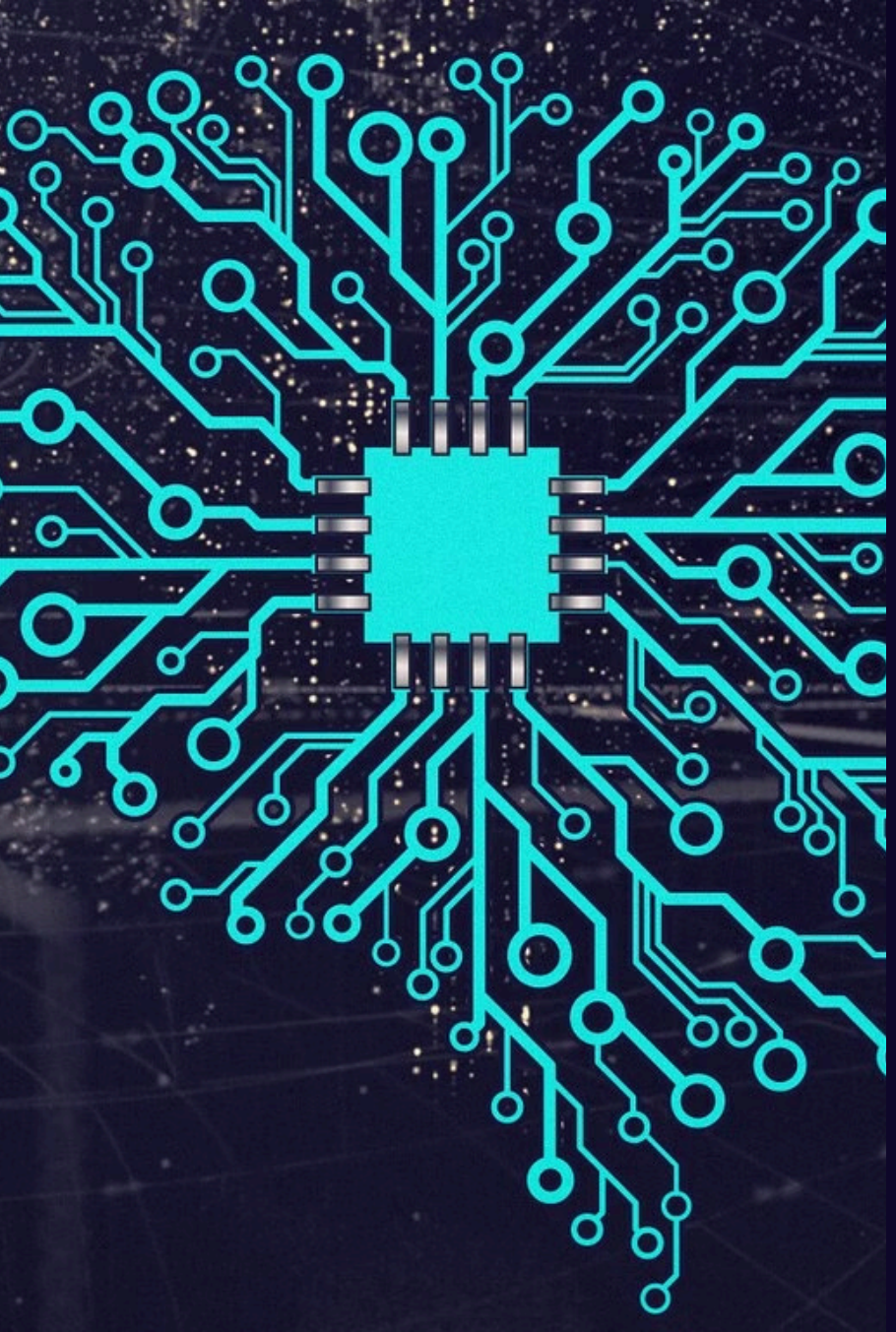


Artificial Intelligence: Search & Machine Learning

Welcome to a practical journey through two fundamental pillars of AI: intelligent search algorithms and the foundations of machine learning. This week, we'll explore how computers find optimal solutions and learn from data.

What is Informed Search?

Informed search is a problem-solving technique where an algorithm uses knowledge about the problem domain to guide its exploration. Unlike blind searching that tries every possibility, informed search makes smart decisions about which paths to explore first, dramatically reducing the time needed to find solutions.

