Introduction to Machine Learning Summary

Machine Learning Concepts:

three ingredients of ML: data, models, learning; ML problem: regression, classification, clustering, anomaly detection and association rule; what are instances, attributes, concepts; supervised learning vs unsupervised learning; training data vs test data; instance is represented as feature vectors; possible attribute types: nominal, ordinal, continuous.

Probability Theory:

Probability of an event: fraction of times the event is true in independent trials(range of 0 to 1);

Joint probability: probability of two events occurring concurrently.

Property: $P(A,B) = P(A) \cdot P(B)$ iff A and B are independent.

Conditional probability: probability of an event given another event occurring.

Property: $P(A|B) = \frac{P(A,B)}{P(B)}$; $P(A,B) = P(A|B)P(B) = P(B|A)P(A)$

Disjoint event: $P(A \text{ or } B) = P(A) + P(B)$ if $P(A,B) = \emptyset$.

Bayes rule: $P(A|B) = \frac{P(A)P(B|A)}{P(B)}$

Terms: $P(A)$: prior probability; $P(B|A)$: likelihood; $P(B) = \sum_A P(A)P(B|A)$: evidence; $P(A|B)$: posterior probability.

Binomial distribution: probability of an event with probability of p occurring m out of n times is:

$$P(m,n,p) = \binom{n}{m} p^m (1-p)^{n-m} = \frac{n!}{m!\,(n-m)!} p^m (1-p)^{n-m}$$

Multinomial distribution: probability of an event with probabilities of different outcomes $p_1, p_2, \cdots p_n$ occurring exact $x_1, x_2, \cdots x_n$ times is:

$$P(X_1 = x_1, X_2 = x_2, \cdots X_n = x_n; \mathbf{p}) = \frac{(\sum_i x_i)\,!}{x_1!\,x_2! \cdots x_n!} \prod_i p_i^{x_i}$$

Marginalization: probability of one event irrespective of the outcome of another event.

$$P(A) = \sum_{b \in \mathcal{B}} P(B)P(A|B)$$

Maximum likelihood estimate: find parameter set that maximizes the likelihood of the data.

Maximum posteriori estimate: find parameter set that maximizes the posterior distribution.

Expectation: the weighted average of all possible outcomes.

discrete variable:

$$E[f(x)] = \sum_{x \in X} f(x)P(x)$$

continuous variable:

$$E[f(x)] = \int_X f(x)P(x)dx$$

Optimization:

What is learning: find a set of model parameters that optimize the performance of a model.

How to find maxima/minima: ① define the objective function $f(\theta)$; ② compute the first derivative with respect to parameter θ; ③ set the derivative to zero; ④ solve for θ.

Multiple parameters: If a model has multiple parameters to be optimized, we need to compute the

partial derivative with respect to each parameter θ_i, which is $\frac{\partial f}{\partial \theta_i}$, and the updating phrase is done for all parameters. The recipe of gradient descent for multiple parameters:

1: Define objective function $f(\theta)$
2: Initialize parameters $\{\theta_1^{(0)}, \theta_2^{(0)}, \theta_3^{(0)}, \dots\}$
3: **for** iteration $t \in \{0, 1, 2, \dots T\}$ **do**
4: Initialize vector of *gradients* $\leftarrow$ []
5: **for** parameter $f \in \{1, 2, 3, \dots F\}$ **do**
6: Compute the first derivative of f at that point $\theta_f^{(t)}$: $\frac{\partial f}{\partial \theta_f^{(t)}}$
7: append $\frac{\partial f}{\partial \theta_f^{(t)}}$ to *gradients*
8: **Update all** parameters by subtracting the (scaled) gradient

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \frac{\partial f}{\partial \theta^{(t)}}$$

Constraint optimization: the parameter we are about to optimize is constrained to certain range, such that:

$$\hat{\theta} = \arg\min_{\theta} f(\theta)$$

$$\text{s.t. } g(\theta) > 0$$

By combining Lagrange multiplier λ, we have the constraint objective function:

$$\mathcal{L}(\theta, \lambda) = f(\theta) + \lambda g(\theta)$$

Naïve Bayes:

Key idea: the objective is to find a class label $\hat{y}$ that maximizes conditional probability $p(y|x)$ and reformulate the equation by using Bayes rule:

$$p(y|x) = \frac{p(y)p(x|y)}{p(x)}$$

Thus, our objective function is:

$$\hat{y} = \arg\max_{y \in Y} p(y|x)$$

$$= \arg\max_{y \in Y} \frac{p(y)p(x|y)}{p(x)}$$

$$= \arg\max_{y \in Y} p(y)p(x|y)$$

Naïve Bayes assumption: ① the features are assumed to be independent given a class label y;

$$p(y)p(x_1, x_2, \cdots x_M|y) = p(y)\prod_{m=1}^{M} p(x_m|y)$$

② instances are independent each other; ③ the distribution of training set and test set is the same.

