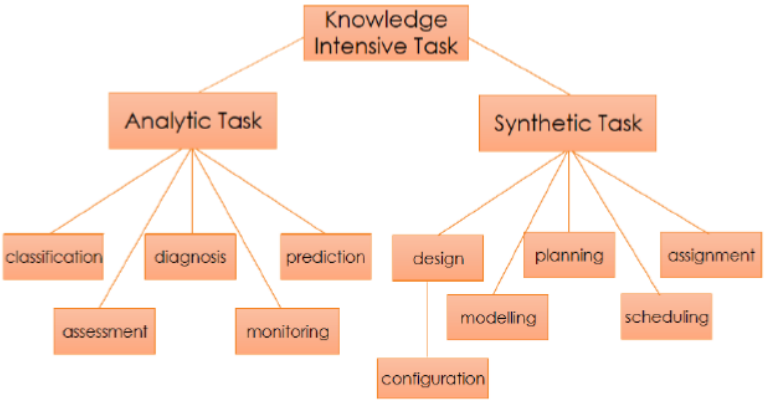
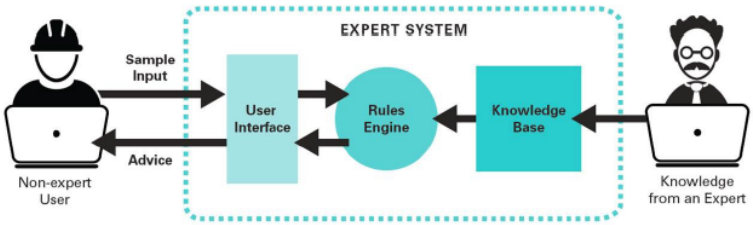
Reasoning Systems

- AI for Knowledge Intensive Task



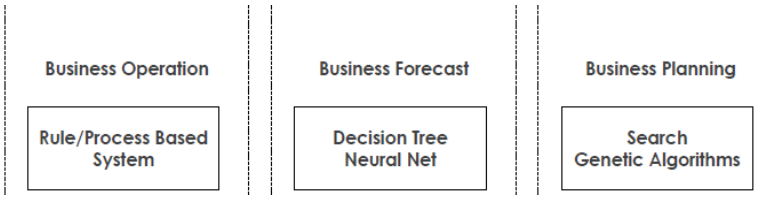
  - Analytic Tasks
    - System/Solution to be analysed pre exists, but usually not completely "known“.
    - Input: some data to trigger the system (e.g. patient symptoms)
    - Output: some characterization or behaviours about the system (e.g. cause of illness)
    - definition
      - A reasoning system which can retrieve/calculate a (known) solution from/using a knowledge base, containing solutions . Obtain knowledge (using knowledge discovery & ML); Represent knowledge (based on various forms of reasoning types); Obtain solution by using knowledge on new problem scenario (inference engine or calling functions)
    - Tasks
      - Identification, Classification, Prediction, Clustering/Grouping
    - Techniques
      - Heuristic Business Rules
      - Decision Trees
      - Similarity Based Reasoning
      - Fuzzy Logic
      - Knowledge Discovery
    - example
      - medical diagnosis
  - Synthetic Tasks
    - System/Solution does not yet exist.
    - Input: requirements about system to be constructed
    - Output: constructed system description
    - definition

        - A reasoning system which can generate a (new) solution using a knowledge base, containing goal test or solution evaluation knowledge . Generate candidate solutions systematically; Test candidate solution using goal test knowledge; Rank candidate solutions based on evaluation function/knowledge; Alter solutions to generate new candidate solutions; Repeat Test candidate until a termination criteria, e.g. one minute, is satisfied;
    - Tasks
        - Planning, Scheduling, Optimization, Design, …
    - Techniques
        - Uninformed (brute force / blind) Search
        - Informed (heuristic) Search
        - Simulations
        - Genetic Algorithms
        - Reinforcement Learning
    - example
        - E.g. Coronavirus (COVID 19) vaccination creation
- Expert System

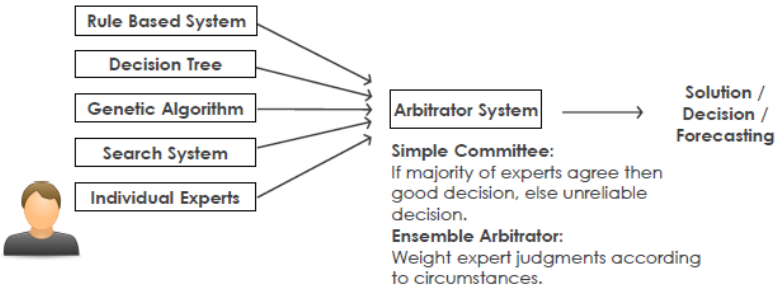


    - Limits
        - Expensive and time consuming!
        - Experts are never cheap
        - •Bad in handling sophisticated
        - sensory inputs (like signals,
        - images)
        - •Possible to make dumb decision
        - since it just goes through rules; no
        - common sense in the system
        - •System not easy to be updated
- Hybrid Reasoning Systems(Hybrid Soft computing Systems)
    - The sum is greater than the parts
    - Hybrid Intelligent systems are intelligent systems which combine two or more machine reasoning techniques:
        - Deductive Reasoning used in Analytic Problem Solving

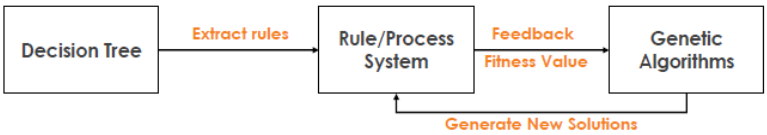
        - Techniques: Knowledge-driven Rule/Process Systems; Information Retrieval; Fuzzy Logic; Belief Desire Intention (BDI) Cognitive framework
    - Inductive Reasoning used in Data Mining & Machine Learning
        - Techniques: Decision Tree; Association Rule; Neural Networks; Bayesian Net; Knowledge Graph; Reinforcement Learning
    - Planning & Optimization used in Synthetic Problem Solving Techniques: Search; Genetic Algorithms; Constrain Satisfaction; Swarm Intelligence;
- Four broad types of system architectures
    - 1. Independent Sub-systems



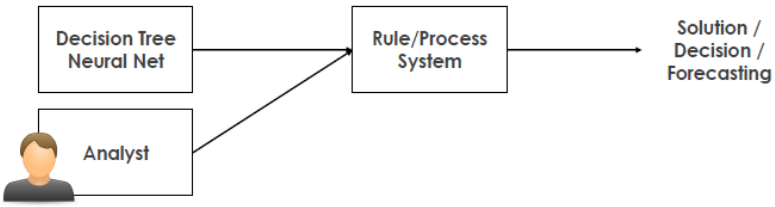
        - Sub divide problem into independent parts.
        - Each is solved by an appropriate technique or sub system.
        - No cooperation is required, e.g. a decision support system has several independent sub systems (functions).
    - 2. Competing Experts



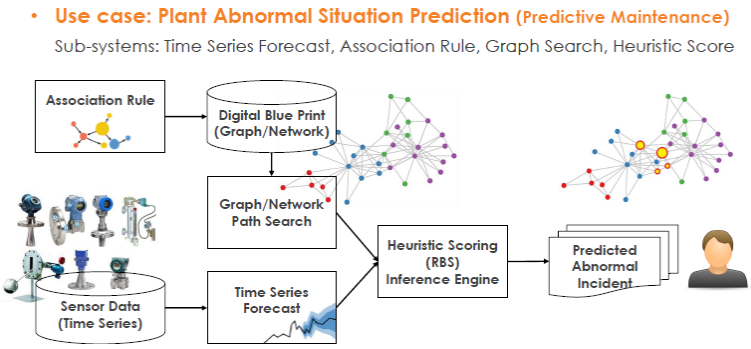
        - Different solution strategies (experts) offer alternative solutions.
        - Another process decides which solution to accept or how to combine the solutions, e.g. majority vote algorithm or a rule based system. (auction, tender, bidding)
    - 3. Self-tuning



        - One technique is used to tune or learn the architecture for another, e.g. Decision Tree used to learn a ruleset, e.g. trading strategies; Genetic Algorithms used to optimise initial solution, e.g. investment portfolio, based on predicted/simulated value from rule based system, in order to maximize trading profit.
    - 4. Cooperating Experts



- Pooled Expertise: Different techniques/sub systems work together as a team to produce a single solution. No single technique/expert is sufficient alone, e.g. Machine learning and human analyst both provide inputs required to forecast macroeconomics based on business rules and weights, under different market conditions.
- example



- Uninformed Search Techniques
  - Solving Problem by Search
  - Synthesis of a new valid solution is performed by searching through the (search/solution) space , which defines all possible solutions;
  - Each possible solution is evaluated to see whether it is valid and/or the optimum (best solution found so far)
    - e.g. a valid employee schedule, a valid vehicle delivery route, an optimal (shortest) vehicle delivery
  - Validity of solution involves satisfaction of a set of constraints/business requirements on the solution variables (goal test)
  - Optimality is measured by a user defined evaluation function which measures the “goodness” of the solution,
    - e.g. the shorter delivery route the better;
  - Pipeline
    - (1)Create a pool of solution candidates (search space)
    - (2)Pick up one candidate solution from pool
    - (3)Check whether this candidate is valid (constraints satisfied?)
      - (3)=True If valid, continue
      - (3)=False If not valid, go to (
    - (4) Check whether this candidate is the best till now (optimal solution?)
      - (4)=True If best, save this solution as the best then continue

        - (4)=False If not best, discard this solution then continue
    - (5) Go to (2). Repeat the cycle until a stopping criteria is
- Tree
    - Nodes in tree represent
      an initial state
      an intermediate state
      a final state (feasible solution, or failure)

    - Connection between nodes represents a search step
    - Search is illustrated using a search space with a particular restricted structure
    - Solutions (search space) can be represented as a Tree
    - Algorithm
        - DFS: Always prefers to search deeper in the search tree rather than wider.

          Algorithm Pseudo Code

          (1) Set N to be a list (open list/frontier list) of initial nodes

          (2) If N is empty, then exit and signal failure

          (3) Set n to be the first node in N, and remove n from N

          (4) Check n:

          (4.1) If n is a goal node, then exit and signal success

          (4.2) Otherwise, add the children of n to the **front** of **N** then go to step **(2)**
        - BFS: Explores all the nodes at a given depth before processing deeper in the search tree.

          Algorithm Pseudo Code

          (1) Set N to be a list (open list/frontier list) of initial nodes

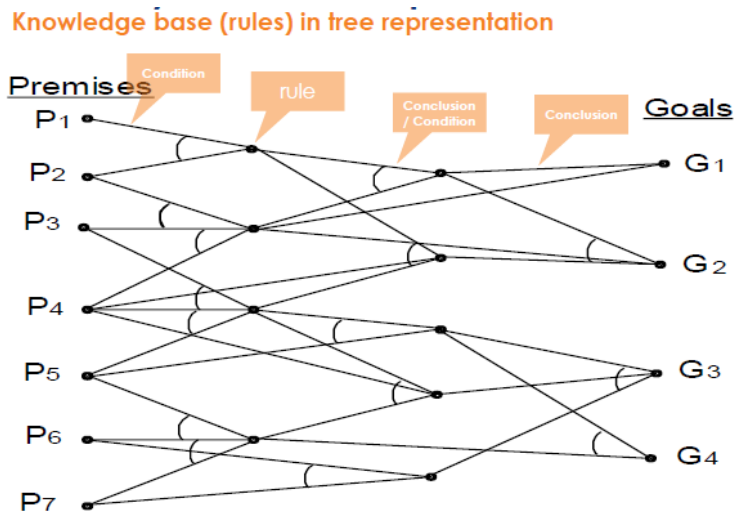
          (2) If N is empty, then exit and signal failure

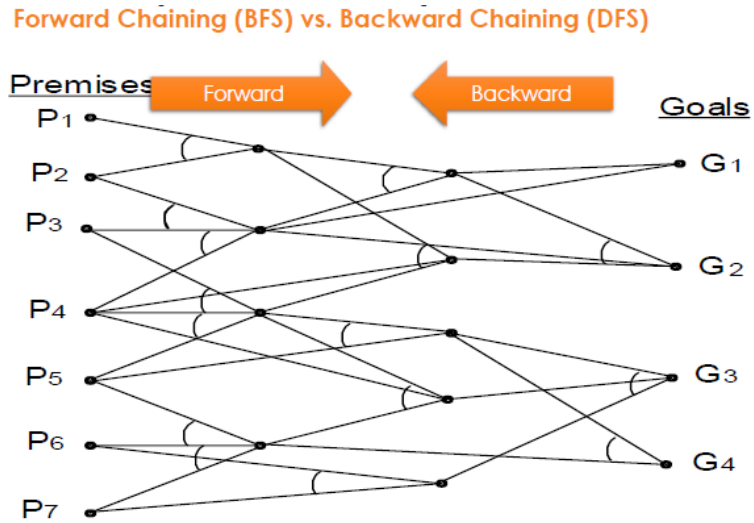
          (3) Set n to be the first node in N, and remove n from N

          (4) Check n:

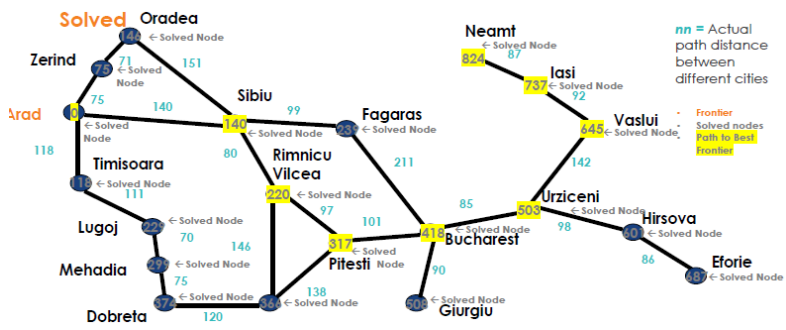
          (4.1) If n is a goal node, then exit and signal success

          (4.2) Otherwise, add the children of n to the **end** of **N** then go to step **(2)**
    - example





- Graph
  - A model (data structure) for word concepts in human cognition, consisting of nodes, links and labels
    - Nodes (vertices) represent objects, concepts, or situations. They can be instances (individual objects as in Knowledge Graph) or classes (generic objects as in Semantic Web)
    - Links (edges) between nodes represent a relationship/predicate
    - Labels (attributes)
      - Labels on nodes indicate the name of the object,concept, etc.
      - Labels on links describe the type of relationship between nodes
  - Algorithm
    - Dijkstra's algorithm(BFS + calculation of path cost)



- 1.Pick the start and end nodes and add the start node to the set of solved nodes with a value of 0 . Solved nodes are the set of nodes with the shortest known path from the start node. The start node has a value of 0 because it is 0 path length away from itself.
- 2.Traverse breadth first from the start node to its nearest neighbors (frontier nodes) and record/calculate the path length against each neighboring node.
- 3.Pick the shortest path to one of these neighbors and mark that node as solved. In case of a tie, Dijkstra's algorithm picks at random.
- 4.Visit the nearest neighbors (frontier nodes) to the set of solved nodes and record/accumulate the path lengths from the start node against these new neighbors. Don't visit any neighboring nodes that have already been solved, as we already know the shortest paths to them.
- 5.Repeat steps three and four until the end/destination node has been marked as " solved node ", or a termination criteria is satisfied

-

Various graph algorithms:

- **Pathfinding**: Paths are fundamental to graph analytics and algorithms, so this is where we'll start our chapters with specific algorithm examples. Finding shortest paths is probably the most frequent task performed with graph algorithms and is a precursor for several different types of analysis. The shortest path is the traversal route with the fewest hops or lowest weight. If the graph is directed, then it's the shortest path between two nodes as allowed by the relationship directions.
- **Centrality**: Centrality is all about understanding which nodes are more important in a network. But what do we mean by importance? There are different types of centrality algorithms created to measure different things, such as the ability to quickly spread information versus bridging distinct groups.
- **Community Detection**: Connectedness is a core concept of graph theory that enables a sophisticated network analysis such as finding communities. Most real-world networks exhibit substructures (often quasi-fractal) of more or less independent subgraphs.

