CSE 486 A

Emil Sayahi

15th November 2023

Lecture notes from the 2023 undergraduate course 'Introduction to Artificial Intelligence', given by Professor Khodakhast Bibak at Miami University at Benton Hall in the academic year 2023-2024. This course covers introductory artificial intelligence concepts. Credit for the material in these notes is due to Professor Khodakhast Bibak, while the structure is loosely taken from the in-class lectures. The credit for the typesetting is my own.

Disclaimer: This document will inevitably contain some mistakes—both simple typos and legitimate errors. Keep in mind that these are the notes of an undergraduate student in the process of learning the material, so take what you read with a grain of salt. If you find mistakes and feel like telling me, I will be grateful and happy to hear from you, even for the most trivial of errors. You can reach me by email, in English, at sayahie@miamioh.edu.

For more notes like this, visit my GitHub profile.

Emil Sayahi,
Fall Term: 2023,
Last Update: 15th November 2023,
Miami University

Contents

Wed, 30 August 2023, 11:40am – 1:00pm

Lecture 1: Week 1, Wednesday

1.1 Uninformed Search

Many AI tasks can be formulated as search problems; the goal is to find a *sequence of actions*.

- Puzzles
- Games
- Navigation
- Assignment
- Motion planning
- Scheduling
- Routing

Fri, 1 September 2023, 11:40am – 1:00pm

Lecture 2: Week 1, Friday

Definition 2.1

Uninformed search is a search strategy that uses no problem-specific knowledge. Only the goal test and the successor function are used; the **successor function** generates all possible states. It is not known which non-goal states are better than others. Strategies that know whether one non-goal state is better than another are referred to as **informed search** or **heuristic search** strategies.

There are five major types of uninformed search strategies:

- Breadth-first search (BFS)
- Uniform-cost search (UCS)
- Depth-first search (DFS)
- Depth-limited search (DLS)
- Iterative deepening search (IDS)

All of these uninformed search strategies are distinguished by the *order* in which nodes are expanded.

2.1 Breadth-first search (BFS)

Breadth-first search operates level-by-level, expanding all nodes at a given level before expanding any nodes at the next level. On a given level, nodes are expanded from left to right by convention.

Definition 2.2

Breadth-first search is implemented using a **first in, first out (FIFO) queue**. The FIFO queue is a data structure that supports two operations: **enqueue** and **dequeue**. The enqueue operation adds an element to the end of the queue, and the dequeue operation removes an element from the front of the queue.

Definition 2.3

If a solution exists, breadth-first search will find it in finite time, provided that the branching factor is finite and the depth of the solution is finite; this means that breadth-first search is **complete**. Breadth-first search is not always **optimal**, however, as the solution found may not have the minimum cost. It is optimal when all edges have the same cost, no cost, or when the cost is a non-decreasing function of the depth of the node.

Definition 2.4

The **time complexity** of an algorithm is the number of steps required to solve a problem of size n, where n is the size of the input; in the worst-case of breadth-first search, the goal node would be the very last node explored (ie, every vertex and edge is explored; $O(|V| + |E|)$). The **space complexity** of an algorithm is the maximum amount of memory required to solve a problem of size n; in the worst-case of breadth-first search, the goal node is discovered after all verticies are explored & stored in memory $O(|V|)$.

Definition 2.5

Complexity is expressed in terms of three quantities:

- b is the **branching factor**, or the maximum number of children, or 'successors', of any node.
- d is the **depth** of the shallowest (ie, closest to the root) goal node.
- m is the **maximum length** of any path in the state space.

The time and space complexity of breadth-first search is exponential, $O(b^d)$, where b is the branching factor and d is the depth of the shallowest goal node. This is because the number of nodes expands exponentially with the depth of the tree.

2.2 Uniform-cost search (UCS)

Uniform-cost search expands the node n with the *lowest* path cost $g(n)$ instead of expanding the shallowest node, where $g(n)$ returns the cost of the path from the starting node, s, to the current node, n. This is also referred to as 'Dijkstra's algorithm'. This algorithm uses a priority queue to order nodes in the frontier list by path cost, with the lowest cost node at the front of the queue.

Wed, 6 September 2023, 11:40am – 1:00pm

Lecture 3: Week 2, Wednesday

3.1 Uniform-cost search (UCS) – continued

UCS is optimal and complete, but its time and space complexity remain exponential in the worst case ($O(b^{1+\lfloor \frac{C^*}{\varepsilon} \rfloor})$), where all edge costs are at least ε, $\varepsilon > 0$, and C^* is the cost of the optimal solution).

3.2 Depth-first search (DFS)

Depth-first search expands the *deepest* node first, removing elements from memory as it proceeds.

Definition 3.6

Depth-first search performs **chronological backtracking**; when a search hits a dead end, it backs up to the level above.

Definition 3.7

DFS is not complete without a **depth bound**, D.

