AI

Machine Learning Basics

Applications

- Email spam detection
- Face detection and matching in smartphones
- Stock predictions
- Product recommendations
- Sentiment analysis
- Self-driving cars
- Post office
- Medical Diagnoses

Supervised Learning

Uses labelled data and predicts outcome/future.

- Classification - predicts categorical class labels
- Regression - predicts continuous outcomes

Unsupervised Learning

No labels or targets, finds hidden structure/insights in data

- Dimensionality Reduction - reduces data sparsity and computational cost
- Clustering - objectives within a cluster share a degree of similarity

Reinforcement Learning

- Decision process
- Reward system
- Learn series of actions

Clustering

The aim is to segment data into clusters, where there is:

- High intra-cluster similarity
- Low inter-cluster similarity

Terminology

- {x1, x2, ..., xn} - data points
- xi = <xi,1, ..., xi,m> - m-dimensional data point
- d(xi, xj) - distance function

Hierarchical Clustering

Creates a hierarchical decomposition of the set of objects using some criterion, producing a dendrogram.

Agglomerative is bottom-up merging, whilst divisive is top-down merging.

Agglomerative Clustering

1. Place each data point into its own singleton group
2. Iteratively merge the two closest groups
3. Repeat until all the data is merged into one single cluster

Distances between clusters

- **Single Linkage** - similarity of the closest pair
- **Complete Linkage** - similarity of the furthest pair
- **Group Average** - average similarity of all pairs

Strengths

- Provides deterministic results
- No need to specify cluster numbers prior
- Can create clusters of arbitrary shapes

Weaknesses

- Does not scale up for large datasets
- Time complexity at least O(n^2)

Caveats

- Different decisions about group similarities can lead to vastly different dendrograms
- The algorithm forces a hierarchical structure on the data, even data where the structure is not suitable.

K-means

A centroid-based clustering algorithm, where the goal is to assign data to K.

The algorithm objective is to minimise within-cluster variances of all clusters.

- Non-deterministic
- Finds local optimal result

Algorithm

1. Initialise data points and initial cluster points
2. Assign each data point to its closest cluster point
3. Move each cluster point to the middle of the data points it is assigned to
4. Repeat until the cluster points stop moving

DBScan

Acronym: Density-Based spatial clustering of applications with noise

- Clusters are dense regions in the data space separated by regions of lower sample density.
- A cluster is defined as a maximal set of density connected points.
- Discovers clusters of arbitrary shape.

There are three exclusive types of points:

- Core points - dense region, has at least Y number of points within the radius X
- Border points - edge of the cluster, has fewer than Y points within the radius X but in the neighbourhood
- Noise - sparse region

Hyperparameters

- A circle of X radius
- A circle containing at least Y number of points

Density-reachable: point q is directly density-reachable from point p if p is a core point and q is in the neighbourhood.

Algorithm

1. Label all points as core, border or noise
2. Eliminate noise points
3. For every core point p that has not been assigned to a cluster, create a new cluster with point p and all points that are density-reachable from point p
4. For border points in more than one cluster, assign it to the closest core point.

DBScan is resistant to noise and can find non-linearly separably clusters. However, it is not entirely deterministic since border points that are reachable from more than one cluster can be part of either cluster depending on implementation.

Supervised Learning

Given some input x, predict an appropriate outcome, y.

Training Data - annotated data for training, in the form of (Input, Output) pairs. After training is completed, present the AI with a new input it hasn't seen before, allowing it to predict the appropriate result.

Input - attributes, features, independent variable

Output - target, response, dependent variable

Function - hypothesis, predictor

It is not possible to find out the function value at other points.

Linear Regression

Regression - learning a function that captures the "trend" between the input and the output.

An example model for linear data is he class of linear functions (univariate linear regression).

$y = f(x;v,w) = wx + v$

- y is the dependent variable
- v and w are the free parameters
-

x is the independent variable

Loss function - tells how bad a given line is for the given data.

Square Loss:

$$g(w_0, w_1) = \frac{1}{N}\sum_{n=1}^{N} \underbrace{\left(f\left(x^{(n)}; w_0, w_1\right) - y^{(n)}\right)^2}_{\text{loss for the n-th training example}}$$

Every combination of v and w has an associated cost, so to find the best fit, we need to find values for v and w such that the cost is minimum.

Vector Notation Advantages

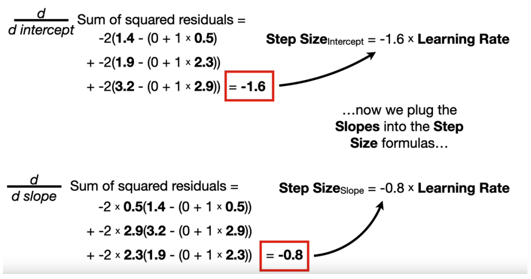
- Vector notation is concise
- With w, v, etc... and x being populated appropriately, these models are still linear in the parameter vector.
- The cost function is still L2.

Gradient Descent

A general strategy to minimise cost functions.