For categorical feature value, we calculate the conditional probability $p(x_m|y)$ by counting.

Categorical Naïve Bayes:

Algorithm 3 Generative Story of Categorical Naive Bayes

1: **for** Observation $i \in \{1, 2, ...N\}$ **do**
2: Draw the label $y^i \sim Categorical(\phi)$
3: **for** Feature $m \in \{1, 2, ...M\}$ **do**
4: Draw feature value $x_m^i | y^i \sim Categorical(\phi_{y^i})$

Maximum likelihood estimate to calculate $p(y)$ and $p(x_m|y)$.

$$p(y_i) = \frac{|y_i|}{N}$$

$$p(x_m|y_i) = \frac{|y_i \wedge x_m|}{|y_i|}$$

For continuous feature value, we calculate the conditional probability $p(x_m|y)$ by using Gaussian distribution function: $\frac{1}{\sqrt{2\pi\sigma_y^2}} e^{\frac{-(x_m-\mu_y)^2}{2\sigma_y^2}}$

Gaussian Naïve Bayes:

Algorithm 2 Generative Story of Gaussian Naive Bayes

1: **for** Observation $i \in \{1, 2, ...N\}$ **do**
2: Draw the label $y^i \sim Categorical(\phi)$
3: **for** Feature $m \in \{1, 2, ...M\}$ **do**
4: Draw feature value $x_m^i | y^i \sim N(\mu_{y^i}, \sigma_{y^i})$

Handling zero probability: ① epsilon smoothing: replace 0 with a very small number ϵ, and we choose a class with the smallest number of ϵ. ② Laplace smoothing with probability:

$$P(x_m = j|y = k) = \frac{\alpha + count(y = k, x_m = j)}{M\alpha + count(y = k)}$$

It will reduce variance but increase bias.

Handling missing value in a test instance: ignore those attributes with missing value.

Log transformation: to avoid numerical underflow issue, it will typically transform original probability to log-probability, such that:

$$\hat{y} = \arg\max_{y \in Y} p(y) \prod_{m=1}^{M} p(x_m|y)$$

$$= \arg\max_{y \in Y} [log\, p(y) + \sum_{m=1}^{M} log\, p(x_m|y)]$$

Evaluation Metrics:

Accuracy = $\frac{TP+TN}{TP+FP+TN+FN}$ Precision = $\frac{TP}{TP+FP}$ Recall = $\frac{TP}{TP+FN}$ F1-score = $\frac{2PR}{P+R}$

Train-test split technique: holdout: randomly partition instances into training and test instances with the fixed portion(e.g. 70-30); repeated random subsampling: perform holdout multiple times and average the performance of these models; cross validation: partition data into m splits, and use $m-1$ splits for training and the rest as test set iteratively, and finally average the performance; stratification: the train-test set partition must satisfy that the data has the same distribution.

Multi-class evaluation:

① macro-averaging:

$$\text{Precision}_M = \frac{\sum_{i=1}^{c} \text{Precision}_i}{c}$$
$$\text{Recall}_M = \frac{\sum_{i=1}^{c} \text{Recall}_i}{c}$$

② micro-averaging:

$$\text{Precision}_\mu = \frac{\sum_{i=1}^{c} TP_i}{\sum_{i=1}^{c} TP_i + FP_i}$$
$$\text{Recall}_\mu = \frac{\sum_{i=1}^{c} TP_i}{\sum_{i=1}^{c} TP_i + FN_i}$$

③ weighted averaging:

$$\text{Precision}_W = \sum_{i=1}^{c} \text{Precision}_i\left(\frac{n_i}{N}\right)$$
$$\text{Recall}_W = \sum_{i=1}^{c} \text{Recall}_i\left(\frac{n_i}{N}\right)$$

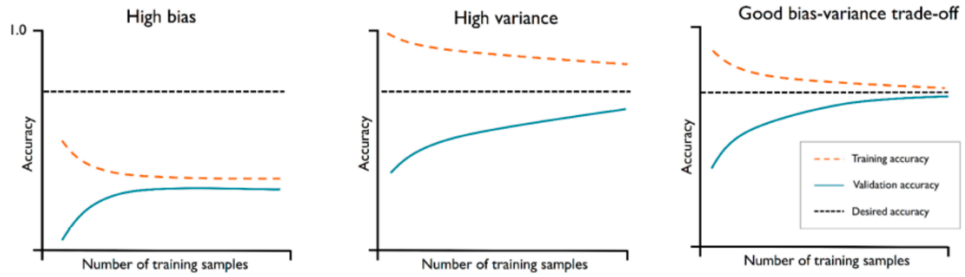
What is baseline and benchmark; Zero-R baseline: classify all instances to the most common class in the training data; One-R baseline: select one attribute with the smallest error rate.