DFS is not optimal or complete. It has an exponential time complexity of $O(b^m)$ and a linear space complexity of $O(bm)$, where M is the maximum length of any path in the state space, and b is the branching factor.

3.3 Iterative deepening search (IDS)

IDS is a combination of BFS and DFS. It performs a DFS with a depth bound, D (typically starting at 1), that increases with each iteration. It is complete (when there are no loops) and optimal, and has a time complexity of $O(b^d)$ and a space complexity of $O(bd)$, where d is the depth of the shallowest goal node.

Definition 3.8

Iterative deepening search is an example of an **'anytime' algoirhtm**; it can return a valid solution to a problem even if it is interrurpted before concluding. It is expected to find better solutions as it continues running.

Generally, IDS is the preferred uninformed search algorithm when the search space is large and the depth of the solution is not known.

Fri, 8 September 2023, 11:40am – 1:00pm

Lecture 4: Week 2, Friday

4.1 Informed Search

Definition 4.9

Informed search algorithms use problem-specific knowledge (ie, **domain knowledge**) to find solutions more efficiently than uninformed search algorithms. They use a **heuristic function** to estimate the cost of the cheapest path from a given node to a goal node. The heuristic function is denoted $h(n)$, where n is a node in the search tree. $h(n) \geq 0$ for all n, while an $h(n)$ close to 0 means that n is close to a goal node, while an $h(n)$ that is very large means that n is far from a goal node.

4.2 A^* Search

A^* search is an informed search algorithm that uses a heuristic function to estimate the cost of the cheapest path from a given node to a goal node. It uses a **cost function**, $f(n)$, to estimate the cost of the cheapest path from the start node to a goal node through n. $f(n)$ is defined as $f(n) = g(n)+h(n)$. $g(n)$ is the cost of the path from the start node to n, and $h(n)$ is the heuristic function. A^* search expands the nodes on the frontier in order of increasing $f(n)$ values (ie, the node with the lowest $f(n)$ is expanded first).

Definition 4.10

A heuristic, h, is **admissible** if it never overestimates the cost of reaching the goal, ie, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost of reaching the goal from n. An admissible heuristic is **optimistic**. The straight-line distance (h_{SLD}) between two points is an admissible heuristic for the problem of finding the shortest path between them. If the heuristic is admissible, then A^* search is optimal.

The time & space complexity of A^* search is polynomial when h satisfies $|h(n) - h^*(n)| = O(\log h^*(n))$. When $h(n) = 0$, this condition is *not* satisfied, and, in effect, A^* search becomes UCS.

Wed, 13 September 2023, 11:40am – 1:00pm

Lecture 5: Week 3, Wednesday

Definition 5.11

Multiagent environments are environments in which multiple agents share the same environment.

Contingency plans are necessary to account for the unpredictability of other agents.

Definition 5.12

Each agent has its own **utility function** that maps states to real numbers.

Definition 5.13

A **zero-sum game** is a game in which the sum of the utilities of all agents is zero. A **game** is a decision-making problem, which is a multiagent environment in which the agents' goals are in conflict. A **competitive game** is a game in which the agents' utility functions are maximised by different states.

Definition 5.14

A **game** is defined by:

- S_0, the initial state.
- PLAYER(s), which returns the player whose turn it is in state s.
- ACTIONS(s), which returns the set of legal moves in state s.
- RESULT(s, a), which returns the state resulting from playing action a in state s.
- TERMINAL-TEST(s), which returns if s is in its terminal state.
- UTILITY(s, p) (referred to as the objective function or payoff function), which returns the final numeric value for a game that ends in terminal state s for a player p.

5.1 α-β pruning

Definition 5.15

Minimax is a decision rule for minimizing the possible loss for a worst case (maximum loss) scenario. It is a recursive algorithm for choosing the next move in an n-player game, usually a two-player game. A value is associated with each position or state of the game. This value is computed by means of a **position evaluation function** and it indicates how good it would be for a player to reach that position. The player then makes the move that maximizes the minimum value of the position resulting from the opponent's possible following moves. If it is A's turn to move, A gives a value to each of their legal moves. A will choose the move with the maximum value of the minimum values resulting from their opponent's possible following moves. If it is B's turn to move, B gives a value to each of their legal moves. B will choose the move with the minimum value of the maximum values resulting from their opponent's possible following moves.

Searching a complete tree takes $O(b^m)$ time, where b is the branching factor and m is the maximum depth of the tree. This is too slow for most games. We can prune the tree to reduce the number of nodes that need to be explored.

Definition 5.16

Pruning is the process of removing parts of a tree that are not relevant to the computation. α-β **pruning** is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

Fri, 15 September 2023, 11:40am – 1:00pm

Lecture 6: Week 3, Friday

6.1 α-β pruning – continued

α-β pruning has the property of reducing the branching factor, b, to its square root (ie, $b \overset{\alpha\beta}{\to} \sqrt{b}$). This is because the algorithm is able to prune away half of the branches at each level of the tree.