Algorithm

1. Define intercept and slope as 0
2. Define the learning rate (a).
3. For each data point, get the square losses for the intercept and slope each, and get the derivative of each point. Add these up together.
4. Plug the original values of the intercept and slope into the sums to get the step size, and multiply by the learning rate.
5. Calculate the new intercept by subtracting the step size away from the original intercept. Do the same for the slope.
6. Repeat this until the line stops moving or we reach the maximum step count.



$$\frac{d}{d\ \textit{intercept}}\ \text{Sum of squared residuals} = -2(\mathbf{1.4} - (0 + 1 \times \mathbf{0.5})) + -2(\mathbf{1.9} - (0 + 1 \times \mathbf{2.3})) + -2(\mathbf{3.2} - (0 + 1 \times \mathbf{2.9})) \boxed{= \mathbf{-1.6}}$$

$$\textbf{Step Size}_{\text{Intercept}} = -1.6 \times \textbf{Learning Rate}$$

...now we plug the **Slopes** into the **Step Size** formulas...

$$\frac{d}{d\ \textit{slope}}\ \text{Sum of squared residuals} = -2 \times \mathbf{0.5}(\mathbf{1.4} - (0 + 1 \times \mathbf{0.5})) + -2 \times \mathbf{2.9}(\mathbf{3.2} - (0 + 1 \times \mathbf{2.9})) + -2 \times \mathbf{2.3}(\mathbf{1.9} - (0 + 1 \times \mathbf{2.3})) \boxed{= \mathbf{-0.8}}$$

$$\textbf{Step Size}_{\text{Slope}} = -0.8 \times \textbf{Learning Rate}$$

https://www.youtube.com/watch?v=sDv4f4s2SB8

Logistic Regression

A linear model for classification, putting a boundary between two different classes.

- With 1 attribute, a single point can do this.
- With 2 attributes, a line can do this.
- With 3 attributes, a plane can do this.
- With more than 3 attributes, a hyperplan can do this (cannot be drawn or perceived by the human mind)

Linear regression cannot work because there is no ordering between categories. Thus, we change the linear model by passing it through a nonlinearlity.

i.e. if x has one attribute:

$$h(x;\boldsymbol{w}) = \sigma(w_0 + w_1 x) = \frac{1}{1+e^{-(w_0+w_1 x)}}$$

Sigmoid function (logistic function):

$$\sigma(u) = \frac{1}{1+e^{-u}}$$

The sigmoid takes a single argument and returns a value between 0 and 1 - this is the likelihood that the probability is 1. If the result is smaller than 0.5, we predict 0. If it is larger, it is predicted 1.

Decision Boundary - all possible inputs where the sigmoid outputs 0.5. This is always linear.

Logistic Cost Function (cross-entropy):

$$Cost(h(\boldsymbol{x};\boldsymbol{w}),y) = \begin{cases} -\log\left(\overbrace{h(\boldsymbol{x};\boldsymbol{w})}\right), & if\ y=1 \\ -\log\left(1-\underbrace{h(\boldsymbol{x};\boldsymbol{w})}\right), & if\ y=0 \end{cases}$$

Neural Networks

These are incredibly nonlinear models with many free parameters. They can be used for either regression or classification, depending on the choice of loss function.

It can replace nonlinear regression and nonlinear logistic regression, which are less practical.

Model - Sometimes called the architecture - designing this for the problem at hand is the main challenge

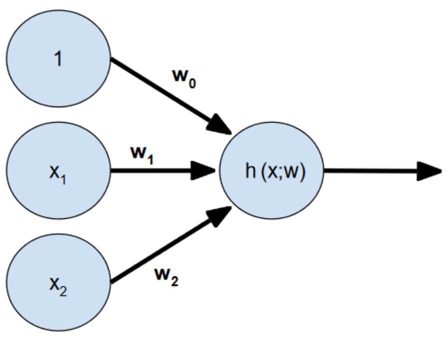
Cost function - Nothing new. For regression, the mean square error between predictions and observed targets. For classification, use logistic loss/cross-entropy.

Learning algorithm - By gradient descent. Update rules are non-trivial due to more complex models. This is done with Backpropagation.

Each iteration of backpropagation takes a gradient descent step. Such implementations exist that can compute the gradient automatically.

- **Node** - one unit, or neuron
- **Weight** - connection via arrow
- **Layers** - node arrangement, with one input layer, one output layer, and a number of hidden layers
- **Activation function** - used by hidden and output nodes, typically a sigmoid function.

Perceptron - a neural network with no hidden layers.



$$h(x, w) = \sigma(w_0 + w_1x_1 + w_2x_2)$$

Multi-layer Perceptron - a neural network with one hidden layer, truly a nonlinear model.

Free parameters:

- Weights (parameters)
- Number of hidden units (hyperparameters)
- Choice of activation function (hyperparameters)

The number of output nodes is the number of targets or labels we want to predict.

Whilst neural networks were popular in the 90s, there was a dip in the 2000s since training fully connected neural networks is a difficult task. They then made a comeback with deep learning.

Deep Learning - Machine Learning, but using a neural network that has multiple hidden layers. The number of hidden layers is another hyperparameter.