What is underfitting(too simple) and overfitting(too complex); What is learning curve;

$$\text{Error} = \text{Bias}^2 + \text{Variace} + \text{Irreducible Error}$$

What is bias, variance and noise, bias-variance tradeoff.

Learning curve:



High bias remedy: ① use more complex model; ② add features; ③ boosting.

High variance remedy: ① add more training data; ② reduce features; ③ reduce model complexity; ④ bagging.

KNN:

Eager learning vs lazy learning: KNN is a lazy learning algorithm that we do not need to train a model, but compare test instances with training instances.

4 steps of KNN classifier: ① store training instances; ② measure distance between test and training instances; ③ find k-nearest neighbors; ④ return the class of the testing instances by majority voting.

Measurement of distance for different types of attributes:

① Nominal:

$$d = \frac{m - k}{m}$$

where m and k is the number of features and the number of matched features.

② Ordinal: we need to normalize these features, since the order matters

$$z = \frac{r-1}{M-1}$$

where M is the number of features, and $r \in \{1, \cdots, M\}$ is the corresponding rank of a feature value.

③ Numerical:

- Euclidean distance:

$$d(A,B) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

- Manhattan distance:

$$d(A,B) = \sum_{i=1}^{n}|a_i - b_i|$$

- Cosine similarity:

$$\cos(A,B) = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i} \cdot \sqrt{\sum_{i=1}^{n} b_i}}$$

Weighted KNN: classify a test instance according to the weighted accumulative class of KNN training instances.

Weighted strategies:

- Inversed linear distance:

$$w_j = \frac{d_k - d_j}{d_k - d_1}$$

where d_k is the maximum distance between neighbors and the test instance, d_1 is the minimum distance between neighbors and the test instance, and d_j is the distance between j-th neighbor and the test instance.

- Inversed distance:

$$w_j = \frac{1}{d_j + \epsilon}$$

where ϵ is the small constant that is to avoid zero denominator.

Breaking tie techniques:

- Random tie breaking
- Choose one with highest prior probability
- See if the $k+1$ th instance breaks the tie

Choice of k: smaller k would have lower performance; higher k would have higher performance but high computational overhead.

Logistic Regression:

Discriminative model: a model that optimizes $p(y|x)$ directly.

Naïve model: use linear combination of parameters $[\theta_0, \theta_1, \cdots \theta_m]$ and features to make a prediction:

$$p(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_m x_m$$

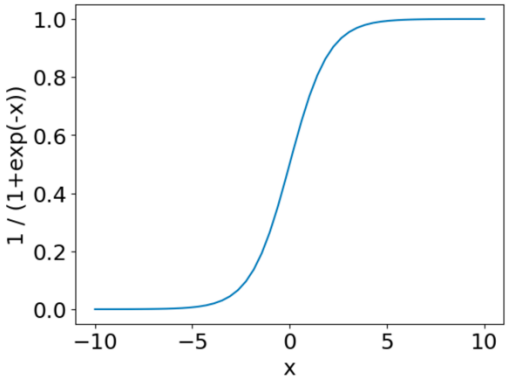
However, the linear combination is not a promising model to predict the label accurately.

Odds: the fraction of success over the fraction of failure.

$$\text{odds} = \frac{p(\text{success})}{1 - p(\text{success})}$$

Sigmoid function:

$$\sigma = \frac{1}{1 + e^{-x}}$$



Objective function:

$$\mathcal{L}(\theta) = -\prod_{i=1}^{N} p\left(y^{(i)} | x^{(i)}; \theta\right)$$

$$= -\prod_{i=1}^{N} \left(\sigma(\theta^T x^{(i)})\right)^{y^{(i)}} \cdot \left(1 - \sigma(\theta^T x^{(i)})\right)^{1-y^{(i)}}$$

$$log\, \mathcal{L}(\theta) = -\, log \sum_{i=1}^{N} y^{(i)} \sigma\left(\theta^T x^{(i)}\right) + \left(1 - y^{(i)}\right)\left(1 - \sigma\left(\theta^T x^{(i)}\right)\right)$$

Gradient of objective function:

$$\frac{\partial\, log\, \mathcal{L}(\theta)}{\partial \theta_j} = \sum_{i=1}^{N} \left(\sigma\left(\theta^T x^{(i)}\right) - y^{(i)}\right) \cdot x_j^{(i)}$$

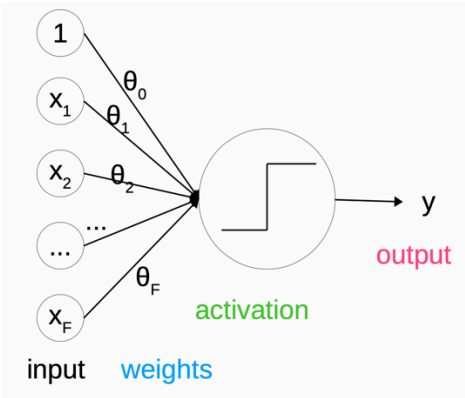
Softmax:

$$p(y = c | x; \theta) = \frac{e^{\theta_c^T x}}{\sum_k e^{\theta_k^T x}}$$

Perceptron:

Motivation: biological imitation of neurons

Structure:



Step function:

$$f(\theta^T x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Perceptron algorithm:

$D = \{(\mathbf{x}^i, y^i) | i = 1, 2, ..., N\}$ the set of training instances
Initialise the weight vector $\theta \leftarrow 0$
t $\leftarrow$ 0
repeat
  t $\leftarrow$ t+1
  for each training instance $(x^i, y^i) \in D$ **do**
    compute $\hat{y}^{i,(t)} = f(\theta^T x^i)$
    if $\hat{y}^{i,(t)} \neq y^i$ **then**
      for each each weight θ_j **do**
        update $\theta_j^{(t)} \leftarrow \theta_j^{(t-1)} + \eta(y^i - \hat{y}^{i,(t)})x_j^i$
    else
      $\theta_j^{(t)} \leftarrow \theta_j^{(t-1)}$
until tired
Return $\theta^{(t)}$

The convergence of perceptron algorithm depends on the ① parameter initialization; ② learning rate; ③ data to be split (non-linearly separatable data is not guaranteed to convergence).

Online learning vs batch learning.

Neural Network:

Three types of layers: input layer, hidden layer, output layer

Structure of neural network: each layer is fully connected with the neighboring layer, each neuron has an activation function that enables model performing non-linear tasks.

Activation function: ReLU, tanh, sigmoid(binary classification), softmax(multi-class classification)

Loss function:

- Regression: mean of square error.

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

- Binary classification:

$$\mathcal{L} = -\sum_{i=1}^{N} y^{(i)}\sigma\left(\theta^T x^{(i)}\right) + \left(1 - y^{(i)}\right)\left(1 - \sigma\left(\theta^T x^{(i)}\right)\right)$$

- Multi-class classification:

$$\mathcal{L} = -\sum_{i=1}^{N}\sum_{j=1}^{M} y_j^{(i)} \log \hat{y}_j^{(i)}$$

Forward propagation:

$$h^{(1)} = \phi^{(1)}(\theta^{(1)^T} x)$$
$$h^{(2)} = \phi^{(2)}(\theta^{(2)^T} h^{(1)})$$
$$y = \phi^{(3)}(\theta^{(3)^T} h^{(2)})$$

Backpropagation: suppose the activation function is sigmoid function for hidden layers and mean square error for output layer.

- Partial derivative w.r.t. the parameter of the output layer:

$$\frac{\partial \mathcal{L}}{\partial \theta_{k1}^{(3)}} = \frac{\partial \mathcal{L}}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial \theta_{k1}^{(2)}}$$
$$= (y_1 - a_1^{(3)})(g(z_1^{(3)})(1 - g(z_1^{(3)}))a_k^{(2)}$$
$$= \delta_1^{(3)} a_k^{(2)}$$

- Partial derivative w.r.t. the parameter of the layers ahead of output layer:

$$\frac{\partial \mathcal{L}}{\partial \theta_{k1}^{(2)}} = \frac{\partial \mathcal{L}}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial \theta_{k1}^{(1)}}$$

$$= (y_1 - a_1^{(3)})(g(z_1^{(3)})\left(1 - g(z_1^{(3)})\right)\theta_{k1}^{(2)}(g(z_1^{(2)})\left(1 - g(z_1^{(2)})\right)a_k^{(1)}$$

$$= \delta_1^{(3)}\theta_{k1}^{(2)}(g(z_1^{(2)})\left(1 - g(z_1^{(2)})\right)a_k^{(1)}$$

$$= \delta_1^{(2)} a_k^{(1)}$$

Recipe of backpropagation algorithm:

Design your neural network
Initialize parameters θ
repeat
  for training instance x_i **do**
    1. **Forward pass** the instance through the network, compute activations, determine output
    2. Compute the **error**
    3. Propagate error **back** through the network, and compute for all weights between nodes *ij* in all layers *l*

$$\Delta\theta_{ij}^{l} = -\eta \frac{\partial E}{\partial \theta_{ij}^{l}} = \eta \delta_i a_j$$