Wed, 20 September 2023, 11:40am – 1:00pm

Lecture 7: Week 4, Wednesday

7.1 Gradient Descent

In many real-world scenarios, states are continuous variables.

Example. Suppose you have several cities with nearby airports, and you want to build three new airports, while minimising the distance from each city to its nearest airport. You could model this problem as a continuous optimization problem, where C_i denote the cities that have the airport i as their nearest airport. If you define (x_i, y_i) as the coordinates of the airport i, and (x_c, y_c) as the coordinates of the city c. The aim is to minimise the function f, $f(x_1, y_1, x_1, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$. There is no need to place a $\sqrt{}$ in the function around the distance formula, since the square root is a monotonic function, and the minimum of the function f is the same as the minimum of $\sqrt{f}$.

Solution:-

A common approach to solving optimisation problems involves calculating the gradient; $\nabla f = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3})$. We can apply the update rule $x_i \leftarrow x_i - \alpha \frac{\partial f}{\partial x_i}$ (ie, $x \leftarrow x - \alpha \nabla f$) to each of the coordinates of the airports, where α is the learning rate. We can then repeat this process until the gradient is close to zero. In this scenario, $\frac{\partial f}{x_i} = 2 \sum_{c \in C_i} (x_i - x_c)$. This applies to y_i as well. In order to choose α, we can use Newton's method.

Definition 7.17: Newton's method

In 1669, Sir Isaac Newton discovered a method for finding the roots of a function g that consists of iteratively applying the update rule $x \leftarrow x - \frac{g(x)}{g'(x)}$ until $g(x)$ is close to zero. This method is called **Newton's method**. In optimisation, the goal is to find a point where ∇g is 0; the update rule can be rewritten as $x \leftarrow x - \frac{\nabla g(x)}{\nabla^2 g(x)}$, where $\nabla^2 g(x)$ is the Hessian matrix of g at x as g'' is multivariate. Therefore, the update rule becomes $x \leftarrow x - H_g^{-1}(x) \nabla g(x)$.

Definition 7.18: Gradient descent

Gradient descent is an algorithm for finding the minimum of a function f that takes a real number x and returns a real number $f(x)$. The gradient descent algorithm is as follows:

1. Pick a random value for x.
2. Compute the gradient of $f(x)$ at x.
3. Update x by taking a small step in the direction of the negative gradient.
4. Repeat steps 2 and 3 until x converges.

Definition 7.19: Hessian matrix

A **Hessian matrix** of second derivatives is a matrix whose elements are the second partial derivatives of a function; $H_f(x)$ would have its elements, H_{ij}, given by $\frac{\partial^2 f}{\partial x_i \partial x_j}$. The Hessian matrix is used to determine whether a critical point of a function is a local maximum, local minimum, or saddle point.

In the prior airport example of gradient descent, the Hessian matrix, H_f^{-1}, would be:

$$\begin{pmatrix} \frac{1}{2|C_1|} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2|C_2|} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2|C_3|} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2|C_4|} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2|C_5|} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2|C_6|} \end{pmatrix}_{6\times 6}$$

Fri, 6 October 2023, 11:40am – 1:00pm

Lecture 8: Week 6, Friday

8.1 Constraint Satisfaction Problems (CSPs)

A constraint satisfaction problem consists of three components:

- A set of variables $X_1, X_2, \ldots, X_n$.
- A set of domains $D_1, D_2, \ldots, D_n$, where D_i is the domain of the variable X_i—that is, $X_i \in D_i$.
- A set of constraints that specify allowable combinations of values.

Definition 8.20

To solve a CSP, the state space must be defined:

- **State**: assignment of values to some or all variables.
- **Consistent assignment**: assignment that does not violate any constraints.
- **Partial assignment**: assignment that does not assign values to all variables.
- **Complete assignment**: assignment that assigns values to all variables.
- **Solution**: an assignment which is consistent & complete.

Definition 8.21

The **four colour theorem** (Appel and Haken, 1976) states that any map can be coloured using only four colours, so that no two adjacent regions have the same colour.

Example. If we wished to colour a map of Australia, we could represent the problem as a CSP with the following variables and domains:

- $X = \{\text{WA}, \text{NT}, \text{Q}, \text{NSW}, \text{NSW}, \text{V}, \text{SA}, \text{T}\}$
- $D_i = \{\text{Red}, \text{Green}, \text{Blue}\}$
- $C = \{\text{SA} \neq \text{WA}, \text{SA} \neq \text{NT}, \text{SA} \neq \text{Q}, \text{SA} \neq \text{NSW}, \text{SA} \neq \text{V}, \text{WA} \neq \text{NT}, \text{NT} \neq \text{Q}, \text{Q} \neq \text{NSW}, \text{NSW} \neq \text{V}\}$

CSPs can be more efficient than state space searchers, as constraints can eliminate large portions of the search space. In this case, setting $\text{SA} = \{\text{Blue}\}$ reduces the number of assignments from 3^5 to 2^5—from 243 to 32.