Core Concept

Imagine you're lost in a city looking for a pizza restaurant. Blind search would check every building randomly. Informed search uses knowledge—like knowing restaurants cluster in certain neighborhoods or being able to see a sign from blocks away—to guide your search efficiently. The algorithm evaluates options using domain-specific information and always pursues the most promising path first.

Real-World Applications

- **GPS Navigation:** Finding the fastest route by evaluating distance, traffic patterns, and road conditions
- **Game AI:** Chess engines evaluating which moves are most likely to lead to victory
- **Medical Diagnosis:** Systems narrowing down possible diagnoses based on symptoms and test results
- **Web Search:** Ranking results based on relevance to your query

Greedy Search: The Quick Decision Maker

Greedy search is an algorithm that always makes the locally optimal choice, hoping to find a global optimum. At each step, it picks the option that looks best right now, without reconsidering previous decisions. It's "greedy" because it wants the best immediate reward.

How Greedy Search Works

The algorithm evaluates all available next steps using a heuristic (a rule of thumb or estimate). It always chooses the single best option and moves forward. This continues until reaching a goal or dead end. Greedy search is **fast**—it doesn't waste time exploring inferior options—but it can get stuck in local traps and miss the truly optimal solution.

Simple Python Example

```
def greedy_search(graph, start, goal, heuristic):
    current = start
    visited = set()

    while current != goal:
        visited.add(current)
        neighbors = graph[current]

        # Filter out already visited nodes
        unvisited = [n for n in neighbors if n not in visited]

        if not unvisited:  # Dead end
            return None

        # Greedily pick neighbor with lowest heuristic value
        current = min(unvisited, key=lambda n: heuristic[n])

    return current
```

Pros and Cons

✓ Fast & Simple	□ Not Complete	□ Not Optimal
Low memory, quick decisions	May fail to find solutions	Often misses best path

A* Search: The Smart Pathfinder

A* (A-star) search is one of the most popular informed search algorithms. It combines the speed of greedy search with the accuracy of exhaustive search by intelligently evaluating each path. A* is used everywhere—from video game character movement to robot navigation.

The Secret Formula

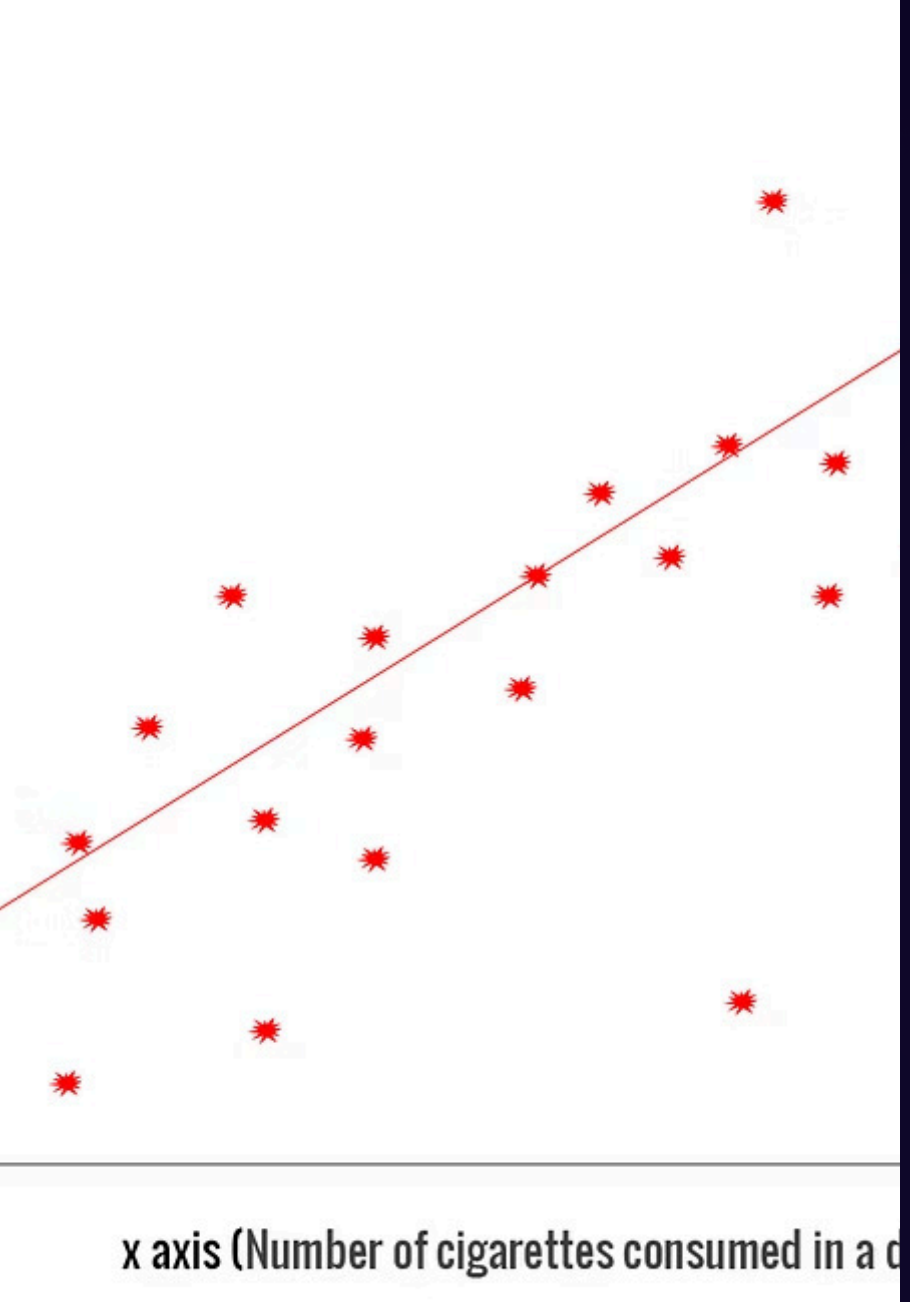
A* evaluates each position using: **f(n) = g(n) + h(n)**