Neurons use the sigmoid function on the weighted sum of their input - this is the activation function.

Why Deep Learning

With one hidden layer, you may need a lot of neurons - training becomes time-consuming.

More hidden layers work better in practice, analogous to human vision.

There is no need to hand-craft input attributes that describe the data.

Generative Adversarial Neural Networks (GAN) - can produce new samples such as faces, with a discriminator classifying which face is real or fake. Generator weights are updated based on how well it fools the discriminator.

Training

All applies to supervised learning, including deep learning.

Training data is used to train the model, and test data is the new inputs to feed the obtained neural net, which then must predict appropriate outputs.

Overfitting - trying to fit the data too well to the training data, making the model more complex than needed. All of the noise will not help.

Neural networks can overfit the data.

Regularisation - a way to prevent overfitting.

These include:

- Add a penalty to the cost function to penalise more complex models
- Prune the model - dropout is the process of "leaving out" a proportion of nodes when training a deep network

Stopping training early can also prevent overfitting. After each gradient update, the training cost will decrease until it reaches 0 (overfitting).

A subset of data must be put aside only for monitoring the cost on previously unseen data.

The error on hold-out set will decrease at first but it can start increasing.

Stop training when the error starts increasing.

https://playground.tensorflow.org/

Evaluation

We always split the available annotated data randomly into a training set and a test set; evaluation of a predictor serves to estimate its future performance before deploying it to the real world.

Each hyperparameter value corresponds to a different model, so we need methods that evaluate each candidate model.

For this evaluation, we can no longer use our cost function computed on the training set.

- The more complex the model, the better it fits the training data.
- The goal is to predict well on future data.
- A model that has capacity to fit any training data will overfit.

To choose between models, we need to estimate future performance with a criterion - this is not evaluating the predictor.

The training set is annotated data, used for training within a chosen model

The test set is also annotated data, used for evaluating the performance of the trained predictor before deploying it.

Thus, neither of these can be used to choose the model.

We cannot use the test set because then we no longer have an independent data set to evaluate the final predictor before deployment.

To choose between models or hyperparameters, split out a subset from the training set to become the validation set.

Holdout Validation Method

One of the methods to pick out a validation set

Method

1. Randomly choose 30% of the data to form a validation set
2. Use the remainder as a training set
3. Train the model on the training set
4. Estimate the test performance on the validation set
    - In regression, we compute the cost function on examples of the validation set
    - In classification, we don't compute cross-entropy cost on the validation set, instead of the validation set we compute the 0-1 error metric.
5. Choose the model with the lowest validation error
6. Re-train with the chosen model on the joined train and validation set to obtain the predictor
7. Estimate future performance of the obtained predictor on the test set
8. Ready to deploy the predictor

Advantages

- Computationally cheaper
- Best for large samples

Disadvantages

- Most unreliable if the sample size is not large enough

k-fold Cross-validation

Another method to pick out and test with a validation set

Method

1. Split the training set into k disjoint sets
2. Use k - 1 of those together for training
3. Use the remaining one for validation
4. Permute the k sets and repeat k times
5. Average the performances on the k validation sets

Advantages

- Slightly more reliable than holdout
- Good on large data samples
- Only wastes 10% of data when k = 10

Disadvantages

- Wastes k% of annotated data
- Computationally k times more expensive than holdout

Leave-one-out Validation

Method

1. Leave out a single example for validation, and train on the remaining annotated data
2.

2. For each example, leave it out at least once in each iteration
3. Take the average of the validation errors as measured on the left-out points

Advantage

- Doesn't waste data

Disadvantage

- Computationally more expensive

Probability Theory

Probabilistic model - a mathematical description of an uncertain situation. The two main elements of a probabilistic model include:

- **Sample space** - a set of all possible outcomes
- **Probability law** - assigns to a set A of possible outcomes

Experiment - an underlying process that produces exactly one of several possibilities

Event - a subset of the sample space

Example: Heads or Tails

- **Sample Space**: head (H) or tail (T)
- **Probability law**: P(H) = 0.5, P(T) = 0.5

Axioms

- **Nonnegativity** - P(A) >= 0 for every event A
- **Additivity** - if A and B are disjoint events, the probability of their union satisfies P(A U B) = P(A) + P(B)
- **Normalisation** - the probability of the entire sample space is equal to 1

Discrete Random Variables

Given an experiment and the corresponding sample space, a random variable maps a particular number with each outcome.

Mathematically, a random variable X is a real-valued function of the experimental outcome.

Probability mass function - captures the probabilities of the values that a (discrete) random variable can take.

Notation

- Random variables are usually indicated with uppercase letters, e.g. X, Temperature or Infection
- The values indicated with lowercase letters, e.g. X ∈ {true, false}
- Vectors are usually indicated with bold letters or a small arrow above the letter, e.g. **X**
- PMF is usually indicated by the symbol $p_x(x)$

Unconditional/Conditional Probability Distributions

Unconditional Probability Distributions - gives us the probability of all possible events without knowing anything else about the problem (e.g. the maximum value of two rolls of a 4-sided die)

- P(X) = {1/15, 3/15, 5/15, 7/15}

Conditional Probability Distributions - gives us the probability of all possible events with some additional knowledge (e.g. the maximum value of two rolls of a 4-sided die knowing the first roll is 3)

- P(X | X1 = 3) = {0, 0, 0.75, 0.25}

Joint Probability Distribution - the probability distribution associated to all combinations of the values of two or more random variables, indicated by commas.