Example. If we wished to schedule the steps of a car assembly line, we could represent the problem as a CSP. The assembly line performs the following steps:

1. Installing the axles, Axle_F & Axle_B, taking 10 minutes each.
2. Affix the wheels, Wheel_{RF}; Wheel_{LF}; Wheel_{RB}; and Wheel_{LB}, taking 1 minute each.
3. Tighten the nuts for each wheel, Nut_{RF}; Nut_{LF}; Nut_{RB}; and Nut_{LB}, taking 2 minutes each.
4. Affix the hubcaps, Cap_{RF}; Cap_{LF}; Cap_{RB}; and Cap_{LB}, taking 1 minute each.
5. Inspect the final assembly, Inspect, which takes 3 minutes.

The problem is finding when each task should be started in the time interval of $[0, 30]$ minutes. We can represent the problem as a CSP with the following variables and domains:

$$\begin{aligned} X = \{&\text{Axle}_F, \text{Axle}_B, \text{Wheel}_{RF}, \text{Wheel}_{LF}, \text{Wheel}_{RB}, \text{Wheel}_{LB}, \\ &\text{Nut}_{RF}, \text{Nut}_{LF}, \text{Nut}_{RB}, \text{Nut}_{LB}, \\ &\text{Cap}_{RF}, \text{Cap}_{LF}, \text{Cap}_{RB}, \text{Cap}_{LB}, \\ &\text{Inspect}\} \\ D_i = [0, 27] \end{aligned}$$

We have the following precedence constraints:

$$\begin{aligned} \text{Axle}_F + 10 \leq \text{Wheel}_{RF}; \quad & \text{Axle}_F + 10 \leq \text{Wheel}_{LF}; \\ \text{Axle}_B + 10 \leq \text{Wheel}_{RB}; \quad & \text{Axle}_B + 10 \leq \text{Wheel}_{LB}; \\ \text{Wheel}_{RF} + 1 \leq \text{Nut}_{RF}; \quad & \text{Nut}_{RF} + 2 \leq \text{Cap}_{RF}; \\ \text{Wheel}_{LF} + 1 \leq \text{Nut}_{LF}; \quad & \text{Nut}_{LF} + 2 \leq \text{Cap}_{LF}; \\ \text{Wheel}_{RB} + 1 \leq \text{Nut}_{RB}; \quad & \text{Nut}_{RB} + 2 \leq \text{Cap}_{RB}; \\ \text{Wheel}_{LB} + 1 \leq \text{Nut}_{LB}; \quad & \text{Nut}_{LB} + 2 \leq \text{Cap}_{LB}. \end{aligned}$$

Additionally, we have the constraint that $X_i + T_i \leq \text{Inspect}$ for each X_i, where T_i is the duration of task X_i. Therefore, the solution to this problem would be an assignment of each variable to a value in D_i such that every constraint is satisfied.

Considering the above example, if we have four workers to install the wheels, but only one pair of workers can install an axle at a time (ie, Axle_F and Axle_B cannot coincide), then we have a **disjunctive constraint**, $(\text{Axle}_F + 10 \leq \text{Axle}_B) \vee (\text{Axle}_B + 10 \leq \text{Axle}_F)$.

Definition 8.22

A **linear program** is an optimisation problem where the objective function and constraints are all linear.

Example. Given the previous example, we could have a linear program that aimed to minimise Inspect such that $\forall X_i \in X - \{\text{Inspect}\} : X_i + T_i \leq \text{Inspect}$;

$$\begin{aligned}
\text{Axle}_F + 10 &\leq \text{Wheel}_{RF}; & \text{Axle}_F + 10 &\leq \text{Wheel}_{LF};\\
\text{Axle}_B + 10 &\leq \text{Wheel}_{RB}; & \text{Axle}_B + 10 &\leq \text{Wheel}_{LB};\\
\text{Wheel}_{RF} + 1 &\leq \text{Nut}_{RF}; & \text{Nut}_{RF} + 2 &\leq \text{Cap}_{RF};\\
\text{Wheel}_{LF} + 1 &\leq \text{Nut}_{LF}; & \text{Nut}_{LF} + 2 &\leq \text{Cap}_{LF};\\
\text{Wheel}_{RB} + 1 &\leq \text{Nut}_{RB}; & \text{Nut}_{RB} + 2 &\leq \text{Cap}_{RB};\\
\text{Wheel}_{LB} + 1 &\leq \text{Nut}_{LB}; & \text{Nut}_{LB} + 2 &\leq \text{Cap}_{LB}.
\end{aligned}$$

While linear programming is appropriate for optimisation problems, CSPs are not necessarily optimisation problems; we're only seeking some solution that satisfies all of the given constraints. For example, the four colour theorem is not an optimisation problem, as there is no objective function to minimise or maximise. CSPs can have non-linear constraints, such as complex constraints on discrete variables (such as in the eight queens puzzle), or constraints on continuous variables (such as in the travelling salesman problem), while linear programs must only have linear constraints.