- Knowledge Graph
  - A knowledge graph is a knowledge base that uses a graph structured data model or topology to store, integrate, represent, and query data/knowledge.
  - Normally, the edge/link is relationship(predicate) predicate). The node/vertex is entity/instance
  - Learning Objectives
    - Learn build knowledge graph (KG) in graph database
    - Learn do inference and query in knowledge graph
  - Package to use
    - Spacy, neo4j(database), openNRE(model to extract relationship between two entities from a sentence.)
  - Technique

- Entity & relationship extraction
- Knowledge graph reasoning

- Construction Flow

1. Extract Entities

```
for ent in doc.ents:
    if ent.label_ in ['PERSON', 'GPE'] and ent.text not in names_of_entities:
        names_of_entities.append(ent.text)
        entities.append(ent)
```

2. Generate Nodes

```
for ent in entities:
    if exist_ent.get(ent.text) is None:
        exist_ent[ent.text] = ent.text
        query = (
            "MERGE (node: "+ent.label_+" {name: $name})"
            "RETURN node"
        )
```

3. Extract Relationships

```
for i in range(len(entities)):
    for j in range(i + 1, len(entities)):
        text_i = entities[i].text
        text_j = entities[j].text
        loc_h = sentence.find(text_i)
        loc_t = sentence.find(text_j)
        result = model.infer({'text': sentence, 'h': {'pos': (loc_h, loc_h + len(text_i))},
                              't': {'pos': (loc_t, loc_t + len(text_j))}})
```

4. Generate Relationships

```
if record not in exist_relationship:
    exist_relationship.append(record)
    if record[3] > 0.8:
        query = (
            "MATCH (n1 {name: $name1})"
            "MATCH (n2 {name: $name2})"
            "MERGE (n1) - [r:"+record[2]+"] -> (n2)"
            "RETURN n1, n2, r"
        )
```

- Propositional logic

1. WHEN BodyCoverType = HasHair THEN AnimalType = Mammal
2. WHEN FeedType = FeedMilk THEN AnimalType = Mammal
3. WHEN BodyCoverType = HasFeather THEN AnimalType = Bird
4. WHEN MoveType = CanFly AND ReproduceType = LayEgg THEN AnimalType = Bird
5. WHEN AnimalType = Mammal AND FeedType = EatMeat THEN AnimalType = Carnivore
6. WHEN AnimalType = Mammal AND ToothType = HasPointedTeeth AND FootType = HasClaws AND EyeType = HasForwardEyes THEN AnimalType = Carnivore
7. WHEN AnimalType = Mammal AND FeedType = EatGrass THEN AnimalType = Herbivore
8. WHEN AnimalType = Mammal AND FootType = HasHooves THEN AnimalType = Herbivore
9. WHEN AnimalType = Carnivore AND ColorType = HasColorTawny AND MarkingType = HasDarkSpots THEN AnimalName = Cheetah
10. WHEN AnimalType = Carnivore AND ColorType = HasColorTawny AND MarkingType = HasDarkStripes THEN AnimalName = Tiger
11. WHEN AnimalType = Herbivore AND ColorType = HasColorTawny AND MarkingType = HasDarkSpots AND NeckType = HasLongNeck THEN AnimalName = Giraffe
12. WHEN AnimalType = Herbivore AND ColorType = HasColorBlackWhite THEN AnimalName = Zebra
13. WHEN AnimalType = Bird AND MoveType = CanWalk AND ColorType = HasColorBlackWhite AND NeckType = HasLongNeck THEN AnimalName = Ostrich
14. WHEN AnimalType = Bird AND MoveType = CanWwim AND ColorType = HasColorBlackWhite THEN AnimalName = Penguin
15. WHEN AnimalType = Bird AND MoveType = CanFly AND ColorType = HasColorBlackWhite THEN AnimalName = Albatross

- Predicate logic

$$p \rightarrow q \equiv \neg p \vee q$$

$$\forall x P(x) = P(1) \wedge P(2) \wedge \cdots \wedge P(k)$$

$$\exists x P(x) = P(1) \vee P(2) \vee \cdots \vee P(k)$$

- Reasoning Using Informed Search

- Machine Learning ≈ Search

Supervised Machine Learning:

Loss function, Accuracy, F1, Precision, Recall, MSE, RMSE, MAE, etc.

Unsupervised Machine Learning:

Silhouette coefficient, Dunn's index, etc.

Reinforcement learning:

Reward function, Q function, etc.

Search:

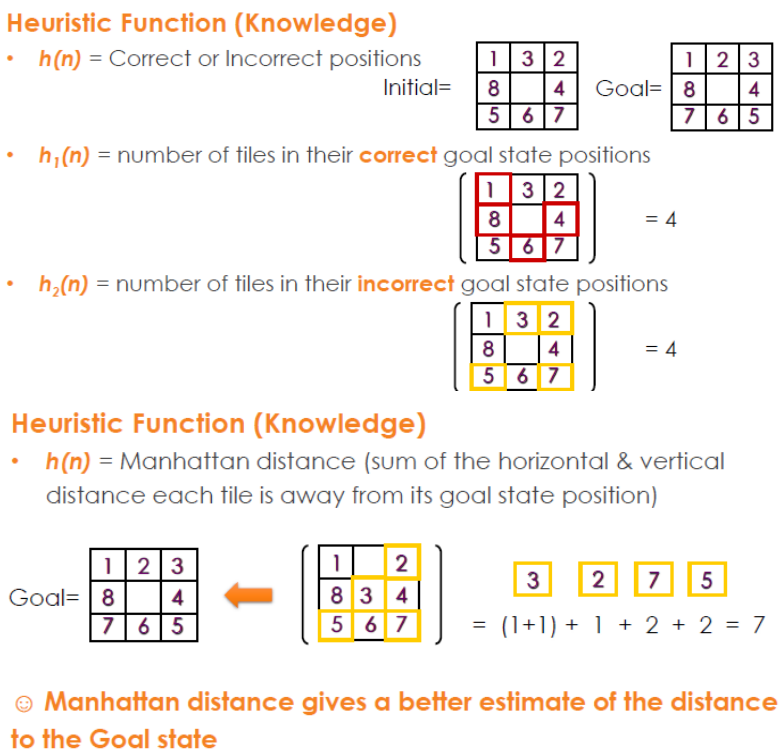
Evaluation function, Heuristic function, Goal test function, etc.

Optimization:

Objective function, Utility function, Expectation maximization, etc.

- Machine learning procedure:

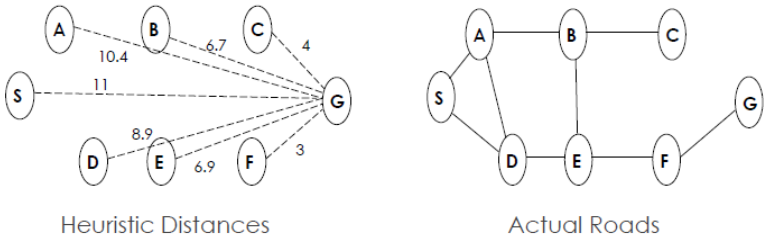
        - Learn the best knowledge (a function + a set of coefficients), which can accurately (minimum error) predict Salary based on Education & Working
    - Search / Optimization procedure :
        - Use ‘generate and test’ method to obtain the knowledge/solution a function + the best set of coefficients among all possible coefficients space), which can minimize the outcome of an evaluation function , e.g. the error) between predicted value and target value, so as to more accurately predict Salary based on Education & Working
- Use Heuristics (domain specific knowledge)
    - Basic Idea
        - It works by firstly sorting the list of nodes, then explore them orderly, according to their optimality (best score) determined by an evaluation function f (n)
            - h(n) = estimated cost of the cheapest path from node n to a goal node. Or estimated degree of difference between the current states/solutions and ultimate goal state.
            - By convention, the lower the heuristic value the more promising the node: better to check first. h(n) = 0 when n is goal
            - When there is no other information available but only the heuristics, f(n) = h(n)
    - example
        - matrix



        - city

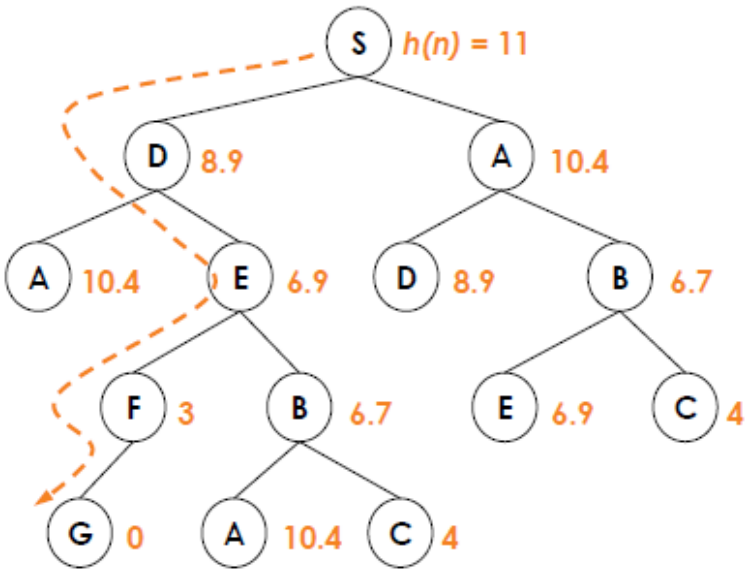
Heuristic Function (Knowledge)

- *h(n)* = "straight-line" distance between each city & the goal (This is useful estimation or heuristic.)



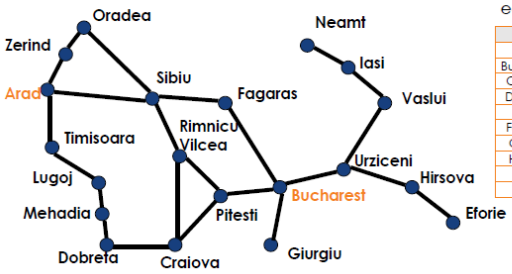
- Typical Best first Strategies
  - Use heuristics only : Hill Climbing, Tabu search
  - Use heuristics and past cost : A*, Late Acceptance
  - Use heuristics and randomness : Simulated Annealing, GA

- Hill Climbing Search (HC Greedy Best First Search)



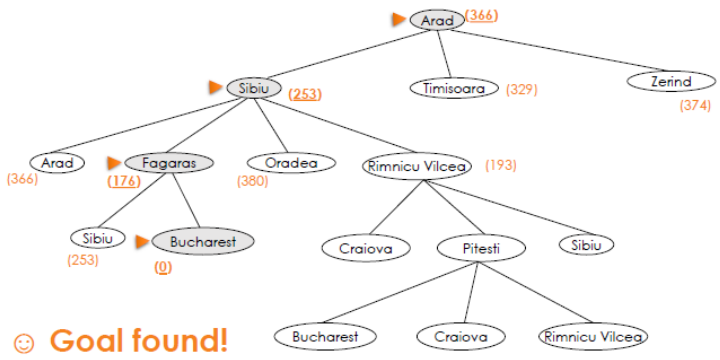
- Minimize the cost to reach the goal state by expanding the node that is closest to the goal.
- Using only the heuristic values for evaluation function:f(n)= h(n)
- Select search node with min(f(n)) at each step.
- Follow a single path all the way to a goal, but can back track when it hits a dead end.
- example

• Arad → Bucharest



h(n) =Straight-line distance from each city to **Bucharest**

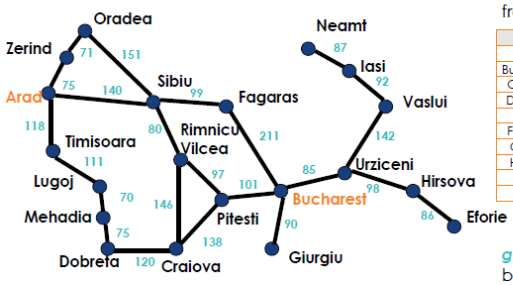
City	Distance	City	Distance
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



- A Star Search (A*)
  - This strategy evaluates each search node by combining g(n), the past (path) cost from the start node to current node n , and
  - h(n) , the estimated future (path) cost: the cheapest path/cost from current node n to a goal node
  - Estimated total cost of the cheapest solution through n