- P(X, Y) or P(Toothache, Cavity)
- We can calculate the join probability distribution by using the product rules:
- P(X, Y) = P(X|Y)P(Y) = P(Y|X)P(X)

Mean (μ) - represents the centre of gravity of the PMF. (e.g. E(X) = 1 * 0.25 + 2 * 0.25 + 3 * 0.25 + 4 * 0.25 = 2.5)

Variance - provides a measure of dispersion around the mean:

$$var(X) = \sum_{x} \left(x - E(X)\right)^2 p_X(x)$$

Standard Deviation - the square root of variance, providing another measure of dispersion.

Probability Density Function - a nonnegative function that describes a probability law. This is used for continuous variables, since there are an infinite amount of values that X can take.

Example: Random Number Generator

Let us consider a random number generator that returns a random value between 0 and 1.

Let us model it with a Gaussian distribution:

$$P(X = a \mid \mu, \sigma^2) = \frac{1}{\sigma\sqrt{(2\pi)}}\, e^{\frac{-(a-\mu)^2}{2\sigma^2}},$$

Where mu is the mean, pi is pi, e is e.

Bayes' Theorem

Recall the product rule for a joint probability distribution of independent variable(s) X and dependent variable Y:

- P(X, Y) = P(X|Y)P(Y) = P(Y|X)P(X)

By taking the second and last term from the above equation and rearranging it, we get:

$$\boldsymbol{P}(X \mid Y) = \frac{\boldsymbol{P}(Y \mid X)\boldsymbol{P}(X)}{\boldsymbol{P}(Y)}$$

This is Bayes' Theorem.

Probabilistic Inference - a method wherein posterior probabilities for query propositions are computed, given some observed evidence.

We use Bayes' Theorem about an underlying process given a knowledge base consisting of the data produced by this process.

Applying Bayes' Theorem

Let us consider output class c and input value(s) a. Bayes' Theorem can be rewritten as:

$$P(c \mid a) = \frac{P(a \mid c)P(c)}{P(a)}$$

Naive Bayes

The issues with Bayes' Theorem is that for increasing numbers of independent variables, all possible combinations must be considered:

$$P(c|a_1, \dots, a_n) = \alpha\, P(c)\, P(a_1, \dots, a_n|c)$$

For a domain described by n Boolean variables, we would need an input table of size O(2^n) and it would take O(2^n) to process the table.

Conditional Independence

We assume each input variable is conditionally independent of any other input variables given the output.

Independence: A is independent of B when the following equality holds. P(A|B) = P(A)

Conditional Independence: x1 is conditionally independent of x2 given y when the following equality holds: P(x1|x2,y) = P(x1, y)

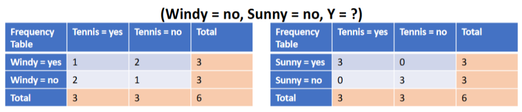
$$P(c|a_1, \dots, a_n) = \alpha\, P(c)\, P(a_1, \dots, a_n|c)$$

$$P(c|a_1, \dots, a_n) = \alpha\, P(c)\, P(a_1|c) P(a_2|c) \dots\, P(a_n|c)$$

$$P(c|a_1, \dots, a_n) = \alpha\, P(c) \prod_{i=1}^{n} P(a_i|c)$$

where $\alpha = 1/\beta$ and $\beta = \sum_{c \in y} (P(c)\, \prod_{i=1}^{n} P(a_i|c))$

Example



(Windy = no, Sunny = no, Y = ?)

Frequency Table	Tennis = yes	Tennis = no	Total
Windy = yes	1	2	3
Windy = no	2	1	3
Total	3	3	6

Frequency Table	Tennis = yes	Tennis = no	Total
Sunny = yes	3	0	3
Sunny = no	0	3	3
Total	3	3	6

P(T|¬S,¬W) = aP(T)P(¬W|T)P(¬S|T) = a * 0.5 * 2/3 * 0 = 0

P(¬T|¬S,¬W) = aP(¬T)P(¬W|¬T)P(¬S|¬T) = a * 0.5 * 1/3 * 3/3 = a* 1/6

a = 1/b = 1/(3/6 * 2/3 * 0 + 3/6 * 1/3 * 3/3) = 6

Thus, P(¬T|¬S,¬W) = 1

There is a problem in this example: there is no data where Tennis = yes and Sunny = no, regardless of the value of Windy, and we get inaccuracies from this.

Laplace Smoothing

To avoid this, we can just add 1 to the frequency of all elements of our training data. Then we use the updated tables when calculating P(ai|c), so we do not get values with 0. When we calculate P(c), we use the original tables.