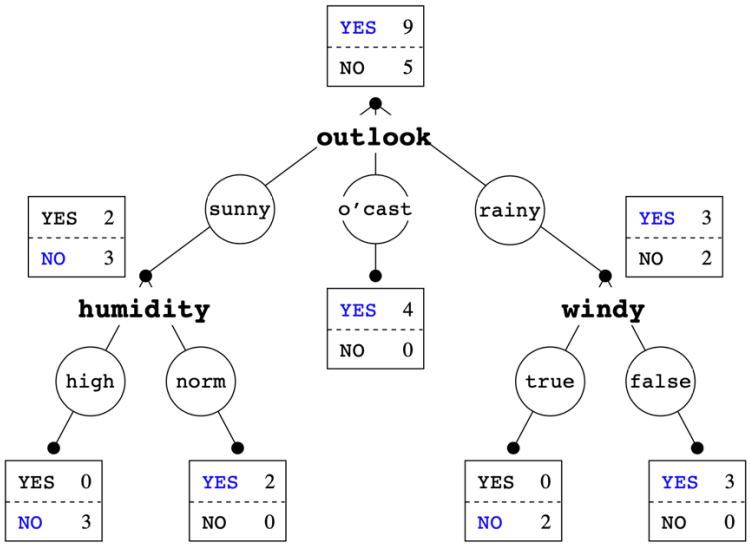
    4. Update **all** parameters **at once**

$$\theta_{ij}^{l} \leftarrow \theta_{ij}^{l} + \Delta\theta_{ij}^{l}$$

until stopping criteria reached.

Decision Tree:

Structure: tree-like structure that classifies an instance based on rules.



ID3 algorithm:

- Overview:

FUNCTION ID3 (Root)
IF all instances at root have same class
THEN stop
ELSE
1. Select a new attribute to use in partitioning root node instances

2. Create a branch for each attribute value and partition up root node instances according to each value

3. Call ID3(LEAF$_i$) for each leaf node LEAF$_i$

- Entropy: the expected value of self-information that measure the level of surprise of a random variable.

$$H(x) = \sum_{i=1}^{N} p(i)\, log_2\, p(i)$$

- Mean information: the weighted entropy of sub-nodes.

$$M(x_1, x_2, \cdots, x_m) = \sum_{i=1}^{m} w(x_i) H(x_i)$$

- Information gain: the difference between entropy of root node and mean information of child nodes.

$$IG(R_A|R) = H(R) - M(x_1, x_2, \cdots, x_m)$$

- Split ratio: to penalize highly branching attributes, we add a normalization term.

$$SI(R_A|R) = -\sum_{i=1}^{m} w(x_i)\, log_2\, w(x_i)$$

- Gain ratio:

$$GR(R_A|R) = \frac{IG(R_A|R)}{SI(R_A|R)}$$

Stopping criteria: ① instances of a single node are of the same class; ② the improvement of information gain or gain ratio is less than a predefined threshold; ③ run out of all possible attributes.

Feature Selection:

Wrapper method:

- Enumerate all possible subsets, train them and find the best attribute subset.
- Starting from an empty set, choose the best attribute iteratively, until the performance does not improve.
- Starting with all attributes, remove one attribute that causes least performance degradation, until the performance does not degrade.

Feature filtering:

- Pointwise mutual information: measure the correlation between one attribute and class label.

$$PMI(a, c) = log_2 \frac{P(a, c)}{P(a) \cdot P(c)}$$

- Mutual information: combine each a, $\bar{a}$, c, $\bar{c}$ PMI.

$$MI(A, C) = p(a, c)PMI(a, c) + p(\bar{a}, c)PMI(\bar{a}, c) + p(a, \bar{c})PMI(a, \bar{c}) + p(\bar{a}, \bar{c})PMI(\bar{a}, \bar{c})$$

- Chi-square:

$$\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}}$$

where $O_{i,j}$ and $E_{i,j}$ is observed value and expected value respectively.

Ensembling Method:

Approaches to ensemble learning: instance manipulation, feature manipulation, class label manipulation, algorithm manipulation.

Stacking: majority voting(each instance's prediction is made by multiple base classifier voting), meta classification(train a classifier over the output of base classifier).

Bagging: construct novel datasets through a combination of random sampling and replacement

- Original dataset:

1	2	3	4	5	6	7	8	9	10

- Bootstrap Samples

7	2	6	7	5	4	8	8	1	10

1	3	8	10	3	5	8	10	1	9

2	9	4	2	7	9	3	10	1	10

⋮

Random tree: for each node, only some of attributes are selected for building decision tree; it is helpful to control unhelpful attributes; it is faster but add more variance.

Random forest: multiple random trees ensemble together to build a strong decision tree; each tree is built using different dataset(bagging); it is helpful to minimize model variance without introducing model bias.