Wed, 11 October 2023, 11:40am – 1:00pm

Lecture 9: Week 7, Wednesday

9.1 Min-conflicts algorithm

CSPs can often be solved effectively with local search algorithms that use a complete state formulation—that is, the initial state assigns values to every variable, and then the search updates the values one variable at a time. With the eight queens puzzle, for example, one may begin by placing the eight queens in a random configuration, before then proceeding to move one randomly selected conflicting queen at a time until the puzzle is solved. To choose these new values, one may use the **min-conflicts heuristic**—when picking a new value for a given variable, one selects the value that would lead to a minimum number of conflicts with the other variables (and if there are multiple possible minimums, randomly select one).

Algorithm 1 Min-conflicts algorithm

procedure MIN-CONFLICTS(csp, max_steps)
    $current \leftarrow$ an initial complete assignment for csp
    for $i = 1$ to max_steps **do**
        if $current$ is a solution for csp **then**
            return $current$
        end if
        $var \leftarrow$ a randomly chosen, conflicted variable from csp.VARIABLES
        $value \leftarrow$ the value v for var that minimizes CONFLICTS($var, v, current, csp$)
        $current[var] \leftarrow value$
    end for
    return FAILURE
end procedure

On the n-queens problem, the performance of min-conflicts appears roughly constant with respect to n. Another advantage of linear search is its ability to adapt to changes in the problem, such as when solving scheduling problems with online data.

Wed, 18 October 2023, 11:40am – 1:00pm

Lecture 10: Week 8, Wednesday

10.1 Probabilistic Reasoning

Example. If an automated taxi aims to deliver a passenger to the airport on-time, it must choose a plan of action. The airport is 10 miles away, and the taxi has two choices: A_{90}, departing for the airport 90 minutes before the flight departs, and A_{180}, departing for the airport 180 minutes before the flight departs. Plan A_{90} faces a greater risk of being late relative to plan A_{180}. Yet, plan A_{180} is more costly, as it requires the passenger to wait at the airport for a longer period of time. To choose between the two plans, an objective function must be defined, where there is a high cost for missing the flight (eg, -1000), and a low cost for waiting for the flight (eg, -1 per minute). Then, to decide, the taxi should know what will occur during the trip. Both *complexity* and *ignorance* prevent us from doing this; it's simply not feasible to model the world in its entirety, and we don't know what will happen in the future. Thus, we must use **probabilistic reasoning** to make decisions.

There are two common interpretations of probabilities; the *frequentist interpretation*, that a probability is inherent to the system being modeled, and the *Bayesian interpretation*, that a probability is a measure of our uncertainty about the system being modeled. For example, with a coin toss where tossing a fair coin will lead to it being heads-up is $50\% = 0.5$, the frequentist interpretation is that the coin is inherently fair (where tossing the coin n times will lead to the heads-up outcome $\frac{n}{2}$ times as $n \to \infty$), and the Bayesian interpretation is that we are 50% uncertain about the outcome of the coin toss. The issue with the frequentist interpretation is that some probabilistic statements (such as the probabilities of events that have not yet occurred before) do not have frequentist interpretations.

The **sample space** is the set of all possible outcomes; possible outcomes are mutually exclusive, and the sample spaces are exhaustive. A fully-specified probabilistic model associates a probability, $P(\omega)$, with each possible outcome, ω, such that $\forall \omega \in \Omega : 0 \leq P(\omega) \leq 1 \wedge \sum_{\omega \in \Omega} P(\omega) = 1$. Generally, if ϕ is a proposition, $P(\phi) = \sum_{\omega \in \phi} P(\omega)$.

Given a probability distribution, we can make **probabilistic inferences**.

Example. Given the following probability distribution below, $P(\text{Cavity} \vee \text{Toothache}) = 0.108 + 0.012 + 0.072 + 0.008 + 0.016 + 0.064 = 0.28$, $P(\text{Cavity}) = 0.108 + 0.012 + 0.072 + 0.008 = 0.2$. Calculating these probabilities is called **summing out**, or **marginalisation**. Generally, $P(X) = \sum_{Y \in y} P(X \cap Y)$. In this example, $P(\text{Cavity}) = \sum_{Y \in \{(\text{Catch}, \text{Toothache})\}} P(X|Y)P(Y)$.