Numerical Independence

For categorical independent variables, we can compute the probability of an event through the probability mass function associated with the training data.

k-Nearest Neighbours

Notation

Probabilistic models

- Variables are denoted by uppercase letters, X or Y
- Values that a variable can take are denoted by lowercase letters
- Vectors are denoted by letters in bold

Now, prepare for a little switcheroo:

- Variables are denoted by lowercase letters, x or y
- Values are typically stated and letters are generally not used to represent values
- Vectors are still denoted by letters in bold.

Nonparametric Model - a model that cannot be characterised by a bounded set of parameters. For example, suppose each prediction made will consider all training examples, including the one from previous predictions. The set of examples grow over time.

This approach is also called instance or memory-based learning.

Asymptotic Analysis

Computer scientists are often asked to determine the quality of an algorithm by comparing it with other ones and measure the speed and memory required.

Benchmarking

- We run the algorithms and measure speed (in seconds) and memory consumption (in bytes).
- The problem with this is that this approach measures the performance of a specific program written in a particular language, on a given computer, with particular input data

Asymptotic analysis

- A mathematical abstraction over both the exact number of operations and exact content of the input
- It is independent of the particular implementation and input.

Method

1. Abstract over the input. In practice, we characterise the size of the input, this being n
2. Abstract over the implementation. The idea is to find some measure that reflects the running time of the algorithm.

Notations

- Big O notation
- Big Omega notation
- Big Theta notation

Given a function f and g(n):

- For big O notation (f is O(n)), |f| is bounded above by a function g asymptotically (e.g. O(n^2) and f(x) = x^2 + 2x)
- For big omega notation, f is bounded below by g asymptotically
- For big theta notation, f is bounded above g asymptotically

Search Problem Formulation

Agent - something that perceives and acts in an environment.

Problem-Solving Agent - an agent that uses atomic representations (each state of the world is perceived as indivisible) and requires a precise definition of the problem and its goal/solution

Problem Formulation - the process of deciding what actions and states to consider given a goal.

We make the following assumptions about the environment:

- Observable, i.e. the agent only knows the current state
- Discrete, i.e. there are only finitely many actions at any state
- Known, i.e. the agent knows which states are reached by each action
- Deterministic, i.e. each action has exactly one outcome

The agent's task is to find out how to act, now and in the future in order to reach a goal state, namely to determine a sequence of actions.

The process of looking for these sequence of actions is called a search.

A solution to a search problem is the sequence of actions from the initial state to the goal state.

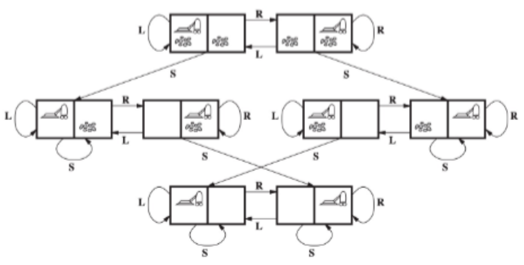
Problem Components

- **Initial state** - the state that the agent starts in
- **Actions** - a description of all possible actions that can be executed in a given state
- **Transition model** - the states resulting from executing each action a from every state s.
- **Goal test** - to determine if a state is a goal state.
- **Path cost** - function that assigns a value (cost) to each path

The initial state, actions and transition model together define the state space of the problem, in the form of a directed graph or network.

Path - a sequence of states connected by a sequence of actions

Example



- Initial state: any state
- Actions: L (left), R (right), S (suction)
- Transition model: see image
- Goal test: if all squares are clean
- Path cost: 1 per step

Assume the initial state is the top-left state, namely the agent is in the left square, both squares being dirty. An example solution:

S + R + S (1 + 1 + 1 = 3)

Typical AI problems have a large number of states and it is virtually impossible to draw the state space graph.

Breadth-First Search

A solution is an action sequence from an initial state to a goal state.

Possible action sequences form a search tree with the initial state at the root; actions are the branches and nodes correspond to the state space.

The idea is to expand the current state by applying each possible action, generating a new set of states.

Uninformed search (blind search): the strategies have no additional information about states beyond that provided in the problem definition. These can only generate successors and distinguish a goal state from a non-goal state.

The key difference between two uninformed search strategies is the order in which nodes are expanded.

Breadth-First Search

This is one of the most common search strategies.

The root node is expanded first, then all successors of the root node are expanded, and the successors of each of these nodes are expanded.

In general, the frontier nodes that are expanded belong to a given depth of the tree. This is equivalent to expanding the shallowest unexpanded node in the frontier; simply use a queue for expansion.

Algorithm

1. Expand shallowest node in the frontier
2. Do not add children in the frontier if the node is already in there or in the list of visited nodes
3. Stop when a goal node is added to the frontier

Measuring Performance

We can evaluate the performance of an algorithm based on the following:

- **Completeness** - whether the algorithm is guaranteed to find a solution if there is one
- **Optimality** - whether the strategy is able to find the optimal solution
- **Time complexity** - the time the algorithm takes to find a solution
- **Space complexity** - the memory used to perform the search

To measure the performance, the size of the space graph is usually used, i.e. v + e, the set of vertices and set of edges.