Boosting: iteratively change the distribution and weights of training instances based on whether it is classified correctly; base classifier is combined via weighted voting; it is helpful to reduce bias but introduce variance

Suppose we have n base classifiers: $C_1, C_2, \cdots, C_T$, initial instance weight $w_j^{(0)} = \frac{1}{N}$, the classifier in iteration i is C_i, we compute the error rate such that:

$$\epsilon_i = \sum_{j=1}^{N} w_j^{(0)} \delta(C_i(x_j) \neq y_j)$$

The importance of C_i:

$$\alpha_i = \frac{1}{2} log \frac{1-\epsilon_i}{\epsilon_i}$$

Weight for instance j in iteration $i+1$:

$$w_j^{(i+1)} = \frac{w_j^{(i)}}{Z_i} \cdot \begin{cases} e^{-\alpha_i} & \text{if } C_i(x_j) = y_j \\ e^{\alpha_i} & \text{if } C_i(x_j) \neq y_j \end{cases}$$

where Z_i is normalization factor that let the sum of $w_j^{(i+1)}$ for all j is equal to 1.

Final prediction:

$$C^*(x) = \arg\max_y \sum_{i=1}^{T} \alpha_i \, \delta(C_i(x) = y)$$

Unsupervised Learning:

What is clustering; types of clustering: exclusive vs overlapping, deterministic vs probabilistic, hierarchical vs partitioning, partial vs complete, heterogenous vs homogenous, incremental vs batch.

The measures of unsupervised clustering include cluster cohesion and cluster separation:

$$\text{cohesion}(C_i) = \frac{1}{\sum_{x,y \in C_i} \text{Distance}(x,y)}$$

$$\text{separation}(C_i, C_j) = \sum_{x \in C_i, y \in C_{j \neq i}} \text{Distance}(x,y)$$

The measures of supervised clustering include entropy and purity:

$$\text{entropy} = \sum_{i=1}^{k} \frac{|C_i|}{N} H(x_i)$$

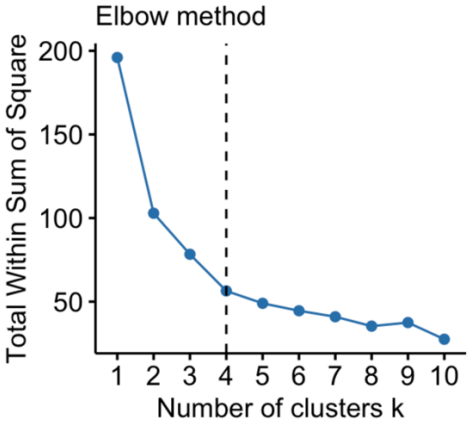
$$\text{purity} = \sum_{i=1}^{k} \frac{|C_i|}{N} \max_j P_i(j)$$

K-means clustering:

1. Select *k* points to act as seed cluster centroids
2. **repeat**
3. Assign each instance to the cluster with the **nearest centroid**
4. Recompute the centroid of each cluster
5. **until** the centroids don't change

complexity: $O(ndki)$, where n is the number of instances, d is the number of attributes, k is the number of cluster centroids, i is the number of iterations.

Elbow method to select k:



Agglomerative clustering:

1. Compute the proximity matrix, if necessary.
2. **repeat**
3. Merge the closest two clusters
4. Update the proximity matrix to reflect the proximity between the new cluster and the original clusters
5. **until** Only one cluster remains

Single link: minimum distance between two points in two clusters

Complete link: maximum distance between two points in two clusters

Group average: average distance between all points in two clusters

Semi-supervised Learning:

Algorithm: Let L be the set of labelled training instances $\{x^{(i)}, y^{(i)}\}_{i=1}^{l}$, U be the set of unlabeled training instances $\{x^{(i)}\}_{i=l+1}^{l+u}$.

Self-training:

Repeat

- Train a model *f* on *L* using any supervised learning method
- Apply *f* to predict the labels on each instance in *U*
- Identify a subset *U′* of *U* with "high confidence" labels
- Remove *U′* from *U* and add it to *L* with the classifier predictions as the "ground-truth" labels ($U \leftarrow U \backslash U'$ and $L \leftarrow L \cup U$)
- Until *L* does not change

If the labelled instances' confidence is below the threshold, it will move back to unlabeled pool.

Active learning: an active classifier that can pose queries for labelling by an oracle. Typically, the most uncertain instance will be sent to oracle for labelling.

Query strategies:

① $x = \arg\max_x (1 - P(y|x;\theta))$: choose instance with the smallest confidence

② $x = \arg\min_x (P(y_1|x;\theta) - P(y_2|x;\theta))$, where y_1 and y_2 are the first and second most probable labels for x.

③ $x = \arg\max_x -\sum_i P(y_i|x;\theta)\, log_2\, p(y_1|x;\theta)$: choose instance with the largest entropy.

④ Query by committee: train multiple classifiers, and query instances with the highest disagreement measured by entropy.

What is data augmentation.

Anomaly Detection:

What is anomaly; what is anomaly detection.

Structure of anomalies: global anomaly, contextual anomaly, collective anomaly.

Supervised anomaly detection (has labels for both normal and anomaly data) and its challenges.