- **g(n)** = actual cost from start to current position (what you've already paid)
- **h(n)** = estimated cost from current position to goal (what you might pay)
- **f(n)** = total estimated cost of the best path through this position

A* always explores the position with the lowest f-score first. This balances moving toward the goal (h) with considering the actual path cost (g), making it both efficient and optimal.

Python Implementation

```
import heapq

def a_star_search(graph, start, goal, heuristic):
 open_set = [(0, start)] # (f_score, node)
 came_from = {}
g_score = {start: 0}
visited = set()

while open_set:
_, current = heapq.heappop(open_set)

if current == goal:
return reconstruct_path(came_from, current)

visited.add(current)

for neighbor in graph[current]:
if neighbor in visited:
continue

tentative_g = g_score[current] + 1

if neighbor not in g_score or tentative_g < g_score[neighbor]:
came_from[neighbor] = current
g_score[neighbor] = tentative_g
f_score = tentative_g + heuristic[neighbor]
heapq.heappush(open_set, (f_score, neighbor))

return None
```



Comparing Search Algorithms

Understanding the differences between search strategies helps us choose the right tool for each problem. Let's examine how Greedy and A* stack up against each other.

Greedy Search

Strategy: Only looks at estimated distance to goal

Speed: Very fast, minimal memory

Completeness: Not guaranteed to find solutions

Optimality: Not optimal—may find long paths

Best For: Quick approximations when accuracy isn't critical

A* Search

Strategy: Balances actual cost and estimated distance

Speed: Moderate—depends on heuristic quality

Completeness: Always finds a solution if one exists

Optimality: Optimal—guaranteed to find shortest path

Best For: When you need reliable, optimal solutions

Introduction to Machine Learning

Machine Learning (ML) is a branch of artificial intelligence where computers learn patterns from data rather than being explicitly programmed. Instead of writing instructions for every scenario, we feed examples to an algorithm, and it discovers the underlying patterns and rules.

The Core Idea

Traditional programming: **Program + Data → Output**

Machine Learning: **Data + Output → Program**

With ML, we show the computer many examples (input-output pairs), and it figures out the relationship. This approach is powerful because it automatically adapts to new data and can handle complexity that would be impossible to code manually.