We use an implicit representation of the graph via the initial state, actions and transition model.

Thus, the following three quantities are used:

- **Branching factor** - the maximum number of successors of each node: b
- **Depth** - the shallowest goal node: d
- The maximum length of any path in the state space: m

Overall, the performance of BFS:

- **Completeness** - if the goal node is at some finite depth d, then the BFS algorithm is complete as it will find it given that b is finite.
- **Optimality** - BFS is optimal if the path cost is a nondecreasing function of the depth of the node
- **Time Complexity** - O(b^d), assuming a uniform tree where each node has b successors, we generate b + b^2 + b^3 + ... + b^d = O(b^d)
- **Space Complexity** - O(b^d), if we store all expanded nodes, we have O(b^(d-1)) explored nodes in memory and O(b^d) in the frontier

Depth-First Search

Depth-first search is another search strategy.

Method

1. The root node is expanded first
2. Then the first successor of the root node is expanded
3. The deepest node in the current frontier is expanded

This is equivalent to expanding the deepest unexpanded node in the frontier; simply use a stack for expansion.

Performance

- **Completeness** - DFS is not complete if the search space is infinite or if we do not check infinite loops. It is complete if the search space is infinite.
- **Optimality** - DFS is not optimal as it can expand a left subtree when the goal node is in the first level of the right subtree.
- **Time Complexity** - O(b^m), depending on the maximum length of the path in the search space (in general m is larger than d)
- **Space Complexity** - O(b^m), as we store all the nodes from each path from the root node to the leaf node.

It has several issues: it is not optimal, it has high time complexity, and high space complexity.

Variations

DFS can have less memory usage and limit depth.

Less memory usage

- Imagine we have a tree and have fully expanded the left subtree - the last node is not the goal node and has no children
- The next step is to expand the right subtree
- Since we've explored the left subtree, we can remove it from memory, reducing space complexity to O(bm)

Depth-limited search

- Infinite state spaces can be mitigated by providing a depth limit, l
- The time complexity is O(b^l) and space complexity is O(bl)

Informed Search

Informed search strategies use problem-specific knowledge beyond the definition of the problem itself

These can find solutions more efficiently compared to uninformed search.

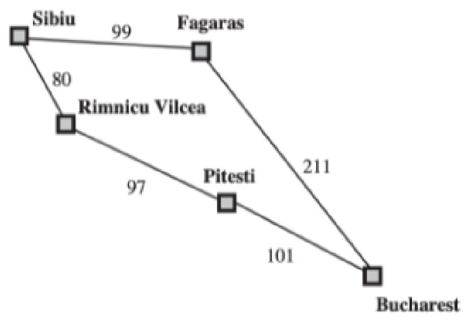
The general approach, best-first search, is to determine which node to expand based on an evaluation function. This function acts as a cost estimate; the node with the lowest cost is the one that is expanded next.

The evaluation function f(n) for most best-fit algorithms includes a heuristic function as a component: h(n) = estimated cost of the cheapest path from node n to a goal node.

Heuristic functions are the most common form in which new knowledge is given to the search algorithm. If n is a goal node, then h(n) = 0. It can be a rule of thumb, common knowledge; it is quick to compute but not guaranteed t work.

Example

Take for example, you need to find the shortest path to Bucharest in Romania:



We can use the straight-line distance heuristic for this. This is correlated with actual road distances.

A* Search

The most widely known informed search strategy is A*.

This search strategy evaluates nodes using the following cosy function:

f(n) = g(n) + h(n)

Where g(n) is the cost to reach the node, and h(n) is the heuristic from the node to the goal.

This is equivalent to the cost of the cheapest solution through node n.

Algorithm

1. Expand the node in the frontier with the smallest cost f(n) = g(n) + h(n)
2. Do not add children in the frontier if the node is already in the frontier or in the list of visited nodes to avoid loopy paths.
3. If the state of a given child is in the frontier, place the child into the frontier and remove the node with larger g(n) from the frontier
4. Stop when a goal node is visited

Performance

A* is complete and optimal if h(n) is consistent.

A heuristic is said to be consistent or monotone if the estimate is always no greater than the estimated distance from any neighbouring vertex to the goal, plus the cost of reaching that neighbour.

h(n) =< cost(n, n') + h(n')

The number of states for the A* search is exponential in the length f the solution, namely for constant step costs: O(b^(e d))

When h* is the actual cost from the root node to goal node, e = (h* - h)/h* is the relative error.

Space is the main issue with A* as it keeps all generated nodes in memory, thus A* is not suitable for many large-scale problems.

Optimisation

Optimisation problems - to find a solution that minimises/maximises one or more pre-defined objective functions. This includes maximisation/minimisation problems. There may be some constraints that must or should be satisfied for a given solution to be feasible.

Solutions do not correspond to paths built step by step from an initial to a goal state.

Instead, the algorithms typically maintain whole candidate solutions from the beginning.

Candidate solutions may be feasible or infeasible.

Example

Routing problem: given a motorway map containing N cities, the map shows the distance between connected cities. We have a city of origin and a city of destination. Problem? Find a path from the origin to the destination that minimises the distance travelled, whilst ensuring that direct paths between non-neighbouring cities are not used.