Semi-supervised anomaly detection (labels only for normal data) and its challenges.

Unsupervised anomaly detection (cluster normal objects) and its challenges.

Statistical anomaly detection: learn a model fitting the given dataset and identify those objects that lay in the low probability region. Typically, we have univariant, multivariant Gaussian distribution, where data not in the range of $\mu \pm 3\sigma$ is treated as anomalies.

Mahalanobis distance:

- Mahalanobis Distance

$$y^2 = (\mathbf{x} - \bar{\mathbf{x}})' S^{-1} (\mathbf{x} - \bar{\mathbf{x}})$$

- S is the covariance matrix:

$$S = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \bar{X})(X_i - \bar{X})'$$

Likelihood approach: suppose we have two distributions: one is majority distribution, another is anomalous distribution.

Initially, assume all the data points belong to M
Let $L_t(D)$ be the log likelihood of D at time t
For each point x_t that belongs to M, move it to A

- Let L_{t+1} (D) be the new log likelihood.
- Compute the difference, $\Delta = L_t(D) - L_{t+1}$ (D)
- If $\Delta > c$ (some threshold), then x_t is declared as an anomaly and moved permanently from M to A

where overall data distribution is $D = (1-\lambda)M + \lambda A$. Data likelihood at time t:

$$L_t(D) = \prod_{i=1}^{N} P_D(x_i) = \left((1-\lambda)^{|M_t|} \prod_{x_i \in M_t} P_{M_t}(x_i)\right)\left(\lambda^{|A_t|} \prod_{x_i \in A_t} P_{A_t}(x_i)\right)$$

$$LL_t(D) = |M_t| \log(1-\lambda) + \sum_{x_i \in M_t} \log P_{M_t}(x_i) + |A_t| \log \lambda + \sum_{x_i \in A_t} \log P_{A_t}(x_i)$$

Proximity-based anomaly detection: An object is an anomaly if the nearest neighbor(typically k-th nearest neighbor) is far away.

Three ways to define outliers in terms of proximity:

- Data points for which there are fewer than *p* neighboring points within a distance *D*
- The top n data points whose distance to the kth nearest neighbor is greatest
- The top n data points whose average distance to the k nearest neighbors is greatest

Density-based anomaly detection: anomaly object is in low-density region.

Density is the average distance to k nearest neighbors:

$$\text{density}(\mathbf{x}, k) = \left(\frac{1}{k} \sum_{\mathbf{y} \in N(\mathbf{x},k)} \text{distance}(\mathbf{x}, \mathbf{y}) \right)^{-1}$$

$$\text{relative density}(\mathbf{x}, k) = \frac{\text{density}(\mathbf{x}, k)}{\frac{1}{k} \sum_{\mathbf{y} \in N(\mathbf{x},k)} \text{density}(\mathbf{y}, k)}$$

Cluster-based anomaly detection: anomalous points are points that do not belong to any clusters.

Assess degree to which object belongs to any cluster:

$$\frac{\text{distance}(\mathbf{x}, centroid_C)}{\text{median}(\{\forall_{x' \in C} \text{distance}(\mathbf{x}', centroid_C)\})}$$

Eliminate objects: remove objects which most improve objective function.

Discard small clusters far from other clusters.

Association Rule:

Itemset: a collection of one or more items; support count(σ): frequency of occurrence of an item set; support: fraction of transaction that contains an itemset($\frac{\sigma(A \cup B)}{\sigma(*)}$); confidence: fraction of items in one itemset in transaction in another itemset($\frac{\sigma(A \cup B)}{\sigma(A)}$); frequent itemset: itemset whose support value is greater than $minsup$.

Find association rules:

- Brute-force approach: enumerate all possible itemset prune those whose support and confidence less than a threshold. Complexity: $O(2^n)$.
- Frequent itemset generation: generate itemset with support value greater than $minsup$, then generate association rule whose confidence value is greater than $minconf$.

  Complexity:

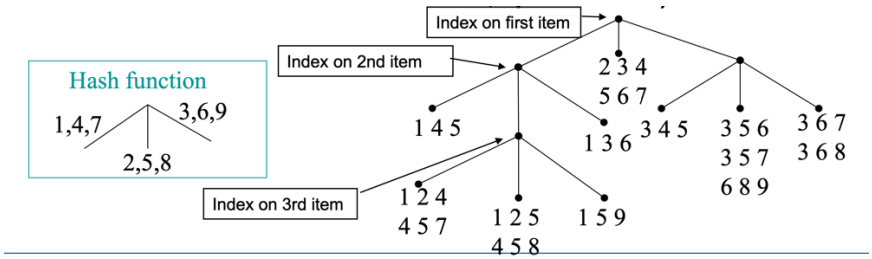
$$\sum_{k=1}^{d-1} \binom{d}{k} \sum_{j=1}^{d-k} \binom{d-k}{j} = 3^d - 2^{d+1} + 1$$