	Toothache		¬Toothache	
	Catch	¬Catch	Catch	¬Catch
Cavity	0.108	0.012	0.072	0.008
¬Cavity	0.016	0.064	0.144	0.576

A variant of marginalisation is **conditioning**; $P(X) = \sum_{Y \in y} P(X|Y)P(Y)$. Conditional probability can be expressed as $P(A|B)$—read as 'the probability of A given that B has already

occurred'—defined as $P(A|B) = \frac{P(A \cap B)}{P(B)}$. The $P(B|A)$ is $P(B|A) = \frac{P(A \cap B)}{P(A)}$. **Independence** refers to situations where $P(X|Y) = P(X) \vee P(Y|X) = P(Y) \vee P(X \wedge Y) = P(X)P(Y)$.

Definition 10.23: Bayes' Rule

Bayes' Rule is a theorem that follows from the definition of conditional probability. As $P(X \wedge Y) = P(X|Y)P(Y)$ and $P(X \wedge Y) = P(Y|X)P(X)$, it follows that $P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$. It is useful for **inverting conditional probabilities**, where we know $P(A|B)$ and want to find $P(B|A)$. When updating an agent's belief when presented with new evidence, for example, it can calculate $P(\text{Hypothesis}|\text{Evidence}) = P(\text{Hypothesis})\frac{P(\text{Evidence}|\text{Hypothesis})}{P(\text{Evidence})}$. $P(X)$ is the **prior probability**, $P(X|Y)$ is the **posterior probability**, $P(Y|X)$ is the **likelihood**, and $P(Y)$ is the **marginal probability**. As $P(Y) = \sum_{X_i} P(Y|X_i)P(X_i)$, $P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} = \frac{P(Y|X)P(X)}{\sum_{X_i} P(Y|X_i)P(X_i)} = \alpha P(Y|X)P(X)$, where α is the **normalisation constant**. This can also be written as $P(X|Y) \propto P(Y|X)P(X)$.

Wed, 25 October 2023, 11:40am – 1:00pm

Lecture 11: Week 9, Wednesday

11.1 Naive Bayes Algorithm

In the context of email spam filtering, we have a training data set and a testing data set. From the training data, we can calculate $P(\text{Spam}) = \frac{|\text{Spam}|}{|\text{Spam}|+|\text{Ham}|}$, $P(\text{Ham}) = \frac{|\text{Ham}|}{|\text{Spam}|+|\text{Ham}|}$. For each $w \in \text{Spam} \cup \text{Ham}$, where w is a word, we may find $P(w \mid \text{Spam}) = \frac{P(|w \in \text{Spam}|)+1}{|\text{Spam}|+2}$—the $+1$ & $+2$ ensure a probability of 50% when $|w \in \text{Spam}| = 0$ (ie, when the word, w, appears only in the 'ham' set in the training data)—and $P(w \mid \text{Ham}) = \frac{P(|w \in \text{Ham}|)+1}{|\text{Ham}|+2}$. Within the testing data, we have several emails, and we are aiming to determine if they're spam or not. For each $\text{Email} \in \text{Testing Set}$, we have in the email a set of distinct words, X, where every word in X also appears in the training data (and, therefore, has a probability associated with it). Where $X := \text{Email} \cap (\text{Spam} \cup \text{Ham}) : P(\text{Spam} \mid X) \approx \frac{P(\text{Spam}) \prod_{x \in X} P(x|\text{Spam})}{P(\text{Spam}) \prod_{x \in X} P(x|\text{Spam}) + P(\text{Ham}) \prod_{x \in X} P(x|\text{Ham})}$; if $P(\text{Spam} \mid X) > 0.5$ then we classify Email as 'spam', otherwise we classify Email as 'ham'. If $P(w \mid \text{Spam}) = 0$, then $P(\text{Spam} \mid X) = 0$, and if $P(w \mid \text{Ham}) = 0$, then $P(\text{Ham} \mid X) = 1$, requiring us to have added the $+1$s & $+2$s earlier to prevent either of these probabilities from being 0.

Fri, 27 October 2023, 11:40am – 1:00pm

Lecture 12: Week 9, Friday

12.1 Bayesian Networks

Example. A patient experiencing shortness of breath ('dyspnoea') visits the doctor; the doctor knows there may be several causes, such as tuberculosis, bronchitis, or lung cancer. Whether or not the patient is a smoker, and what air conditions the patient has been exposed to are factors. An X-ray may be performed, which may indicate that the patient has lung cancer or tuberculosis. Assuming the patient is experiencing dyspnoea, doesn't smoke, and has been in clean air, but that the X-ray comes back positive, what is the probability that the patient has cancer?

We may use a probabilistic model to compute

$$\begin{aligned} &P(\text{Cancer} \mid \text{Dyspnoea}, \text{X-ray}_{\text{positive}}, \neg\text{Smoker}, \text{Pollution}_{\text{low}}) \\ &\approx \frac{\text{Cases}(\text{Cancer}, \text{Dyspnoea}, \text{X-ray}_{\text{positive}}, \neg\text{Smoker}, \text{Pollution}_{\text{low}})}{\text{Cases}(\text{Dyspnoea}, \text{X-ray}_{\text{positive}}, \neg\text{Smoker}, \text{Pollution}_{\text{low}})}. \end{aligned}$$

Estimating this is difficult, due to the large number of parameters. Instead, simpler relations may be used to find marginal probabilities, which can be used to calculate the joint probability distribution.