Real-World ML Examples

- **Email Spam Filters:** Learn what spam looks like from millions of examples
- **Recommendation Systems:** Netflix learns your movie preferences from your viewing history
- **Facial Recognition:** Systems learn to identify faces from thousands of labeled images
- **Medical Prediction:** Algorithms predict disease risk from patient data
- **Language Translation:** Systems learn patterns from translated text pairs

The Three Types of Machine Learning

Machine learning divides into three main categories based on how the algorithm learns from data. Each approach is suited to different kinds of problems and data availability.

Supervised Learning

You provide labeled examples where the correct answer is known. The algorithm learns to map inputs to outputs.

Example: Predicting house prices using features like size, location, age.

Unsupervised Learning

You provide raw data without labels. The algorithm discovers hidden patterns and structure.

Example: Grouping customers by shopping habits to find market segments.

Reinforcement Learning

An agent learns by taking actions and receiving rewards or penalties. It discovers optimal strategies through trial and error.

Example: Teaching a robot to walk by rewarding successful steps.

Supervised Learning: Learning from Examples

Supervised Learning is the most common ML approach. You train an algorithm with labeled data—examples where you already know the correct answer. The algorithm learns the relationship between inputs and outputs, then predicts outputs for new, unseen inputs.

How It Works

- **Training Phase:** Feed labeled examples to the algorithm
- **Learning Phase:** Algorithm adjusts internal parameters to minimize errors
- **Testing Phase:** Evaluate performance on new, unseen data
- **Prediction Phase:** Use trained model to predict outputs for future inputs

Simple Python Example: Predicting House Prices

```
from sklearn.linear_model import LinearRegression
import numpy as np

# Training data: (square_feet, price)
X_train = np.array([[1000], [1500], [2000], [2500]])
y_train = np.array([200000, 300000, 400000, 500000])

# Create and train model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
new_house_size = np.array([[1800]])
predicted_price = model.predict(new_house_size)
print(f"Predicted price: ${predicted_price[0]:,.0f}")
```

Common Supervised Learning Tasks

Classification

Predicting categories

- Spam detection (spam/not spam)
- Disease diagnosis (sick/healthy)
- Sentiment analysis (positive/negative)

Regression

Predicting numbers

- Stock price prediction
- Weather forecasting
- House price estimation

Unsupervised Learning: Finding Hidden Patterns

Unsupervised Learning works with unlabeled data—you don't provide correct answers. The algorithm's job is to discover structure, patterns, or relationships hidden in the data. It's like detective work: finding clues without being told what to look for.

Key Unsupervised Learning Techniques

→ Clustering

Grouping similar data points together. Example: Netflix groups users with similar movie preferences to make recommendations.

→ Dimensionality Reduction

Simplifying complex data by removing redundant features. Example: Visualizing 100-dimensional customer data in 2D to spot patterns.

→ Anomaly Detection

Finding unusual or outlier data points. Example: Credit card fraud detection identifies suspicious transactions.

Python Example: K-Means Clustering

```
from sklearn.cluster import KMeans
import numpy as np

# Customer data: spending and visit frequency
X = np.array([
 [100, 2], # Low spender, visits rarely
 [150, 3], # Low spender, visits occasionally
 [5000, 50], # High spender, visits frequently
 [4500, 48], # High spender, visits frequently
 [120, 1], # Low spender, rare visitor
])

# Find 2 customer segments
kmeans = KMeans(n_clusters=2, random_state=42)
labels = kmeans.fit_predict(X)

# Results show which customers belong to which segment
for i, label in enumerate(labels):
 print(f"Customer {i}: Segment {label}")
```

Your AI Learning Journey

You've now explored two essential AI foundations: intelligent search algorithms that find optimal solutions efficiently, and machine learning approaches that automatically discover patterns in data. These concepts form the building blocks for almost all modern AI systems.

What You've Learned

1 Informed Search

Using domain knowledge to guide problem solving—Greedy finds quick approximations, A* guarantees optimal solutions.

2 Machine Learning Fundamentals

Three learning paradigms: Supervised learning from labeled examples, unsupervised discovery of hidden patterns, and reinforcement learning through feedback.

3 Practical Applications

Real algorithms with real code that power recommendation systems, navigation, spam filters, and countless AI systems you use daily.

Next Steps

Continue exploring by implementing these algorithms yourself. Start with simple datasets, experiment with different approaches, and build intuition. The best way to understand AI is to get hands-on experience. Good luck on your AI learning journey!