Apriori principle(anti-monotone property):

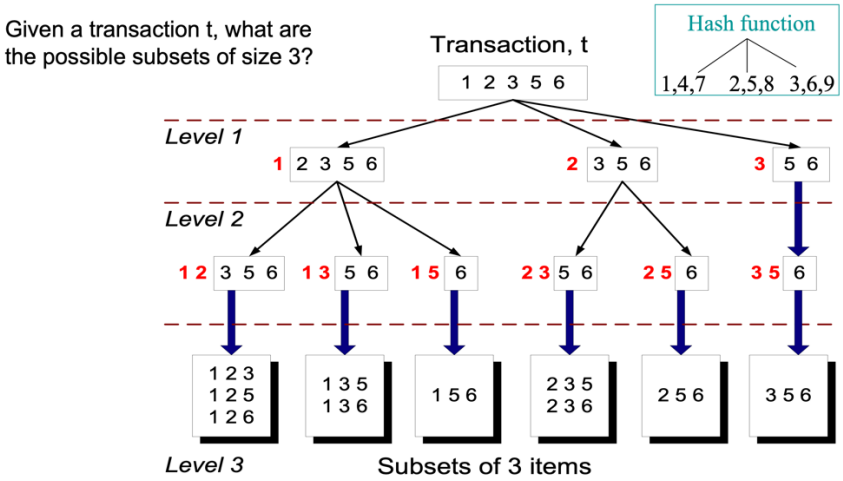
$$\forall X, Y: X \subseteq Y \Rightarrow s(X) \geq s(Y)$$

- Let k=1
- Generate frequent itemsets of length 1
- Repeat until no new frequent itemsets are identified
  - Prune candidate itemsets containing subsets of length k that are infrequent
  - Count the support of each candidate by scanning the database
  - Eliminate candidates that are infrequent, leaving only those that are frequent
  - Generate length (k+1) candidate itemsets from length k frequent itemsets

To reduce the number of comparisons, we store the candidates in a hash structure, where each transaction compares with the candidate in the hash structure.



Subset stored by hash function:



If the length of frequent itemset is k, the number of candidate association rules is $2^k - 2$.

Anti-monotone property of association rule:

e.g., L = {A,B,C,D}:

$$c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$$

Recommender System:

Goals: relevance, novelty, serendipity, diversity.

Content-based recommendation: items are described as features $w_s = (w_{1s}, w_{2s}, \cdots w_{ks})$, users are also described as features $w_c = (w_{1c}, w_{2c}, \cdots w_{kc})$. We use cosine similarity to measure whether we recommend item s to user c:

$$\cos(w_c, w_s) = \frac{\sum_{i=1}^{K} w_{ic} w_{is}}{\sum_{i=1}^{K} w_{ic}^2 \sum_{i=1}^{K} w_{is}^2}$$

Collaborating filtering: predict user preferences by collecting taste information from many other users.

User-based model: similar users have similar rating on similar items.

Suppose we have a rating matrix R: r_{uk} represents rating by user u for item k.

Pearson correlation:

$$\mathrm{Sim}(u,v) = \mathrm{Pearson}(u,v) = \frac{\sum_{k\in I_u\cap I_v}(r_{uk}-\mu_u)\cdot(r_{vk}-\mu_v)}{\sqrt{\sum_{k\in I_u\cap I_v}(r_{uk}-\mu_u)^2}\cdot\sqrt{\sum_{k\in I_u\cap I_v}(r_{vk}-\mu_v)^2}}$$

$$\mu_u = \frac{\sum_{k\in I_u} r_{uk}}{|I_u|}$$ Mean rating of user u

I_u Set of items rated by user u

Prediction of rating of user u for item j:

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v\in P_u(j)}\mathrm{Sim}(u,v)\cdot(r_{vj}-\mu_v)}{\sum_{v\in P_u(j)}|\mathrm{Sim}(u,v)|}$$

$P_u(j)$ Set of nearest users of user u who rated item j

Item-based model: similar items are rated in similar way by the same user.

Adjusted cosine similarity:

$$\mathrm{AdjustedCosine}(i,j) = \frac{\sum_{u\in U_i\cap U_j} s_{ui}\cdot s_{uj}}{\sqrt{\sum_{u\in U_i\cap U_j} s_{ui}^2}\cdot\sqrt{\sum_{u\in U_i\cap U_j} s_{uj}^2}}$$

$s_{uj} = r_{uj} - \mu_u$ Mean-centred rating

Prediction:

$$\hat{r}_{ut} = \frac{\sum_{j\in Q_t(u)}\mathrm{AdjustedCosine}(j,t)\cdot r_{uj}}{\sum_{j\in Q_t(u)}|\mathrm{AdjustedCosine}(j,t)|}$$