Node name	Type	Values
Pollution	Binary	$\{\text{low}, \text{high}\}$
Smoker	Boolean	$\{T, F\}$
Cancer	Boolean	$\{T, F\}$
Dyspnoea	Boolean	$\{T, F\}$
X-ray	Binary	$\{\text{positive}, \text{negative}\}$

Table 1: The nodes in the Bayesian network for the previous example.

Definition 12.24: Bayesian Network

A **Bayesian network** is a type of directed acyclic graph where each node corresponds to a random variable; each node, X_i, has a conditional probability distribution $P(X_i \mid \text{Parents}(X_i))$. These nodes have marginal probabilities and are independent of the other nodes in the graph. The joint probability distribution of the network is $P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid \text{Parents}(X_i))$, where x_i is the value of X_i, and X_i is the ith node in the network.

The previous example may be solved using a Bayesian network, where,

$$\frac{\text{Cases}(\text{Cancer}, \text{Dyspnoea}, \text{X-ray}_{\text{positive}}, \neg\text{Smoker}, \text{Pollution}_{\text{low}})}{\text{Cases}(\text{Cancer}, \text{Dyspnoea}, \text{X-ray}_{\text{positive}}, \neg\text{Smoker}, \text{Pollution}_{\text{low}}) + \text{Cases}(\neg\text{Cancer}, \text{Dyspnoea}, \text{X-ray}_{\text{positive}}, \neg\text{Smoker}, \text{Pollution}_{\text{low}})}$$

$$= \frac{P(\text{Cancer} \mid (\neg\text{Smoker} \wedge \text{Pollution}_{\text{low}})) \cdot P(\text{Dyspnoea} \mid \text{Cancer}) \cdot P(\text{X-ray}_{\text{positive}} \mid \text{Cancer}) \cdot P(\neg\text{Smoker}) \cdot P(\text{Pollution}_{\text{low}})}{(P(\text{Cancer} \mid (\neg\text{Smoker} \wedge \text{Pollution}_{\text{low}})) \cdot P(\text{Dyspnoea} \mid \text{Cancer}) \cdot P(\text{X-ray}_{\text{positive}} \mid \text{Cancer}) \cdot P(\neg\text{Smoker}) \cdot P(\text{Pollution}_{\text{low}})) + (P(\neg\text{Cancer} \mid (\neg\text{Smoker} \wedge \text{Pollution}_{\text{low}})) \cdot P(\text{Dyspnoea} \mid \neg\text{Cancer}) \cdot P(\text{X-ray}_{\text{positive}} \mid \neg\text{Cancer}) \cdot P(\neg\text{Smoker}) \cdot P(\text{Pollution}_{\text{low}}))}.$$

Fri, 3 November 2023, 11:40am – 1:00pm

Lecture 13: Week 10, Friday

13.1 Speech Recognition

The general flow for the process of speech recognition is: analog speech signal $\overset{\text{signal processing}}{\rightarrow}$ digital speech signal $\overset{\text{machine learning model}}{\rightarrow}$ word sequence.

Using Bayes's rule, it's possible to find $P(\text{Words} \mid \text{Signal})$, where $P(\text{Words})$ is the language model and $P(\text{Signal} \mid \text{Words})$ is the acoustic model. The language model provides the likelihood of word sequences (eg, the words 'recognise speech' may be a more appropriate caption generated from some audio than 'wreck a nice beach'), and the acoustic model provides the likelihood of the signal given the word sequence (eg, given the word 'nice', we may find the likelihood that it is pronounced like some audio). This overall speech recognition model is $\hat{W} = \underset{W \in L}{\arg\max} \frac{P(O|W)P(W)}{P(O)} \propto \underset{W \in L}{\arg\max} P(O \mid W)P(W)$, where O is the observed signal, W is the word sequence, L is the set of all possible word sequences, $P(O \mid W)$ is the acoustic model (ie, observation likelihood), $P(W)$ is the language model, and $\hat{W} = \underset{W \in L}{\arg\max}$ is the 'best match' metric. We may ignore the $P(O)$ as it would be the same for every word sequence; it is a constant.