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

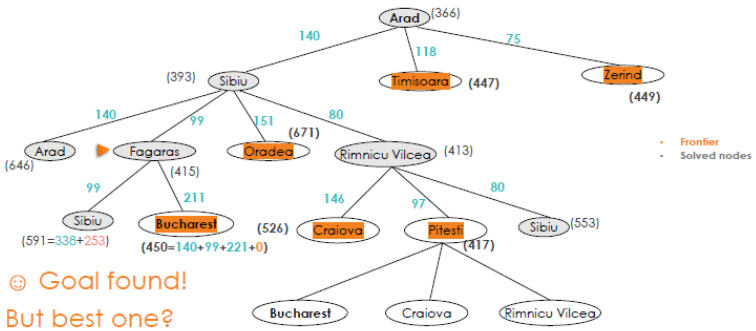
    $g(n)$ is exactly known, but $h(n)$ is only an estimation with possible error.
  - example
    - **Arad → Bucharest**



$h(n)$ = Straight-line distance from each city to **Bucharest**

City	Distance	City	Distance
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

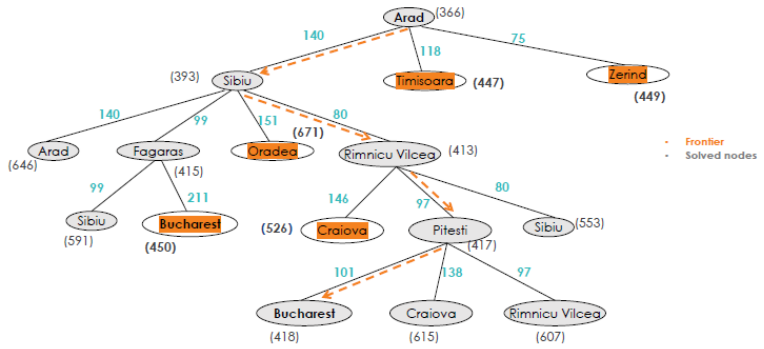
$g(n)$ = Actual path distance between adjacent cities



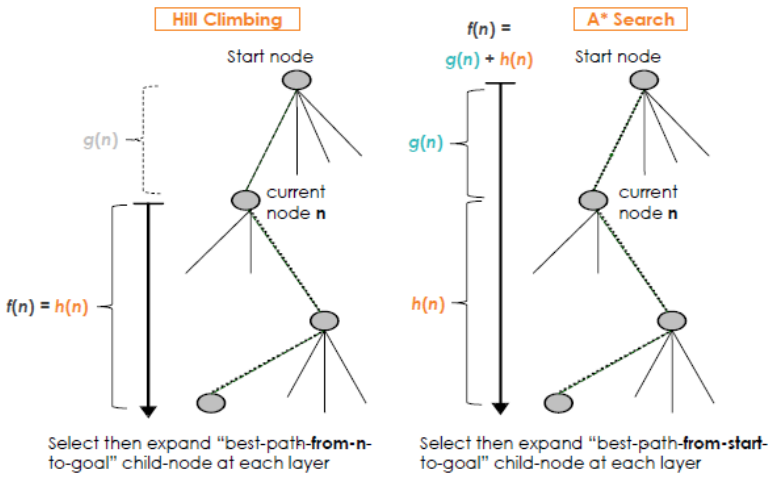
☺ Goal found!

But best one?

(Path distance/cost **450** is not in front of frontier node list)



- A* vs Hill



- Semantic Reasoner
  - Overview
    - A semantic reasoner, reasoning engine, rules engine, or simply a reasoner, is a piece of software able to infer logical consequences from a set of asserted facts or axioms. The notion of a semantic reasoner generalizes that of an inference engine, by providing a richer set of mechanisms to work with. The inference rules are commonly specified by means of an ontology language, and often a description logic (DL) language.
  - Knowledge Representation: Graph Models:
    - Semantic Network (WordNet)
      - A semantic network, or frame network is a knowledge base that represents semantic relations between concepts in a network. This is often used as a form of knowledge representation. It is a directed or undirected graph consisting of vertices, which represent concepts, and edges, which represent semantic relations between concepts, mapping or connecting semantic fields. A semantic network may be instantiated as, for example, a graph database or a concept map.
      - expressed as semantic triples, e.g. (subject , predicate , object), subject points to object through predicate.
      - WordNet® is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms synsets), each expressing

a distinct concept. Synsets are interlinked by means of conceptual semantic and lexical relations. The

- semantic relations

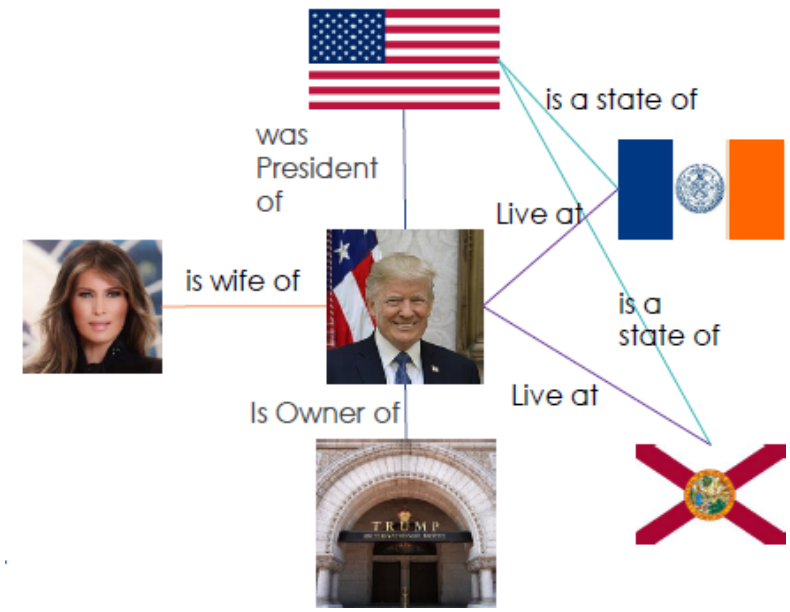
  Semantic relations:
  - Nouns
    - hypernyms: Y is a hypernym of X if every X is a (kind of) Y (X=Red, Y=Color)
    - hyponyms: Y is a hyponym of X if every Y is a (kind of) X (X= Color, Y= Red)
    - coordinate terms: Y is a coordinate term of X if X and Y share a hypernym (X=Red, Y=Yellow)
    - meronym: Y is a meronym of X if Y is a part of X (X= Tree, Y=Leaves)
    - holonym: Y is a holonym of X if X is a part of Y (X=Leaves, Y=Tree)
  - Verbs
    - hypernym: the verb Y is a hypernym of the verb X if the activity X is a (kind of) Y (X=Listen, Y=Perceive)
    - troponym: the verb Y is a troponym of the verb X if the activity Y is doing X in some manner (X=Beat, Y=Hit)
    - entailment: the verb Y is entailed by X if by doing X you must be doing Y (X= Chew, Y=Eat)
    - coordinate terms: those verbs sharing a common hypernym

- knowledge structure

  Dog, domestic dog, Canis familiariscanine, canid
  ∟Carnivore
  ∟Placental, placental mammal, eutherian, eutherian mammal
  ∟Mammal
  ∟Vertebrate, craniate
  ∟Chordate
  ∟Animal, animate being, beast, brute, creature, fauna
  ∟...

- Limitations
  - 1. Some of the semantic relations are more suited to concrete concepts than to abstract concepts.
  - 2. Many of the concepts in WordNet are specific to certain languages.
  - 3. WordNet does not include information about the etymology or the pronunciation of words and it contains only limited information about usage.
  - 4.WordNet aims to cover most everyday words and does not include much domain specific terminology.

- Knowledge Graph



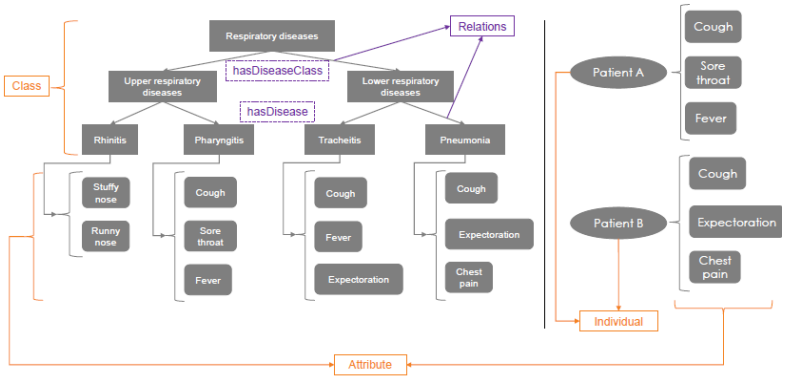
- A knowledge base that uses a graph structured data model or topology to store, integrate, represent, and query data/knowledge.

- Knowledge graphs are often used to store interlinked descriptions of entities objects, events, situations or abstract concepts with free form semantics.
- Normally, the edge/link is relationship(predicate). The node/vertex is entity/instance.
- Advantages: Concise and structed information container (knowledge representation) than strings/narratives:

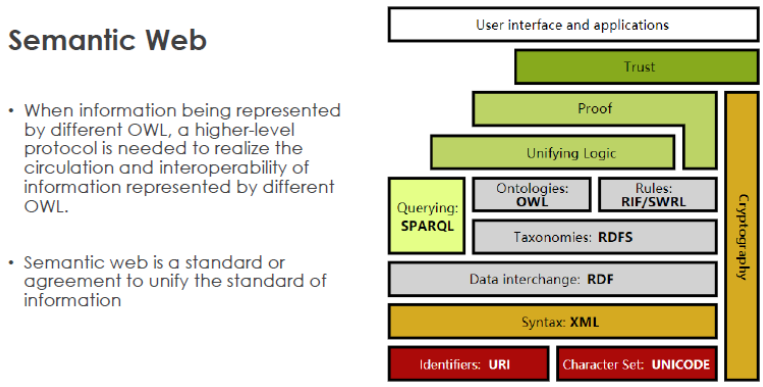
- Ontology

  Ontology Components:
  - **Classes**: objects in ontology
  - **Properties**: Attributes a Class has
  - **Relations**: semantic annotations between objects/classes
  - **Individuals**: instantiation of a class/object
  - Restrictions
  - Rules
  - Function Terms
  - Axioms
  - Events

  - an explicit specification of a conceptualization
  - Definition
    - In computer science and information science, an ontology encompasses a representation, formal naming and definition of the categories, properties and relations between the concepts, data and entities that substantiate one, many or all domains of discourse. More simply, an ontology is a way of showing the properties of a subject area and how they are related, by defining a set of concepts and categories that represent the subject.
    - An ontology consists of a set of axioms which place constraints on sets of individuals (called "classes") and the types of relationships permitted between them.
  - Rules (when then) can be defined based on the respiratory diseases,for automatic inference: diagnosis.
    - WHEN:
      - A person has cough,Sore throat and fever
    - THEN:
      - The person has Pharyngitis
  - Restrictions
    - Used to restrict the input form, for example, a person cannot have both male and female attributes
  - example



- Knowledge Representation: Frameworks/Standards:
  - Web Ontology Language(OWL)
    - Definition
      - the Web Ontology Language is a family of knowledge representation languages for authoring ontologies.
      - Ontologies are a formal way to describe taxonomies and classification networks, essentially defining the structure of knowledge for various domains: the nouns representing classes of objects and the verbs representing relations between the objects.
      - OWL are built upon the World Wide Web Consortium (W3C) 's Resource Description Framework (RDF)
      - The language is based on the RDF framework of the World Wide Web Consortium's (W3C). In other words, the language used to describe the ontology in the RDF specification can be called OWL.
      - Use OWL to express the knowledge of ontology in the RDF/XML framework, realize the machine readability of ontology, and provide the feasibility for using a reasoner to reason about ontology
  - Semantic Web

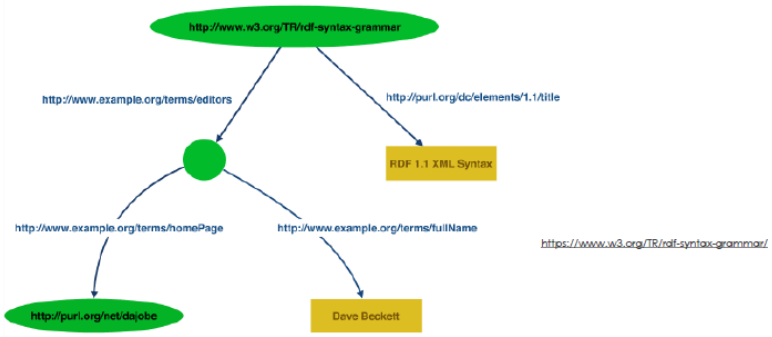


    - The Semantic Web is an extension of the World Wide Web through standards set by the World Wide Web Consortium (W3C). The goal of the Semantic Web is to make Internet data machine readable.
    - It is a framework incorporating: OWL, SPARQL, SWRL, RDFS, RDF, XML, etc.
  - Resource Description Framework (RDF)

- The Resource Description Framework (RDF) is a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata data model. It has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax notations and data serialization formats. It is also used in knowledge management applications.
- The RDF data model is subject predicate object, known as triples.

Subject	Predicate	Object
Péter Szeredi	author	The Semantic Web Explained-The Technology and Mathematics behind Web 3.0

- RDF graph is a directed graph composed of triples (subject,predicate, object), subject points to object through predicate.



- vocabulary

Classes		Properties	
rdf	rdfs	rdf	rdfs
rdf:XMLLiteral	rdfs:Resource	rdf:type	rdfs:subClassOf
rdf:Property	rdfs:Literal	rdf:first	rdfs:subPropertyOf
rdf:Statement	rdfs:Class	rdf:rest	rdfs:domain
rdf:Alt, rdf:Bag, rdf:Seq	rdfs:Datatype	rdf:value	rdfs:range
rdf:List	rdfs:Container	rdf:subject	rdfs:label
rdf:nil	rdfs:ContainerMembershipProperty	rdf:predicate	rdfs:comment
		rdf:object	rdfs:member
			rdfs:seeAlso
			rdfs:isDefinedBy

- Serialization formats:
  - Turtle
    - Terse RDF Triple Language (Turtle) is a syntax and file format for expressing data in the Resource Description Framework (RDF) data model.
    - Turtle was defined as a subset of Notation3 (N3) language.
    - Unlike full N3, which has an expressive power that goes much beyond RDF, Turtle can only serialize valid RDF graphs.
    - Turtle is an alternative to RDF/XML
    - Turtle does not rely on XML and is generally recognized as being more readable and easier to edit manually than its XML counterpart.
  - N Triples
  - N Quads
  - JSON LD
  - N3 or Notation3

- Notation3, or N3 as it is more commonly known, is a shorthand non-XML serialization of Resource Description Framework models.
- N3 is much more compact and readable than XML RDF notation.