Search and Optimisation

In search, we are interested in searching for a goal state.
In optimisation, we are interested in searching for an optimal state.

As many search problems have a cost associated to actions, they can also be formulated as optimisation problems.

Similarly, optimisation problems can frequently be formulated as search problems associated to a cost function.

Many search algorithms will "search" for optimal solutions such as A*.

Optimisation algorithms may also be used to solve search problems if they can be associated to an appropriate function to be optimised.

Advantages

- Usually more space efficient, frequently requiring the same amount of space from the beginning to the end of the optimisation process. They do not maintain alternative paths and are frequently able to find reasonable solutions for problems with large state spaces, for which tree-based search algorithms are unsuitable.
- Can potentially be more time efficient.
- Do not necessarily require problem-specific heuristics

Disadvantages

- Not guaranteed to retrieve the optimal solution in a reasonable amount of time.
- Depending on the problem formulation and operators, not guaranteed to be complete either.

It can be used for any problem that can be formulated as an optimisation problem.

Example

Bin packing problem: Given bins with maximum volume V, which cannot be exceeded. We have n items to pack with volume v. We must pack all items. Problem? Find an assignment of items to bins that minimises the number of bins used, ensuring all items are packed and the volume of bins is not exceeded.

Learning vs Optimisation

From an algorithmic perspective, learning can be seen as finding parameters that minimise a loss function. We can compute the loss based on the training set.

From a problem perspective, the goal of machine learning is to create models able to generalise to unseen data.

In supervised learning, we want to minimise the expected loss, i.e. the loss considering all possible examples, including those that we have not observed yet.

We cannot calculate the loss based on unseen data using training time.

So, learning can essentially be seen as trying to optimise a function that cannot be computed.

Thus, our algorithms may calculate the loss based on the training set, and design a loss function that includes a regularisation term, in an attempt to generalise well to unseen data.

From a problem perspective, optimisation usually really wants to minimise (or maximise) the value of a given (known) objective.

In that sense, learning and optimisation are different.

However, there will be some optimisation problems where we can't compute the exact function to be optimised, causing the distinction between learning and optimisation to become more blurry.

Optimisation Problem Formulation

Minimise f(x)

Minimise - minimise or maximise?

f - the objective function

x - design variables

Constraints

Subject to gi(x) <= 0, i = 1, ..., m

hj(x) = 0, j = 1, ..., n

Design variables - represent a candidate solution. These define the search space of the candidate solutions.

Objective function - defines the quality (or cost) of a solution, the function to be optimised.

Solutions must satisfy certain constraints, which define solution feasibility. Candidate solutions may be feasible or infeasible.

Example: Routing problem

Design variables represent a candidate solution.

- Sequence x containing the cities to be visited where xi ∈ C
- C is the set of available cities, and x can be of any size
- The search space consists of all possible sequences of cities.
- Oradea - 1, Sibiu - 10, Fagaras - 2, Bucharest - 14

Objective function defines the quality (or cost) of a solution.

- Minimise the sum of the distances between consecutive cities in x.
-

$$\text{Minimise } f(\mathbf{x}) = \sum_{i=1}^{\text{size}(\mathbf{x})-1} D_{x_i, x_{i+1}}$$

- where D is a matrix of distances, with each position Di,j containing the distance in km to travel directly between city i and j, or -1 if such direct path does not exist.

Solutions must satisfy certain constraints, which define solution feasibility.

- (Inexistent) direct paths between non-neighbouring cities must not be used - explicit constraint.
- Assume we have a matrix D where position Di,j contains the direct distance between city i and j, or -1 f the path does not exist:
- $h_1 : \mathbf{x} \rightarrow \{0,1\} \qquad h_1(\mathbf{x}) = \begin{cases} 0 & \text{if } D_{x_i, x_{i+1}} \neq -1, \quad \forall i \in \{1,\ldots,\text{size}(\mathbf{x}) - 1\} \\ 1 & \text{otherwise} \end{cases}$
- We must start at the city of origin and end at the city of destination - explicit constraint.
- $h_2 : \mathbf{x} \rightarrow \{0,1\}$

  $h_2(\mathbf{x}) = \begin{cases} 0 & \text{if } x_1 = \text{OriginCity and } x_{\text{size}(\mathbf{x})} = \text{DestinationCity} \\ 1 & \text{otherwise} \end{cases}$
- Only cities in C can be used - implicit constraint.