$P(\text{Words})$ is a joint probability, and can be expressed using the chain rule—$P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1 w_2) \ldots P(w_n \mid w_1, \ldots, w_{n-1})$. This isn't feasible to compute, so **first-order Markov assumption** is used. This is the relationship that $P(w_i \mid w_1, \ldots, w_{i-1}) \approx P(w_i \mid w_{i-1})$, essentially claiming that the past and future are roughly independent. This simplifies the language model into the **bigram model**, $P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_2) \ldots P(w_n \mid w_{n-1})$, which relates consecutive pairs of words. The **trigram model** considers a two-word **context window** (ie, $P(w_i \mid w_1, \ldots, w_{i-1}) \approx P(w_i \mid w_{i-1}, w_{i-2})$). A unigram model's probabilities would simply be $P(w_i) = \frac{1}{|L|}$ (ie, all words in the language's dictionary are equally likely, as no context is being considered). Accuracy of the probabilities increases as the context window increases, but the computational complexity increases as well. A language model's *prediction accuracy* improves given more training data and parameters.

The acoustic model has two parts, being $P(\text{Sounds} \mid \text{Word})$ (which may be specified as a Markov model; the probability of a sequence of sounds given Word), and $P(\text{Signal} \mid \text{Sounds})$ (which may be specified as a **hidden Markov model**; the probability of some digital signal values given a sequence of sounds).

Fri, 10 November 2023, 11:40am – 1:00pm

Lecture 14: Week 11, Friday

14.1 Face Recognition

One may think of facial recognition as a template matching problem, where:

1. We obtain training data of face images—all centred and of the same size—being $I_1, I_2, \ldots, I_M$.
2. We represent every image I_i as a vector Γ_i—ie, an $N \times N$ square image (represented as a matrix) is transformed into an $N^2 \times 1$ column vector.
3. We compute the average face vector, $\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i$.
4. We subtract the mean face from each face; $\Phi_i = \Gamma_i - \Psi$.
5. We compute the covariance matrix, $C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T = AA^T$ (resulting in an $N^2 \times N^2$ matrix), where $A = \begin{bmatrix}\Phi_1 & \Phi_2 & \cdots & \Phi_M\end{bmatrix}$ (an $N^2 \times M$ matrix). This is a reduction of the data into lower-dimensional space.
6. We compute the eigenvectors u_i of AA^T:
   (a) If we consider A^TA (an $M \times M$ matrix), we can compute the eigenvectors, ν_i of A^TA.
   (b) $A^TA\nu_i = \mu_i\nu_i \Rightarrow AA^TA\nu_i = \mu_iA\nu_i \Rightarrow CA\nu_i = \mu_iA\nu_i$; we may express $A\nu_i$ as u_i. AA^T (which may have up to N^2 eigenvalues and eigenvectors), and A^TA (which may have up to M eigenvalues and eigenvectors) have the same non-zero eigenvalues, and their eigenvectors are related by $u_i = A\nu_i$. The M eigenvalues of A^TA, along with their related eigenvectors, correspond to the M *largest* eigenvalues & eigenvectors of AA^T.
   (c) We compute the M best eigenvectors of AA^T : $u_i = A\nu_i$, where u_i has been normalised such that $\|u_i\| = 1$.
7. We keep the eigenvectors associated with the K largest eigenvalues.

Each normalised face, Φ_i, in the training data set is a linear combination of the K best eigenvectors; $\hat{\Phi}_i - \Psi = \sum_{j=1}^{K} w_j u_j$, where $w_j = u_j^T\Phi_i$—the u_js are called **eigenfaces**. Our basis vector, Ω_i, represents each normalised training face, where $\Omega_i = \begin{bmatrix} w_1^i \\ w_2^i \\ \vdots \\ w_K^i \end{bmatrix}, i = 1, 2, \ldots, M.$

Given Γ, an unrecognised, centred face image of the same size as the training images, we may perform facial recognition using the following steps:

1. Normalise $\Gamma : \Phi = \Gamma - \Psi$.
2. Considering that u_i is $N^2 \times 1$, and $u_i^T\Phi$ is $N^2 \times 1$, we project onto the eigenspace $\hat{\Phi} = \sum_{j=1}^{K} w_i u_i$, where $w_i = u_i^T\Phi$.

3. We represent Φ as $\Omega = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \end{bmatrix}$.
4. We compute the minimum Mahalanobis distance within the face space between the Ωs of the training images and the Ω of the unrecognised image; $e_r = \min_l \|\Omega - \Omega^l\| = \sum_{i=1}^{K} \frac{1}{\lambda_i}(w_i - w_i^l)^2$. Each face cluster (ie, each Ω^l) is calculated from several images of the same face in the training set.
5. If $e_r < T_r$, where T_r is a threshold, then the unrecognised image is recognised as face l from the training set.

Detecting whether or not a face is present in an image, Γ, follows a similar process:

1. Compute $\Phi = \Gamma - \Psi$.
2. Compute $\hat{\Phi} = \sum_{i=1}^{K} w_i u_i$, where $w_i = u_i^T \Phi$.
3. Compute the distance from the face space, $e_d = \|\Phi - \hat{\Phi}\|$.
4. If $e_d < T_d$, where T_d is a threshold, then Γ is an image of a face.

Notes