RDF/XML

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
  <rdf:Description
rdf:about="http://en.wikipedia.org/wiki/Tony_Benn">
    <dc:title>Tony Benn</dc:title>
    <dc:publisher>Wikipedia</dc:publisher>
  </rdf:Description>
</rdf:RDF>
```

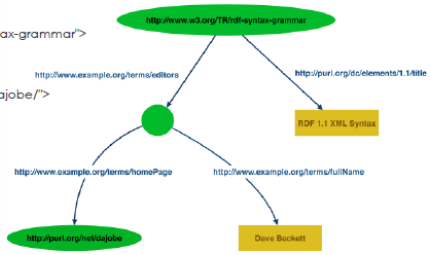
Notation3

```
@prefix dc: <http://purl.org/dc/elements/1.1 />.

<http://en.wikipedia.org/wiki/Tony_Benn>
  dc:title "Tony Benn";
  dc:publisher "Wikipedia".
```

- RDF/XML
  - A XML syntax defined for RDF in terms of Namespaces in XML, the XML Information Set and XML Base is called RDF/XML

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
      </ex:homePage>
      <ex:fullName>Dave Beckett</ex:fullName>
    </rdf:Description>
  </ex:editor>
  <dc:title>RDF 1.1 XML Syntax</dc:title>
</rdf:Description>
```



- RDF/JSON
- Query and inference languages of RDF:SPARQL

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT?name
       ?email
WHERE
  {
    ?person a        foaf:Person .
    ?person foaf:name        ?name .
    ?person foaf:mbox        ?email .
  }
```

- Result representation

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT?name
       ?email
WHERE
  {
    ?person a        foaf:Person .
    ?person foaf:name        ?name .
    ?person foaf:mbox        ?email .
  }
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

_:x foat:name   "XIAO Yuchao"
_:x foat:mbox   "yuchao_xiao@u.nus.edu"
```

```
<rdf:RDF xmlns="http://xmlns.com/foaf/0.1/">
  <rdf:Description>
    <foat:name>XIAO Yuchao</foat:name>
    <foat:mbox>yuchao_xiao@u.nus.edu</foat:mbox>
  </rdf:Description>
</rdf:RDF>
```

- Extensible Markup Language (XML)
  - Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human readable and machine readable.
  - Component

Markup and Content

Markup: Begin with "<" or "&". End with ">" or ";".

Content: Strings of characters that are not markup are content.

Tag: One markup construct that has three type:

Start tag: <Book Class>

End tag: </Book Class>

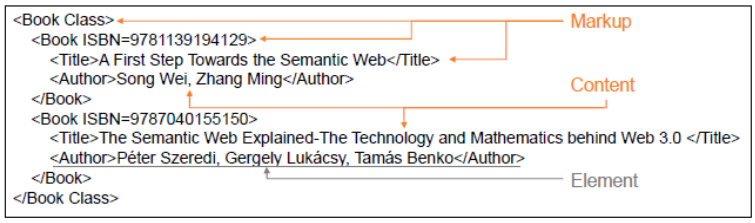
Empty-element tag:<line-break />

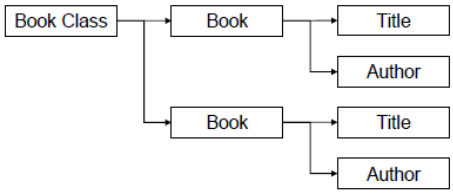
Element: Start tag + Content + End tag:

<Author>Péter Szeredi, Gergely Lukácsy, Tamás Benko</Author>

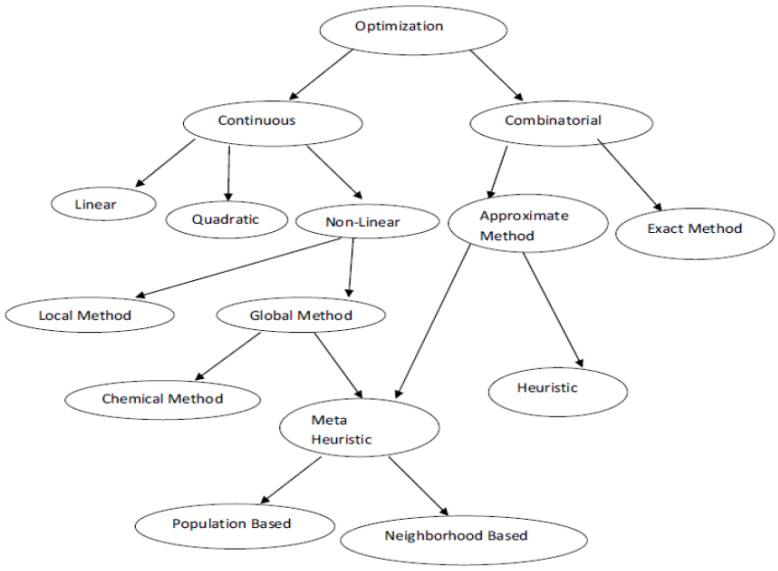
Attribute: An attribute is a markup construct consisting of a name–value pair that exists within a start-tag or empty-element tag:

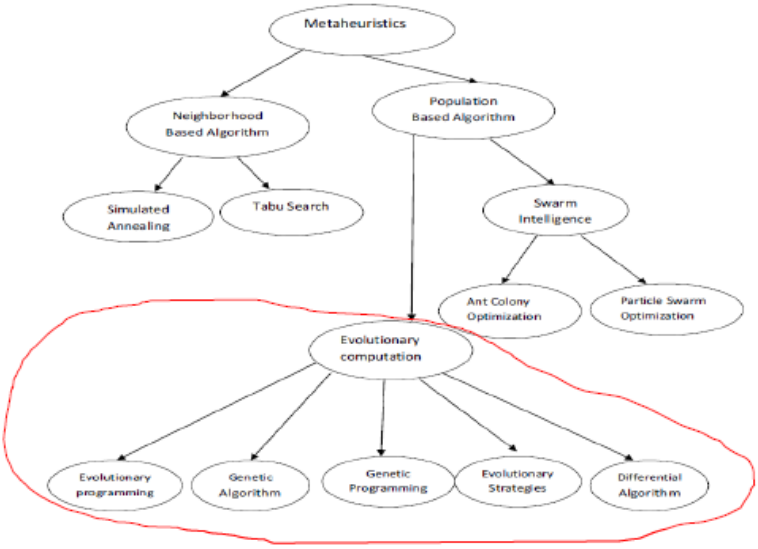
<Book ISBN=9781139194129>



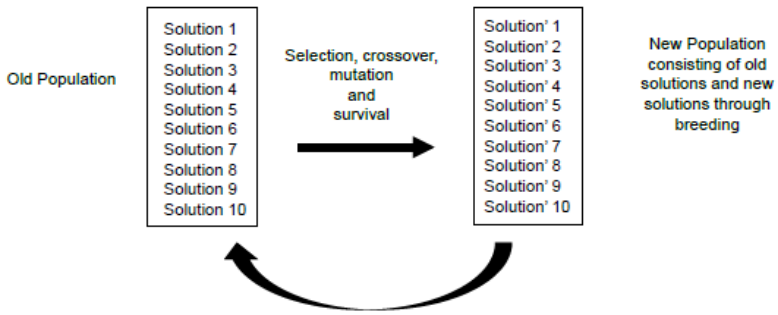


- Summary
  - Semantic Web is a set of (knowledge representation) technologies that want to make the data on the web readable and understood by machines. The domain that semantic web is the web. Therefore the URI is in the bottom of the technologies stack.
  - Semantic Network e.g. WordNet, is a graph model to store the information. If you can model your data as a graph with node and edge, you can think it as a semantic network.
  - Knowledge Graph & Ontology, are also graph models. However, it emphasis the store of all data on the level of both schema and individual. Therefore, it has no essential difference with Semantic network.
  - Knowledge Base is an instantiated storage of a domain knowledge. You can think it as a database. It has not to be a graph model, if you can store the knowledge in some way, you can call it as knowledge base.
- Open World Assumption T he OWA assumption states that everything we don't know is undefined (whether true or false)
- OPTIMIZATION TECHNIQUES
  - Brief
    -



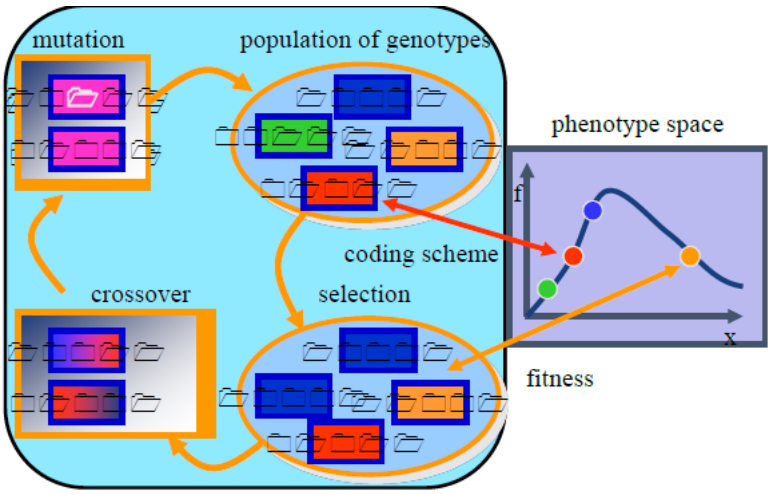


- Evolutionary Computation
  - Genetic Algorithms



    - idea
      - Genetic algorithms are general-purpose search and optimization algorithms that use principles inspired by natural population genetics to evolve solutions to problems
      - GAs operate on a population of individuals representing potential solutions to a given problem.

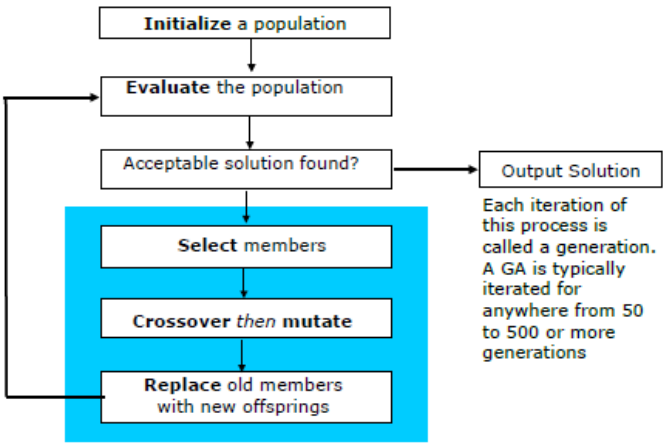
- GAs seek to produce better (fitter) individuals (solutions) by combining the better of the existing ones (through breeding -crossover and mutation).
- Evolution as Optimisation



- Pseudocode

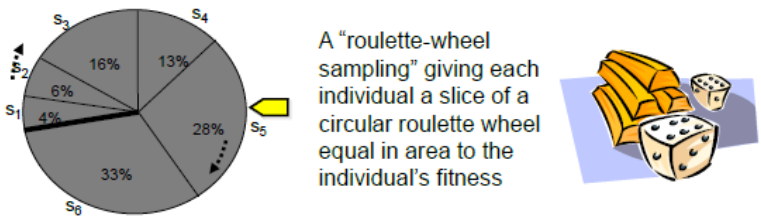
```
Generate the initial population P(0);
t=0;
repeat
        Evaluate the fitness of each individual in P(t);
        Select parents from P(t) based on their fitness;
        Applying crossover and mutation to parents to create O(t);
        Obtain population P(t + 1) by combining P(t) and O(t);
        t =t + 1;
until termination criterion satisfied;
```

- GA Search Process



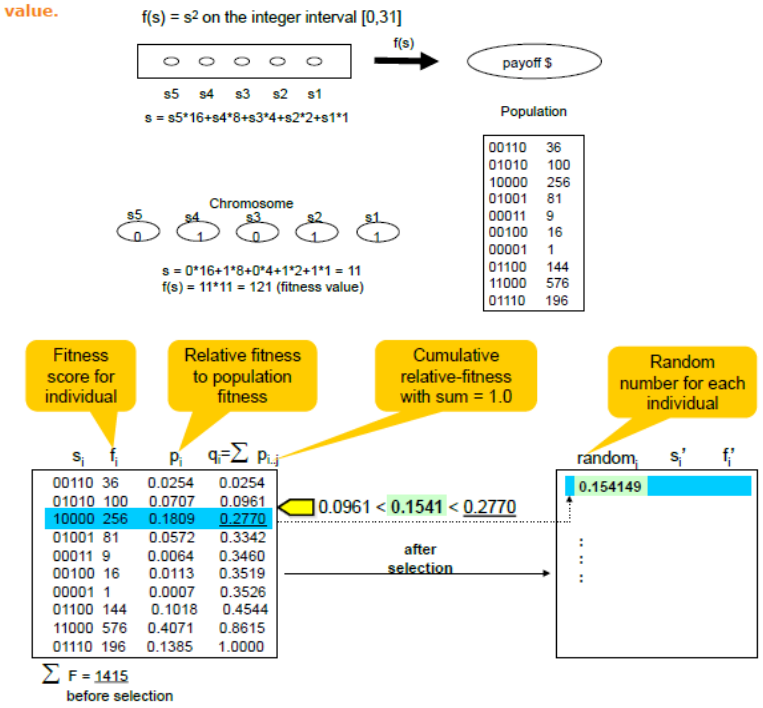
- Initial population
  - Can be initialized using whatever knowledge is available about the possible solutions
  - Should represent a random sample of the search space

- Each member of the population is evaluated and assigned a measure of its fitness as a solution
- New population
  - Structures in the current population are selected for replication based on their relative fitness
  - Genetic operators (crossover, mutation) are performed to generate the new population
- Selection
  - Reproduction focuses attention on high fitness individuals, thus exploiting on the available fitness information.
  - High-performing structures might be chosen several times for replication, while poor-performing structures might not be chosen at all.
  - Selection Methods
    - Roulette Wheel

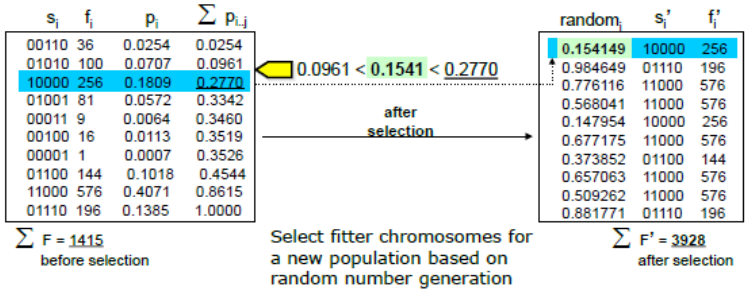


      - Select new population with respect to the probability distribution based on fitness values
      - A roulette wheel with slots sized according to the fitness is used
        - Calculate the fitness value f(si) for each chromosome si
        - Find the total fitness of the population
          - F = Σif(si)
        - Calculate the probability of selection pifor each chromosome si
          - pi= f(si)/F
        - Calculate a cumulative probability qifor each chromosome si
          - qi= Σjpj
      - Spin the roulette wheel, each time a single chromosome for a new population is selected in the following way
        - Generate a random (float) number r from the range [0..1]
        - If r<q1 then select the first chromosome; otherwise select the i-thchromosome si such that qi-1<r <qi
      - example

Consider a black box switching problem which concerns a black box device with a bank of five input switches. For every setting of the five switches, there is an output signal f; mathematically f = f(s) where s is a particular setting of the five switches. The objective of the problem is to set the switches to obtain the maximum possible f value.