Formal Routing Problem Format

$$\text{Minimise } f(\mathbf{x}) = \sum_{i=1}^{\text{size}(\mathbf{x})-1} D_{x_i, x_{i+1}}$$

$$\text{Subject to } h_1(\mathbf{x}) = 0 \text{ and } h_2(\mathbf{x}) = 0$$

Where $x_i \in \{1, \cdots, N\}$; $\{1, \cdots, N\}$ are the cities in the map; $\mathbf{x}$ has any size;

D is a matrix of distances, with each position $D_{i,j}$ containing:

- the distance in km to travel directly between city i and j, or
- -1 if such direct path does not exist.

$$h_1(\mathbf{x}) = \begin{cases} 0 & \text{if } D_{x_i, x_{i+1}} \neq -1, \quad \forall i \in \{1,\ldots,\text{size}(\mathbf{x}) - 1\} \\ 1 & \text{otherwise} \end{cases}$$

$$h_2(\mathbf{x}) = \begin{cases} 0 & \text{if } x_1 = \text{OriginCity and } x_{\text{size}(\mathbf{x})} = \text{DestinationCity} \\ 1 & \text{otherwise} \end{cases}$$

Hill Climbing

Algorithm

1. Generate the current solution randomly
2. Generate neighbour solutions (differing from the current solution by a single element)
3. Note the best neighbour
4. If the quality of the best neighbour is worse than the current solution, return the current solution
5. Otherwise, note the best neighbour and consider it as the current solution
6. Repeat from step 2 until something proper is returned

Advantage

- Hill climbing allows you to quickly reach the top.

Disadvantages

- Hill-climbing may get you trapped in a local optimum
- It is a local search method
- It's... greedy?
- It may get trapped in a plateaux.

Performance

- Not guaranteed to find optimal solutions.
- We will run until the maximum number of iterations m is reached.
- Within each iteration, we will generate a maximum number of neighbours n, each of which may take O(p) each to generate.
- Worst case scenario: O(mnp)
- Assume the design variable is represented by O(q)
- Within each iteration, we will generate a maximum number of neighbours n.
- Space complexity: O(nq)

Simulated Annealing

In simulated annealing, instead of picking the best neighbour, we pick a random neighbour. There will be a chance it accepts a bad neighbour.

```
1. current_solution = generate initial solution randomly

2. Repeat:
    2.1 rand_neighbour = generate random neighbour of current_solution
    2.2 If quality(rand_neighbour) <= quality(current_solution) {
            2.2.1 With some probability,
                  current_solution = rand_neighbour
        } Else current_solution = rand_neighbour
    2.3 Reduce probability
  Until a maximum number of iterations
```

The probability to accept solutions with much worse quality would be lower. We don't want to be dislodged from the optimum.

There is high probability in the beginning, a more similar effect to random search; it allows us to explore the search space.

The probability gets lower as time goes by, giving a similar effect to hill-climbing and allows us to exploit a hill.

We would like to decrease the probability slowly - if we do this, we start to form basis of attraction, but you can still walk over small hills initially. As the probability decreases further, the small hills start to form basis of attraction too.

But if you do so slowly enough, you give time to wander to the higher value hills before starting to exploit, so you can find the global optimum.

If you decrease too quickly, you can get trapped in a local optima.

Probability of accepting a solution of equal or worse quality, inspired by thermodynamics:

$$e^{\Delta E/T}$$

$\underset{(<=0)}{\Delta E}$ = quality(rand_neighbour) - quality(current_solution)

Assuming maximisation…

$\underset{(>0)}{T}$ = temperature

As the quality of the random neighbour decreases, delta E decreases, which results in the probability in its entirety decreasing. If it increases though, the chance increases.

There is high probability in the beginning, and lower probability as time goes by.

T affects the probability of accepting a solution of equal or worse quality. If T is higher, the probability of accepting the neighbour is higher. Reducing the temperature over time would reduce the probability of accepting the neighbour.

At first, the probability should be high, so T should start high before lowering. There can be a lowering rule where T = aT, where a = 0.95.

Performance

- Not guaranteed to find the optimum in a reasonable amount of time.
- Whether it finds the optimum depends on the termination criteria and schedule.
- If we leave simulated annealing to run indefinitely, it will find an optimal solution depending on the schedule used.
- Time required for that can be prohibitive, even more than the time to enumerate all possible solutions using brute force.
- Thus, it can obtain good solutions by escaping several poor local optima in a reasonable amount of time.
- We will run more or less iterations depending on the schedule and minimum temperature/termination criterion.
- It is possible to compute the time complexity to reach the optimal solution, but it varies depending on the problem and may be even worse than brute force.
- Space complexity depends on how the design variable is represented in the algorithm.

Dealing with contraints

Most real world problems have constraints.

Optimisation algorithms themselves usually do not contain strategies to deal with constraints, so they need to be desogned for each problem.

Examples include:

- Representation, initialisation and neighbourhood operators
- Objective function

Representation

- 1D array of size N, where N is the number of cities to visit
- The fact that the return to the initial city is not in the representation helps to deal with the implicit constraint that we must return to the city of origin

Initialisation

- Draw cities uniformly at random from {1, ..., N} without replacement
- This ensures there are no missing or duplicated cities and that only cities in {1, ..., N}

Neighbourhood Operator

- Reverse the path between two randomly picked cities
- Ensures there are no missing or duplicated cities and that only cities in {1, ..., N} are used

Advantage

- Ensures that no infeasible candidate solutions will be generated, facilitating the search for optimal solutions

Disadvantages

- May be difficult to design and the design is problem-dependent
- Sometimes, it could restrict the search space too much, making it difficult to find the optimal solution