type of machine learning where an agent interacts with an environment, taking actions that yield rewards. The agent's goal is to learn the best policy that maximizes its total reward over time. The agent uses trial and error to learn from its experiences in the environment.

State-action, action-reward

**Policy ( $\pi$ )**: The agent's behavior by mapping states to actions.  $\pi: S \rightarrow A$ . The goal is to find the optimal policy that maximizes cumulative reward.

**Value func.**: Value of a state  $s_t$  is the expected cumulative reward that will be received while the agent follows the policy, starting from state  $s_t$ .

ways that we can compute the value: the state-value function ( $V^\pi(s)$ ), the state-action value function,  $Q^\pi(s, a)$

We can associate a value with each state: for a fixed policy, how good is it to run policy  $\pi$  from that state?

We can define value without specifying the policy: specify the value of taking action  $a$  from state  $s$  and then performing optimally.

$V^\pi(s) = R(s, \pi(s)) + \gamma V^\pi(\pi(s))$ ,  $Q^\pi(s, a) = R(s, a) + \gamma \max_{a'} Q^\pi(s, a')$

Finding the policy: If we have the value func., then finding the best policy is easy. We're looking for the optimal policy ( $\pi^*$ ). Optimal policy defined.

The easiest way to learn the optimal policy is to learn the optimal value func.  $1^a$ .

Discount Rate: (consider this MDP, no. of steps is now unlimited. no. of steps, value of  $s_t$  and  $s_{t+1}$  is definite for some policies).

We can introduce a term into the value func. to get around the problem of infinite value = called discount rate ( $\gamma$ )

Q-learning iteratively approximates the state-action value func.  $Q(s, a)$ , learns the value func. and policy simultaneously.

① Initialize  $Q(s, a)$  to small random values,  $V(s)$ , ② Observe state,  $s$ , ③ Pick an action,  $a$ , and do it. ④ Observe next state  $s'$  and reward

⑤  $Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha (r + \gamma \max_{a'} Q(s', a'))$  ⑥ go to ②

Action Selection: At each state of the RL process, the algorithm looks at the actions that can be performed in the current state and computes the value of each action.

$\hookrightarrow$  greedy Pick the action that has the highest value of  $Q(s, a)$ .  $\epsilon$ -greedy similar to the greedy algorithms, but with some small probability  $\epsilon$  we pick some other action at random. So nearly every time we take the greedy option, but occasionally we try an alternative in the hope of finding a better action.

$Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha (r + \gamma \max_{a'} Q(s, a'), Q(s, a), Q(s, a'))$

Deep Reinforcement Learning = Deep learning + Reinforcement Learning, Use deep neural networks to represent Value function, Policy, Model

Deep Q-network training ① Do a feedforward pass for the current state  $s$  to get predicted  $Q$ -values for all actions.

② Do a feedforward pass for the next state  $s'$  and calculate maximum overall network outputs

③ Set  $Q$ -value target for action to  $r + \gamma \max_{a'} Q(s', a')$  use the max val in ②

④ Update the weights using backpropagation.

**SOMs**: type of Artificial neural network based on competitive learning and are used to create topological maps that represent data in a lower-dimensional space, preserving the spatial relationships. Input layer: input variables, Output layer: composed of neurons arranged in a one-dimensional or two-dimensional grid. Each neuron has a weight vector of the same dimensionality as the input vector.

① Initialization: The weight vectors are initialized with small random values, and a small positive learning rate  $\eta$  is assigned.

② Activation and similarity matching: The input vector  $x$  is applied to the network, and the winning neuron is identified using the min. Euclidean distance criterion.

③ Learning: Update the synaptic weights, where the weight update depends on the learning rate and the neighborhood of the winning neuron

④ Iteration

$\rightarrow$  Increase iteration  $y$  by one, go back to ② and continue until the minimum-distance criterion is satisfied, or no noticeable changes occur in the feature map.