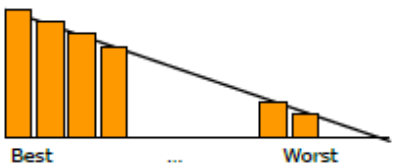


Generation 0
Probability distribution calculation based on fitness values



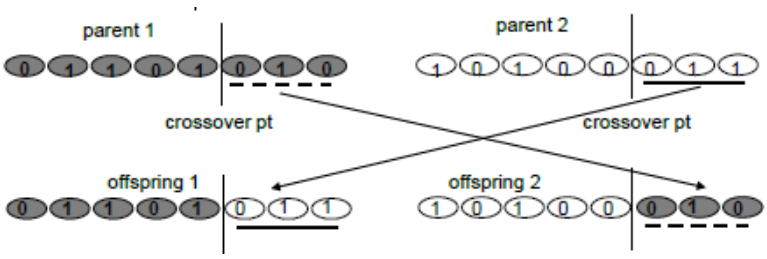
This represents an improvement of +2513

- Tournament selection method
  - Choosing individuals for reproduction randomly.
  - Binary tournament
    - Two individuals are chosen at random and the better of the two individuals is selected.
  - Larger or N-tournament
    - N individuals are chosen at random from the population, with the best being selected for reproduction.
- Rank selection method

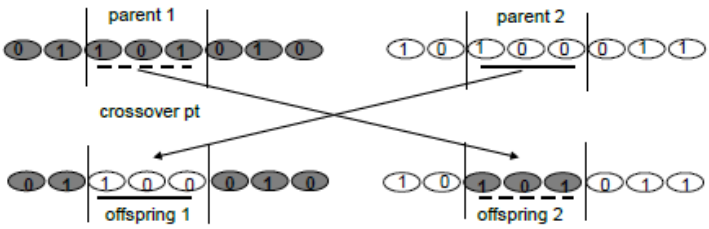


- The population is ordered according to the measured fitness values. A new fitness value is then ascribed, inversely related to their rank.
- Linear ranking
  - Each chromosome in the population is ranked in increasing order of fitness, from 1 to N, and assigned a new subjective fitness using a linear ranking function.
- Exponential ranking
  - First chromosome assigned fitness of 1; the second fitness of s (typically 0.99); the third fitness of s2and so on.

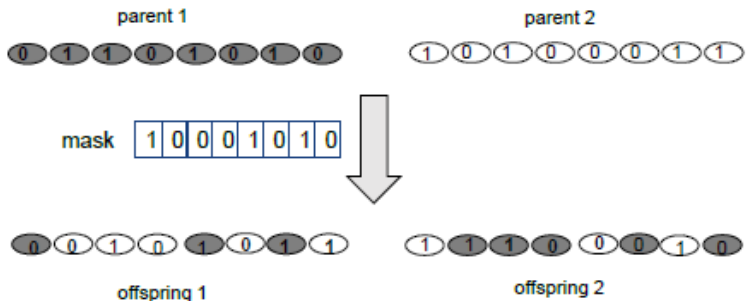
- Crossover
  - Combines the features of two parent structures (new combinations of genes are generated from previous ones) to form two similar offspring →conservesgenetic information.
  - crossover operator
    - Segments are cut-and-spliced between the chromosomes (strings)
      - Segments of two parent chromosomes are swapped producing two offsprings
      - Crossover site chosen randomly
    - Single-point crossover



    - Multi-point crossover(k-point crossover)

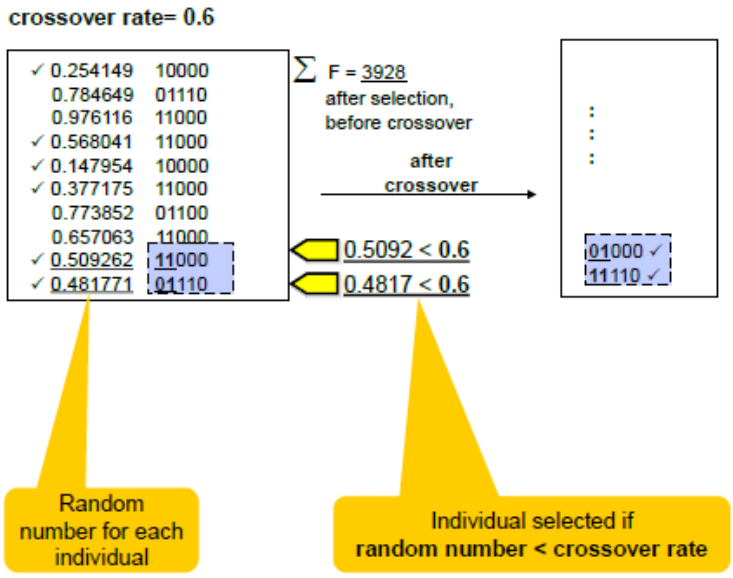


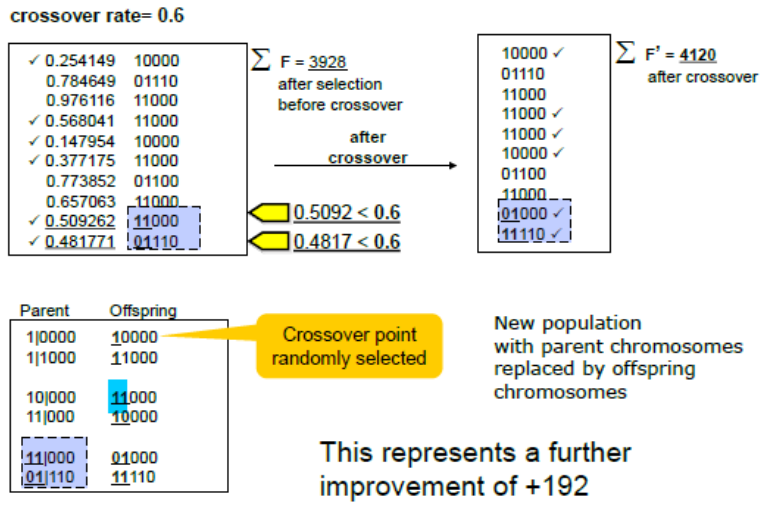
      - Traditionally, the number of crossover points has been fixed at a very low constant value of 2.
    - Uniform crossover



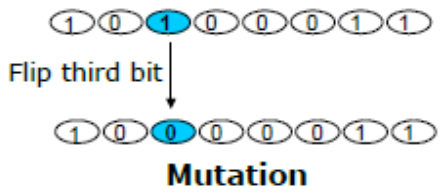
- Genes are copied from the first parent or from the second parent based on a randomly created mask

- crossover rate
  - The probability of crossover (crossover rate) pcgives the expected number pc×population size of chromosomes to undergo the crossover operation
    - Generate a random number r from the range [0..1]
    - If r<pc, select given chromosome for crossover
    - For each pair of coupled chromosomes we generate a random integer number (the position of the crossover point) from the range [1..m-1] where m is the number of bits in a chromosome
      - The parent chromosomes are replaced by the offspring chromosomes.
- example

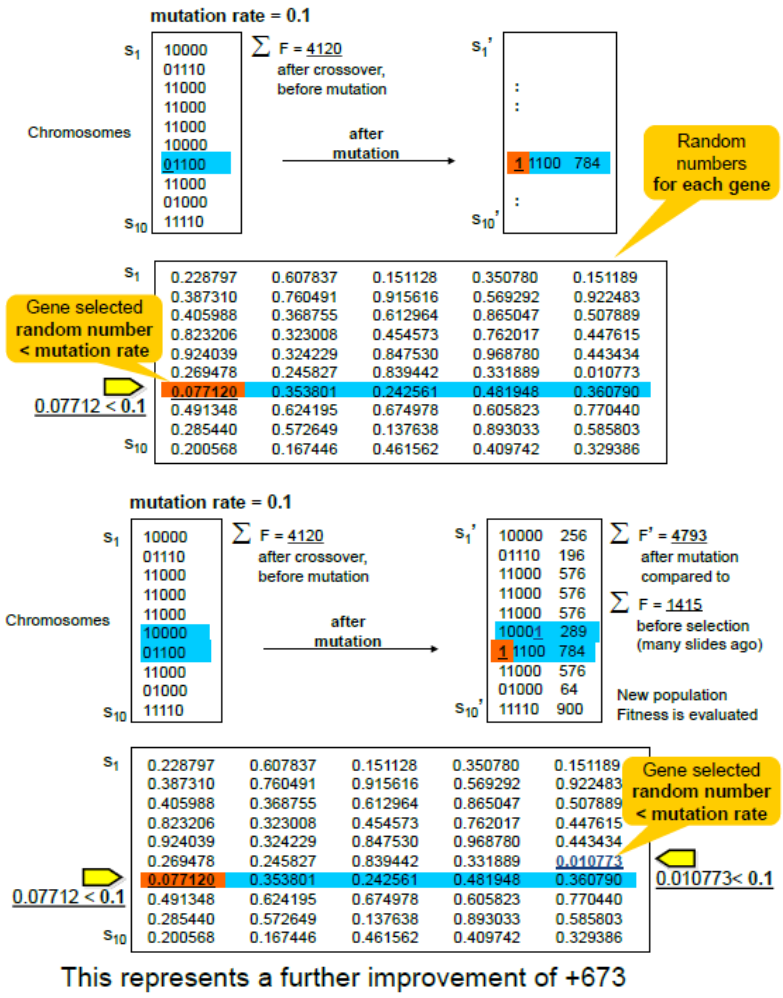




- Mutation
  - Alters one or more components of a selected structure randomly (a direct analogy from nature and provides the way to introduce new information into the population) →population diversity
  - mutation operator



    - Replace the value at some randomly-chosen position by a new arbitrary value
      - The role of mutation is to maintain genetic diversity
      - A range of mutation operators have been proposed, ranging from completely random alterations to more heuristically motivated local search operators
      - Bear in mind that there are other kinds of mutation operatorsbesides bit-flipping
  - mutation rate
    - Applying mutation operator to the individuals in the new population
      - The probability of mutation (mutation rate) pmgives the expected number of mutated bits pm×population size ×number of bits in a chromosome
      - Every bit (in all chromosomes in the whole population) has an equal chance to undergo mutation
        - Generate a random number r from the range [0,1]
        - If r < pm→mutate the bit
  - example



- The resulting offspring are then evaluated and inserted back into the population, replacing older members.
  - Specific decisions about how many members are replaced during each iteration, and how members are selected for replacement, define a range of alternative implementations
  - Generational GA
    - GAs that replace the entire population
  - Steady-state GA
    - GAs that replace only a small fraction of chromosomes. Typically new chromosomes replace the worst chromosomes
- The rest of evolution is just cyclic repetition of these steps.
- The total fitness of the new population is higher than total fitness of the previous generation –or so, one would hope □
- stopping criterion must be specified:
  - After a fixed number of generations
  - After a chromosome with a certain high fitness value is located
  - After all the chromosomes in the population have attained a certain degree of homogeneity
- Parameters for GAs

- GA's performance in problem solving will depend on
  - The method for encoding candidate solutions
  - The method for selecting the individuals in the population to reproduce for the next generation
  - Choices of genetic operators
  - The parameter settings
    - Population size
    - Probability of crossover (crossover rate) gives the probability that a pair of chromosomes will be combined
    - Probability of mutation (mutation rate) gives the probability that a bit will be flipped.
- Trade-off Considerations on Control Parameters
  - Increasing the crossover probability increases recombination of schemata, but it also increases the disruption of good chromosomes.
  - Increasing the mutation probability tends to transform the genetic search into a random search, but it also helps re-introduce lost genetic material.
  - Increasing the population size increases its diversity and reduces the probability that the GA will prematurely converge to a local optimum, but it also increases the time required for the population to converge to the optimal regions in the search space.

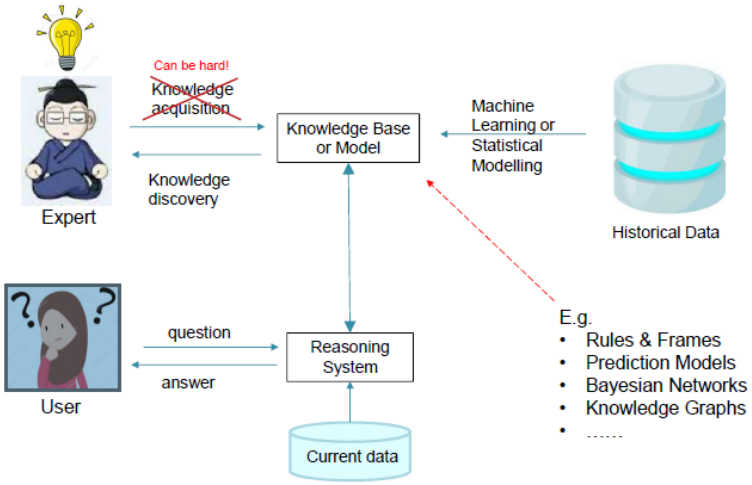
- When & How to Use GAs?
  - When would you want to apply GA on a problem?
    - No idea how to reasonably solve the problem
    - You cannot enumerate all possible solutions
    - You know how to evaluate how good or bad a solution is
  - What do we need to use Genetic Algorithms?
    - A goodness measure→this is eventually translated into the fitness function
    - A representationof the solution →the more natural, the better
    - A problem modelwhich defines the behaviour
- Application
  - GAs can be applied to a wide range of optimization and learning problems
    - Routing and scheduling, machine vision, engineering design optimization, gas pipeline control systems, machine learning
    - Hundreds of applications have now been discovered and a variety of commercial software tools have been introduced.
  - Numerical function optimisation

            - GA has been shown to outperform conventional optimisation techniques on difficult, discontinuous, multimodal, noisy functions.
        - Combinatorial optimisation
            - Resource allocation problems
            - Classical problem of travelling salesperson / bin-packing / job shop scheduling
            - For example:
                - Assign jobs to machines over time such that the machines' idle time and the jobs' throughput time are minimized
                - Timetabling of examinations or classes in the universities, colleges
        - Financial forecasting
            - Searching for a set of rules or equations that will predict the ups and downs of a financial market, such as that for foreign currency and stocks.
            - Hypothesis: linear combination of technical indicators can be used to forecast periods of rising price movement in the stock market.
            - The contribution of each indicator in a forecasting model is indicated by a weighting coefficient.
            - The coefficients are derived by a genetic algorithm.
            - The resulting models are evaluated against historical price movement of the stocks.
        - Design optimisation
            - network routing
                - To maximise throughput for given bandwidth (first objective) while minimizing cost (second objective).
                - Use of network simulation tools to determine bandwidth utilisation and throughput.
            - Satellite orbit selection
                - Avoid collision and reduce blackout window
        - Machine learning
            - Used to evolve aspects of particular machine learning systems, such as weights for neural networks, rules for learning symbolic production systems, and sensors of robots.
            - Classifier systems which evolve if-then rules for a specific problem domain
                - For example: rule induction for financial decision making
- Recommender Systems
    - Basic concept

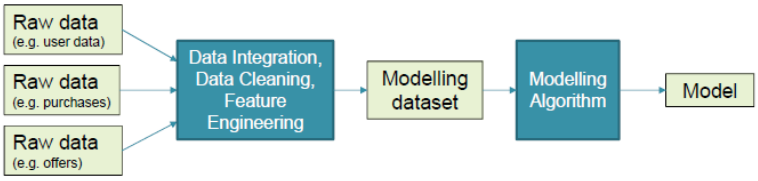
- Data Sources and Sizes

Data Type	Size	Examples	Possible Applications
User-level data	Moderate	Account data, Registration data, Demographics, Interests, …	Customer Relationship Management (CRM), Customer Retention Sale / Marketing Personalised Offers Recommendations Prediction Diagnosis Fraud detection
Events (discrete)	Huge	purchase transactions financial transactions healthcare visits bus trips web page views clicks, likes, downloads…	
Sensor Data (multi-media and often continuous)	Massive	Images, video, speech, sound, IOT…	Facial recognition Surveillance Inspection Chatbots Robots / Vehicles Smart city, Smart home

- Reasoning From Data versus From Expert Knowledge



- Creating a Modelling Dataset



Feature Engineering typically refers to the process of transforming and combining raw variables to create new features with better predictive power, e.g:

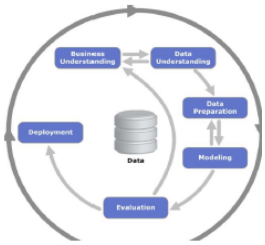
Raw variables
Unit price
Number sold
→ Total cost

- Raw data is usually not suitable for direct input into a model
- Data Preparation and Data Transformation are equally as important as the modelling algorithm (arguably more important). This process can be very time consuming
- The (Complete) Model Building Process

- Also called the Data Mining Process - many methodologies exist, but all similar. Iteration is crucial – keep refining the data and testing the model until its good enough for deployment

https://elearningindustry.com/machine-learning-process-and-scenarios

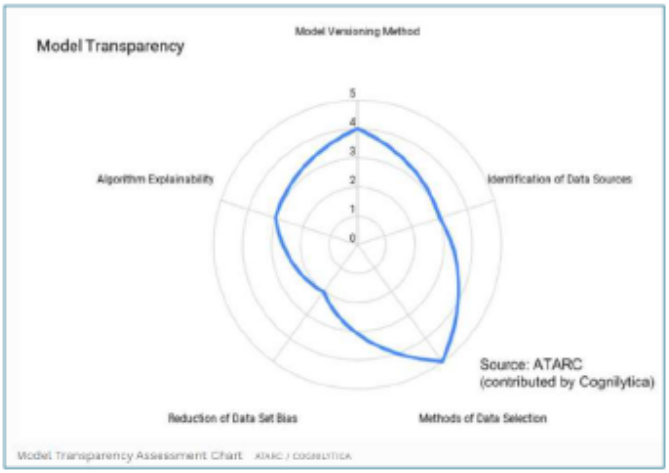
CRISP methodology



Many emphasise Business and Data Understanding as crucial for project success – obtaining good model accuracy against a test set is not enough - the model must fit the situation in which it is to be deployed (a good solution to the wrong question is of no use)

Note : the model does not need to be executable, it can be a representation of knowledge discovered without necessarily modelling a specific outcome (knowledge discovery)

- Knowledge Discovery & Transparency
  - Knowledge Discovery in Databases (KDD) is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns or relationships within a dataset in order to make important decisions (Fayyad et al, 1996). The term KDD is often used interchangeably with Data Mining
  - Transparency and Trust - for some AI applications there is no need for the user to understand how a decision is made as long as the result/decision is correct. But in many applications we need transparency in the decision making process, we need to understand:



Different Aspects of Model Transparency

    - Why did you recommend that drug?
    - Why did you show me that ad?
    - How do I know the system is trustworthy, does not have a hidden bias?
  - Explainable AI
    - One way to gain explainability and trust in AI systems is to use machine learning algorithms that are inherently explainable:
      - Decision tree and Decision rule learning
      - Association rule learning
      - Similarity-based Reasoning (Nearest Neighbour algorithms)

- Linear Regression
- Logistic Regression
- Naïve Bayes
- Clustering

- But increasingly, many AI models are black boxes. Explainable AI is an emerging field in machine learning that aims to address how black box decisions of AI systems are made
- Association Rules

Association Rules	Support (%)	Confidence (%)	Lift
Soap → Perfume	30	75	1.87
Perfume → Soap	30	75	1.87
Bread → Jam	40	80	1.6
Jam → Bread	40	80	1.6
Chocolate → Bread	30	75	1.5
Beer → Snacks	30	100	2

  - Data that represents things that happen at the same time or in the same context can be mined to find associations
  - Finding association between items in a shopping baskets is a common application
  - (aka Market Basket Analysis).
  - The learned rules can be used to guide manual decisions (e.g. how to layout the supermarket shelves) – or can be used in a rule-based system (e.g. to make recommendations)
  - Sort by Support and Confidence to see the most important rules
- Similarity-Based Reasoning

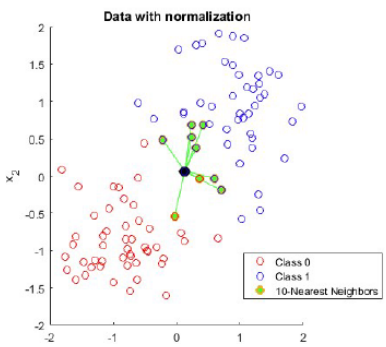
probably also similar in some further respect

Nearest Neighbour Systems

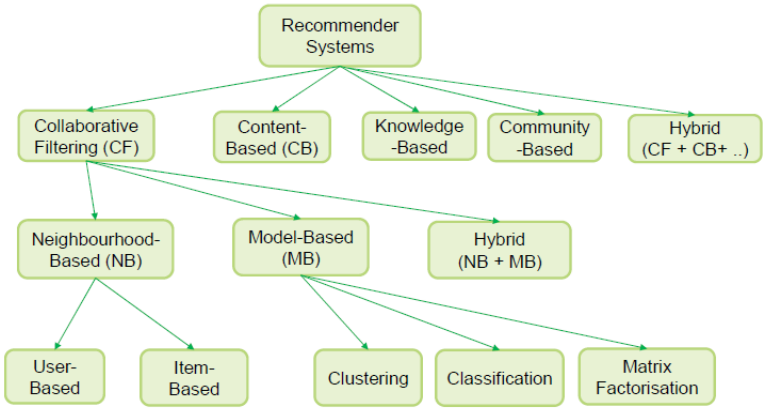
Represent situations (cases) as vectors.

To reason about a new case :
- Find the K most similar neighbours to the new case

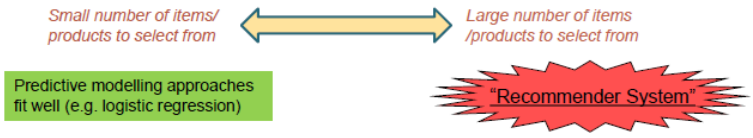
Make a decision by either:
- taking the majority vote *(for classification)*
- averaging over the K neighbours *(for numerical prediction)*



- Baesic Approaches



- Scoping Recommender Systems
  - The phrase Recommender System is very broad – it could be used to describe a very wide range of applications, including common predictive modelling applications. (Just replace the word predicting with the word recommending!)
  - One distinguishing feature is the number of possible recommendations that can bemade:



- Market Basket Analysis
  - Much can be done with simple analysis of things that happen together
    - Items bought together (in same shopping basket)
    - Items viewed in the same browsing session
    - Items bought by same users but on different days
  - Requires a list of transactions
- Method
  - Cooccurrence Counting

	Pizza	Milk	Cola	Chips	P/nuts
Pizza	2	1	2	0	0
Milk	1	3	1	1	1
Cola	2	1	4	1	2
Chips	0	1	1	2	1
P/nuts	0	1	2	1	3

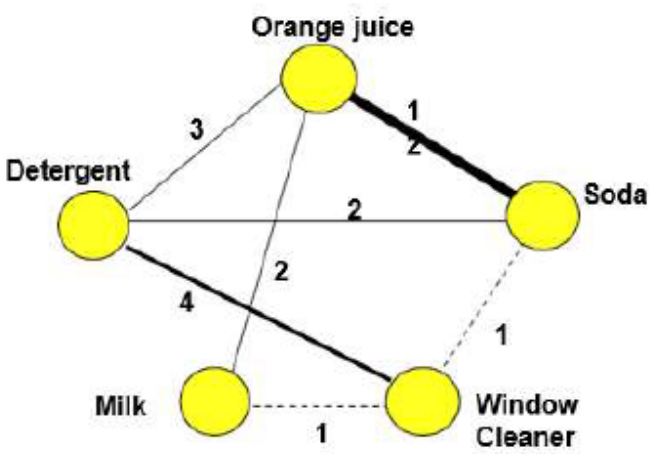
Pizza buyers (2) always also buy cola (2)

But Cola buyers (4) do not always buy pizza (2)

Milk sells well with everything

Peanut buyers (3) often buy cola (2)

    - A simple approach to MBA is to cross-tabulate into a table to show how often each possible pair of products were bought together:
    - The table (the co-occurrence matrix) is symmetrical since the items in each basket have no temporal ordering. The diagonal shows the total number of times the item was bought.
  - Link Analysis



- Cooccurrences can be visualized using link analysis
- Nodes represent items, thickness of joining line indicates number of times they occurred together

- Association Rule
  - Association Rule Learning
    - Cooccurrence counting is not viable for large datasets and large item-sets
    - Algorithms such as Apriori (Agrawal et al, '93) use heuristics to reduce the combinatorial size of the search space
      - If an item-set is frequent, then all of its subsets must also be frequent
      - The user specifies minimum rule support and rule confidence
    - The found associations are expressed as rules, e.g.

      If buy pizza then also buy cola

      LHS (left-hand side) RHS (right-hand side)
    - In general, association rules can have multiple items in the LHS or RHSNote: rules indicate co-occurrence, not causality or a sequence over time

      If Coffee and Milk then Sugar
      If BBQ charcoal then Sausages and Steak

      Note: rules indicate co-occurrence, not causality or a sequence over time
  - Association Rule Metrics
    - Rule Support
      - The proportion of transactions that contain the item set (LHS + RHS items)
    - Rule Confidence
      - The proportion of rule firings that are correct predictions
      - Support for combination (LHS & RHS) / Support for condition (LHS)
    - example

$$= \frac{\#\text{transactions containing all items in the rule (LHS and RHS)}}{\#\text{transactions containing all items in the rule condition (LHS)}}$$

e.g. consider page-views in a web browsing session as the baskets

session1:	home, news, sport
session2:	finance, news
session3:	fashion, home
session4:	news, finance, home
session5:	sport, home, finance
session6:	fashion, home, news
session7:	home, finance, news, sport

home → news
Support = 4/7
Confidence = 4/6

news → home
Support = 4/7
Confidence = 4/5

Note: news→home does not have same *confidence* as home→news

To get intuition consider: whisky->coke (often), but coke->whisky (less)

$$\text{Rule lift} = \frac{\text{Support (LHS \& RHS)}}{\text{Support (LHS)} * \text{Support (RHS)}} =$$

the ratio of the observed support to that expected if LHS and RHS were independent

If lift = 1 then this implies that the probability of occurrence of X and Y are independent (no association)
If the lift is > 1, then LHS and RHS are dependent on each other (one makes the other more likely)
If the lift is < 1, then one (LHS or RHS) has a negative effect on presence of other

session1:	home, news, sport
session2:	finance, news
session3:	fashion, home
session4:	news, finance, home
session5:	sport, home, finance
session6:	fashion, home, news
session7:	home, finance, news, sport

home → news or news → home

Lift = (4/7) / ((6/7)*(5/7))
= 0.57 / 0.61 = 0.93

- Association Mining Applications
  - Items purchased on a credit card (e.g. rental cars, hotel rooms) give insight into the next product the customer may buy
  - Optional services bought by telecom customers (call waiting, forwarding, auto-roam etc) show how best to bundle these services
  - Banking services used by retail customers (investment services, car loans, home loans, money market accounts etc) show possible cross-sells
  - Unusual combinations of insurance claims may indicate fraud
  - May find associations between certain combinations of medical treatments and complications in medical patients
- Issues
  - Can generate a huge number of rules, often trivial and with repetition:
    - If coffee and milk then sugar
    - If milk and sugar then coffee
    - If sugar and coffee then milk
  - Define minimum support and minimum confidence for rule pruning/filtering to get “strong” rules
  - Analyst must make decisions regarding validity & importance of rules to be accepted (subjective)
- example

- Association Rule Examples

on-air & business & sports & bbs	-->	frontpage	86.22%
news & tech & misc & bbs	-->	frontpage	86.18%
on-air & misc & business & sports	-->	frontpage	86.16%
tech & misc & travel	-->	on-air	86.09%
tech & living & business & sports	-->	frontpage	86.08%
news & living & sports & bbs	-->	frontpage	85.99%
misc & business & sports	-->	frontpage	85.79%

- Sequence Rules Examples

on-air → news	1.51%
news → frontpage → news	1.49%
local → news	1.46%
frontpage → frontpage → business	1.35%
news → sports	1.33%
news → bbs	1.23%
health → local	1.16%

Are these rules likely to be useful?

*(*Frequent Pattern Mining in Web Log Data, Ivancsy, Vajk, 2006. Using their own association and sequence finding algorithms)*

Data from the Pagview log of MSNBC.com (a news site) on 28 Sep'99

- Raw data ~ Datetime, URL, cookieID, IPaddress, UserAgent(browser) etc...

Preprocessing

- **Page view categorization** – the pages viewed (URLs) were first converted into topic categories
- **Grouping by unique user** - each data row shows the sequence of pageview categories for one user on that day

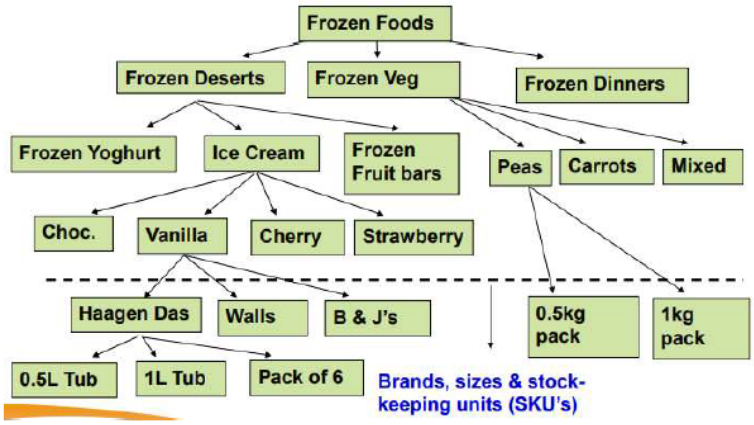
Codes for the msnbc.com page categories

category	code	category	code	category	code
frontpage	1	misc	7	summary	13
news	2	weather	8	bbs	14
tech	3	health	9	travel	15
local	4	living	10	msn-news	16
opinion	5	business	11	msn-sport	17
On-air	6	sports	12		

Number of users: 989,818
Average number of visits per user: 5.7
No. of URLs per category: 10 to 5000

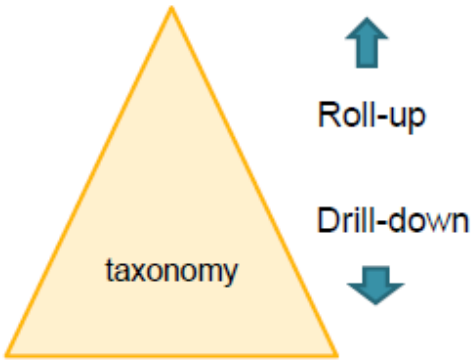
```
% Sequences:
6
1 1
6
6 7 7 7 6 6 8 8 8 8
6 9 4 4 4 10 3 10 5 10 4 4 4
1 1 1 11 1 1 1
12 12
```

- Detail



- Note that the taxonomy is often not unique

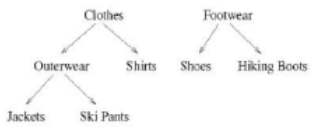
- Recommending with Taxonomies



- No need to have all items at the same level
- Level should reflect importance
  - Is brand more important than flavour? Is pack size important?
- Items should occur in approximately the same frequency, otherwise rules are dominated by the common items
  - E.g. rice is bought very frequently, durian cakes very infrequently, hence the rule: if durian cake then rice is likely to be found, but is not useful
- Hence …..
  - Roll-up rare items to higher levels so they are more frequent
    - e.g. salami pizza, tuna pizza, veg.pizza etc -> pizza
  - Break-down (drill-down into) very frequent items
    - e.g. pizza -> salami pizza, tuna pizza, veg.pizza etc

- Generalised Association Mining

Transaction	Items Bought
100	Shirt
200	Jacket, Hiking Boots
300	Ski Pants, Hiking Boots
400	Shoes
500	Shoes
600	Jacket



Frequent Itemsets

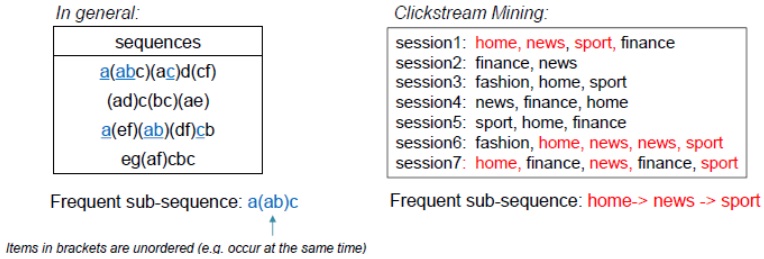
Itemset	Support
{ Jacket }	2
{ Outerwear }	3
{ Clothes }	4
{ Shoes }	2
{ Hiking Boots }	2
{ Footwear }	4
{ Outerwear, Hiking Boots }	2
{ Clothes, Hiking Boots }	2
{ Outerwear, Footwear }	2
{ Clothes, Footwear }	2

Rules

Rule	Support	Conf.
Outerwear ⇒ Hiking Boots	33%	66.6%
Outerwear ⇒ Footwear	33%	66.6%
Hiking Boots ⇒ Outerwear	33%	100%
Hiking Boots ⇒ Clothes	33%	100%

  - Given a large database of transactions, where each transaction consists of a
  - set of items, and a taxonomy (is-a hierarchy) on the items. Find associations
  - between items at any level of the taxonomy.
- Virtual Items
  - Expand the scope of association mining from items / products to any categorical variable of interest e.g.
    - Type of promotion
    - Store location (urban, suburban, rural)
    - Season or month, time of day (am, lunch, pm, evening)
    - Payment mode (cash, cheque, credit card)
    - Gender of customer (male, female)
  - Can add any relevant information as an 'item' into the basket E.g. To find associations between purchased items and new customers
    - Enter transaction as: {sweater, jacket, new customer}
    - Possible Association Rule: If new customer and jacket then sweater
  - Can include numerical concepts into the baskets by first binning them

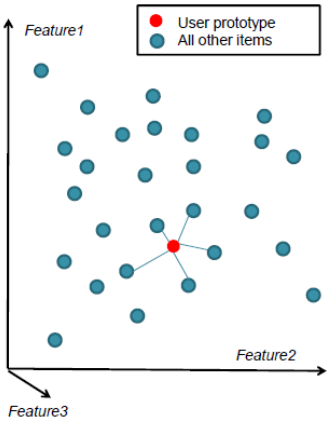
        - E.g. Age -> young-age, middle-age, senior-age
        - Possible Association Rule: If outdoor jacket and middle-age then hiking boots
    - Sequence Mining



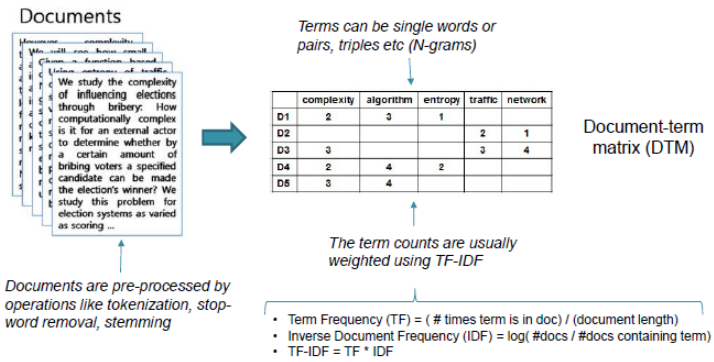
        - Similar to association mining, but where the order in which items are placed into the basket is important
            - E.g. A sequence of events over time, DNA sequences, word sequences
            - Many algorithms based on extended apriori , e.g. SPADE
        - Useful for session-based recommender systems
            - Many web-site browsing sessions are made by anonymous users, the only ID is the cookie ID. Hence storing historical data for the user is usually not possible. Making recommendations must use only the browsing behaviour in the current session
- Personalised Recommendation Approaches
    - Content-Based Recommendation
        - Recommend based on what you have bought or viewed in the past
        - Match users directly to products & content
        - Commonly used for document recommendation: webpages, news articles, blogs etc.
        - Content-Based Recommenders
            - Content Analyser ~ extracts structured product descriptions from a diversity of products
                - E.g. for movie: director, length, year, genre etc.
                - Data may already be available – e.g. structured data provided by the manufacturer
                - OR extract using text mining -parse product descriptions (e.g. product webpages) and extract features using keyword extraction
            - Profile Learner ~ collects data about the user preferences and generalises into a user profile
            - Filtering ~ matches the user profile against the product descriptions. Returns the products with the closest
        - Content-Based Filtering Example

	Length	studio	Director	Lead Actor1	Genre	Date
movie1	90	Disney	Spielberg	Tom Hanks	Sci-fi	2014
movie2	110	MGM	Petersen	Brad Pitt	action	2012
movie3	120	Disney	Nolan	C. Bale	action	2008
.....						
movieN						

- Similarity-Based Reasoning



  - Each item is represented by a set of features (a feature vector)
    - The user is represented by the same feature set, often obtained by:
    - The user selects preferences for the various features using pull-down menus
    - Computing an “average” vector from items already bought/liked by the user. This will represent a “typical” item for the user (often called a “prototype vector)
  - Then use a distance (or similarity) measure to find the nearest items to the user
  - The nearest items are then recommended –ranked by their distance
  - If the retailer is selling an diverse product range there is probably no single “typical” item for a user
  - Instead we can recommend items that are close to the items that the user has already bought (or viewed)
  - Transparency -> Can explain a recommendation by showing the factors that trigger the recommendation(s)
- Text-Based Features

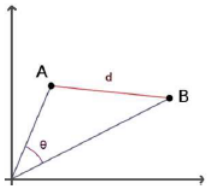


  - Instead of explicitly defining product features we can extract text-based features from product documentation, product reviews, product webpages etc.

        - E.g. The Bag-of-Words approach (BOW) represents a document by features that count the number of times each word (or term) occurs in the document
    - Similarity & Distance Measures

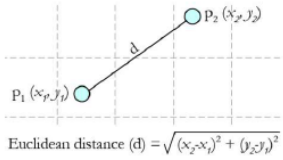
$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

**Content-based recommender systems have their roots in Information Retrieval (IR). The goal of IR is to identify relevant documents in response to a user query*



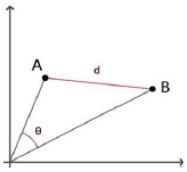
When the documents are similar the angle between their vectors is small and the cosine similarity approaches +1

For general classification problems, the distance metric used in k-NN systems is commonly the Euclidean distance



Can convert to "similarity" using (for example):
similarity = 1/(1 + distance)

If the vectors are documents then Cosine Similarity is often better because it ignores the magnitude of the vectors (only considers angle)



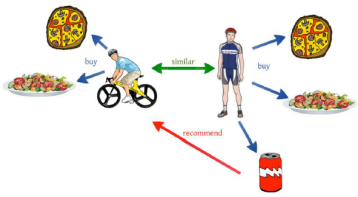
E.g. if one doc has word "disease" 10 times and another has it 100 times then using cosine they will still be similar but they will be different using Euclidean distance

E.g. Two docs with similar word profiles

	w1	w2	w3	w4	w5	w6
Item1	0	10	20	100	200	400
Item2	0	100	200	1000	2000	4000
				Eucl Dst	Eucl Sim	Cosine
				4129.2	0.0002	1.0000

    - Content-Based Recommender Systems typically use the cosine distance to measure the similarity between user and items
    - In information retrieval systems* the similarity between two documents (A and B) is given by:
- Collaborative Filtering

Match the user to similar users and recommend what they like



    - Match users to people with similar tastes
        - recommend what they like
    - Commonly used in e-retail
    - Avoids the issue of users only being recommended more of what they already like(allows serendipity)
    - Measuring User Similarity

Entries are numbers, e.g. 1->5, where 1 means do not like, 5 means like a lot

Ratings can be explicit (directly given) or implicit (inferred from page-views, purchases etc.)

- How do we find similar users? What user data can we use?
  - Most common data used is the user-ratings matrix, a matrix of the user likes or interests in the various products/items on offer
- How to compute distances/similarities between entries in the ratings matrix?

Common Metrics:
- Euclidean Distance
- Cosine Similarity
- Pearson Correlation

  - Sparsity (many missing values) is a problem
- Users may use different ratings to quantify the same level of appreciation for an item. This can create bias. E.g. one user may rate a movie as 5 and the other 4, yet they both like it equally well
- Pearson Correlation Coefficient is similar to cosine similarity but eliminates this bias by subtracting each users mean rating from their individual ratings. Value ranges from -1 to +1

$$\text{Sim}(\mathbf{u},\mathbf{v}) = \frac{\sum_{i \in C} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in C} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in C} (r_{v,i} - \bar{r}_v)^2}}$$

$r_{u,i}$ = rating of user u on item i
$r_{v,i}$ = rating of user v on item i
$\bar{r}_u$, $\bar{r}_v$ = mean ratings of user's u, v
C is the set of all co-rated items

User	movie1	movie2	movie3	movie4	movie5	movie6	Mean?
target	1	2			3	2	2
1	2	5	4		5	1	3.4
2	2	3	5	4		2	3.2
3			5	3		4	4

```
> users
       m1 m2 m3 m4 m5 m6
target  1  2 NA NA  3  2
u1      2  5  4 NA  5  1
u2      2  3  5  4 NA  2
u3     NA NA  5  3 NA  4
```

```
> cor(t(users), use="pairwise.complete.obs")
          target        u1        u2        u3
target 1.0000000 0.5940885 0.5000000        NA
u1     0.5940885 1.0000000 0.6454972 1.0000000
u2     0.5000000 0.6454972 1.0000000 0.3273268
u3            NA 1.0000000 0.3273268 1.0000000
```

- Making a Recommendation

$$\text{Predicted rating } (\mathbf{u},i) = \overline{r}_u + \frac{\sum_{v \in V} \text{sim}(\mathbf{u},\mathbf{v}) * (r_{v,i} - \overline{r}_v)}{\sum_{v \in V} |\text{sim}(\mathbf{u},\mathbf{v})|}$$

V is the set of other users

User	similarity	m1	m2	m3	m4	m5	m6	Mean
target	1	-1	0	?	?	1	0	2
user1	0.59	-1.4	1.6	0.6		1.6	-2.4	3.4
user2	0.5	-1.2	-0.2	1.8	0.8		-1.2	3.2
user3	NA			1	-1		0	4
weighted average				$\frac{0.59*0.6 + 0.5*1.8}{0.59+0.5} = 1.15$	$\frac{0.5*0.8}{0.5} = 0.8$			
predicted rating				2 + 1.15 = 3.15	2 + 0.8 = 2.8			

- Once we have the similarities between the target user and the other users then we compute a recommendation based on the preferences of other users. One method is to sum the preferences of these other users weighted by their similarity to the target (weighted average)

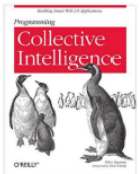
- example

The raw ratings….

User	Lady In Water	Snakes On A Plane	Just My Luck	Superman	You Me And Dupree	The Night Listener
Rose	2.5	3.5	3	3.5	2.5	3
Seymour	3	3.5	1.5	5	3.5	3
Philips	2.5	3		3.5		4
Puig		3.5	3	4	2.5	4.5
LaSalle	3	4	2	3	2	3
Matthews	3	4		5	3.5	3
Toby		4.5		4	1	

The goal is to make movie recommendations for Toby.

Do this by predicting his likely rating for his unseen movies:
***Lady** in Water, Just My **Luck**, The **Night** Listener*



**Example from "Programming Competitive Intelligence", O'Reilly*

(1) Calculate Toby's Similarity to Others

User	Lady In Water	Snakes On A Plane	Just My Luck	Superman	You Me And Dupree	The Night Listener
Rose	2.5	3.5	3	3.5	2.5	3
Seymour	3	3.5	1.5	5	3.5	3
Philips	2.5	3		3.5		4
Puig		3.5	3	4	2.5	4.5
LaSalle	3	4	2	3	2	3
Matthews	3	4		5	3.5	3
Toby		4.5		4	1	

The Pearson implementation in the book and also in R (the cor() fn) works with sparse data by only considering the "pairwise complete" items.

Hence:

- Mean(Rose) = (3.5+3.5+2.5)/3 = 3.16, Mean(Toby) = (4.5+4+1)/3 = 3.16
- Sim(Toby, Rose) = $\frac{(3.5-3.16)*(4.5-3.16) + (3.5-3.16)*(4-3.16) + (2.5-3.16)*(1-3.16)}{\text{sqrt}(((3.5-3.16)^2+(3.5-3.16)^2+(2.5-3.16)^2) * ((4.5-3.16)^2+(4-3.16)^2+(1-3.16)^2))} = 0.99$

(2) Compute Weighted Avg of Neighbour Ratings

Critic	Similarity	Night	S.xNight	Lady	S.xLady	Luck	S.xLuck
Rose	0.99	3.0	2.97	2.5	2.48	3.0	2.97
Seymour	0.38	3.0	1.14	3.0	1.14	1.5	0.57
Puig	0.89	4.5	4.02			3.0	2.68
LaSalle	0.92	3.0	2.77	3.0	2.77	2.0	1.85
Matthews	0.66	3.0	1.99	3.0	1.99		
Total			12.89		8.38		8.07
Sim. Sum			3.84		2.95		3.18
Total/Sim. Sum			3.35		2.83		2.53

The similarity of the other users to Toby (this was calculated using the Pearson Coefficient)

All neighbours with similarity > 0 are includes in the weighted average (avoid having to select a value for K)

The weighted average of the other users ratings for each unseen movie.
Note that, for simplicity, only the raw ratings are used here

- User-Based CF: Scalability Issues
  - User-Based Collaborative Filtering does not scale well
    - User likes/interests may change often hence similarities between users are best computed at the time of the recommendation – this becomes computationally prohibitive for large numbers of users
  - Computational load
    - Worse case ~ O(M*N) where M = #users, N = #items
    - In practice ~ O(M + N) since most users have rated/purchased few items
  - Possible solutions
    - Dimensionality Reduction
    - Item-Item Collaborative Filtering
- Item-Based Collaborative Filtering

item	user1	user2	user3	user4	....	userN
movie1	2	5	4			
movie2		3			1	
.....						

  - Transpose the user-rating matrix, compute distances between items not users
  - Faster than User-Based – there are usually far fewer items than users AND the similarities between items change much less frequently than between users hence all item-item distances can be pre-computed
  - A well known application of this method is Amazon's recommendation engine
  - Item-Item Similarity Matrix

	item1	item2	item3	item4	item5	item6	…
item1	1	0.90	0.61	0.13	0.68	0.75	…
item2		1	0.58	0.78	0.12	0.29	…
item3			1	0.55	0.32	0.69	…
item4				1	0.11	0.98	…
item5					1	0.75	…
item6						1	…
….							…

The matrix is symmetrical. Similarity is measured from 0 to 1 or -1 to 1, diagonals hence take the value 1

Typically precomputed, the frequency of updating depends on type of item (e.g. fast or slow seller, speed of new releases) – weekly or monthly updates may suffice.

- Item-Based Collaborative Filtering
  - Compute the distance between items ***i*** and ***j*** using the adjusted cosine similarity*:

$$\text{sim}(i, j) = \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_u)(r_{u,j} - \bar{r}_u)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{u \in U} (r_{u,j} - \bar{r}_u)^2}}$$

$r_{u,i}$ = rating of user u on item i
$r_{u,j}$ = rating of user u on item j
$\bar{r}_u$ = mean rating of user u
U = set of all users

  - Predict user ***u***'s rating for an item ***i*** using:

$$\text{Predicted rating } (u,i) = \frac{\sum_{j \in J} \text{sim}(i,j) * r_{u,j}}{\sum_{j \in J} |\text{sim}(i,j)|}$$

j is the set of K similar items to ***i*** (among items that u has rated)

** Other similarity measures can be used (e.g. Euclidean Similarity)*

- Recommendations

↓ Movies seen by Toby → Movies not seen by Toby

Movie	Rating	Night	R.xNight	Lady	R.xLady	Luck	R.xLuck
Snakes	4.5	0.182	0.818	0.222	0.999	0.105	0.474
Superman	4.0	0.103	0.412	0.091	0.363	0.065	0.258
Dupree	1.0	0.148	0.148	0.4	0.4	0.182	0.182
Total		0.433	1.378	0.713	1.764	0.352	0.914
Normalized			3.183		2.473		2.598

(= 1.378/0.433)

Toby's ratings for the movies he has seen

The similarities between ***Night Listener*** and the movies seen by Toby – these are derived from the pre-computed item-item similarity matrix (using Euclidean similarity)

Toby's estimated rating for ***Night Listener*** is obtained by computing his average rating for all of his seen movies – weighted by their distance to ***Night Listener***

- Which is Best: User-Based or Item-Based?
  - If #users >> #items (e.g. Amazon) then Item-Item may be better
  - If #users << #items (e.g. recommending web content) then User-User may be better

- Community-Based/Social
  - Who are your friends and what do they like?
  - Users trust their friends opinions
- Knowledge-Based
  - Often more appropriate for recommending products where many user specific requirements and constraints (often conflicting) must be taken into account

- E.g. cars, houses, computers, financial services

- Example Methods:
  - Constraint-based Reasoning /Search
  - Multi-Criteria Decision Making (MCDM)

- Evaluating Recommender Systems
  - Evaluating the accuracy of rating predictions
    - For recommender systems that predict ratings
    - Hold-out testing:
      - Hide, in turn, each rating* made by each test user and then predict the rating for the hidden item and compute:
        error = actual rating – predicted rating

**or we can hide (hold-out) more than one rating for each test*

User	movie1	movie2	movie3	movie4	movie5	movie6	etc....
Test user	2	5	4		3	1	

For each movie rated by the test user:

1. Set the movie rating to blank (NA) – but keep a copy
2. Make a prediction for that movie using the training data
3. Compare the prediction with actual rating:
   error = abs(predicted rating – actual rating)
4. Increment a count of the number of tests
5. Restore the blank movie rating

Do this for all test users

At the end compute the MAE (Mean Absolute Error)

$$MAE = \frac{\Sigma\ abs(errors)}{\#\ tests}$$

  - Evaluating the accuracy of predicting "like"
    - Assume the user likes an item if they rate it highly , e.g. rating >= 4 (on scale of 1 to 5)
    - This enables us to compute a confusion matrix:

	Recommendation	
Actual	Won't Like	Will Like
Rated Poor	TN	FP
Rated High	FN	TP

TN = true negative, FP = false positive
FN = false negative, TP = true positive

$$Precision = \frac{TP}{TP+FP}$$ *% of predicted likes that were good*

$$Recall = \frac{TP}{TP+FN}$$ *% of likes that we predicted (aka sensitivity)*

- Procedure

Same as before - for each test user and each test item:
- increment TP when predicted rating is >=T AND actual rating is >= T*
- increment FP when predicted rating is >= T AND actual rating is < T
- increment TN when predicted rating is < T AND actual rating is < T
- increment FN when predicted rating is < T AND actual rating is >= T

BUT this only evaluates the accuracy when predicting "like". It is not the ***overall effectiveness*** of the set of recommendations made

**T is the "like" threshold, e.g. 4*

  - Precision at K (P@K)
    - P@K is the proportion of recommended items in the top-K set that are relevant
    - An item is relevant if the user is known to like it (e.g. actual rating >=4)

E.g. for K = 10:

As ordered by the recommender system

Item ID's for the top 10 recommended items:

Test user1	29	3	102	56	38	120	12	18	83	33	p@k = 3/10 = 0.3

Items that the user has liked (e.g. rated >= T)

This user has only 8 unrated items:

Test user2	56	18	12	43	5	19	77	39	-	-	p@k = 4/8 = 0.5

**When computing p@k for a user from a test set, we remove from the recommended item list all items not in the test set for that user (these are the items that we do not know the actual rating for). For users with few ratings this may cause issues.*

https://medium.com/@m_n_malaeb/recall-and-precision-at-k-for-recommender-systems-618483226c54

  - Mean Average Precision at K (MAP@K)

P@K

> How many relevant items are present in the top-k recommendations of your system

For example, to calculate P@3: *take the top 3 recommendations for a given user and check how many of them are good ones. That number divided by 3 gives you the P@3*

AP@K

> The mean of P@i for *i=1, ..., K*

For example, to calculate AP@3: *sum P@1, P@2 and P@3 and divide that value by 3*

AP@K is typically calculated for one user.

MAP@K

> The mean of the AP@K for all the users.

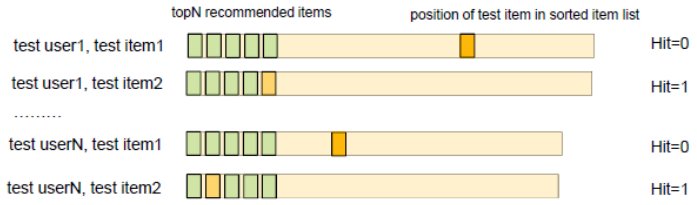
- Estimating Lift over Random

Make recommendations for each test user by selecting the top-K items in the sorted list.

Count how many times a test item appeared in the top-K (call this the #hits).

Compare with how many would be expected if random recommendations are made

E.g. if top-K = 5:



Hit% = (#hits) / (#recommendations made)

Lift = (hit% using model) / (hit% using random)

***For large item sets you may need large test sets (or large K) to get significant results!*

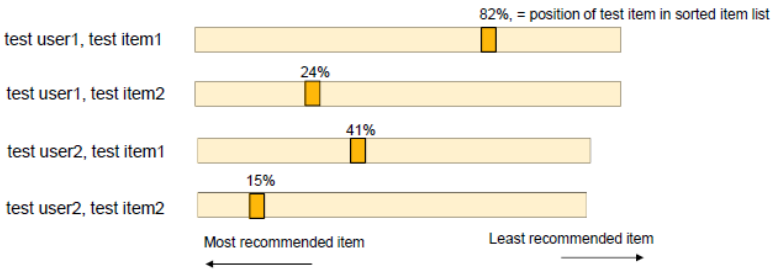
- Mean Percentage Ranking

As before, but only consider items that are ***liked***

For each test item, predict its position (as a %) in the ranked list of all items
(ranked by their decreasing likeability by the user, e.g. their predicted rating)

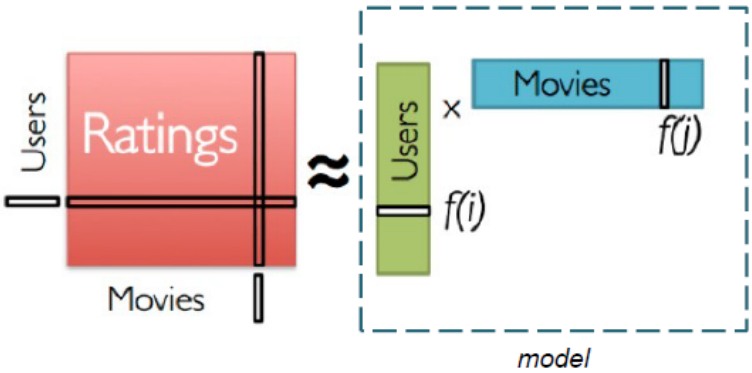
Since we know the item is liked we hope its predicted rank is low (to the left),
e.g. Rank=1 => top of list => top recommendation

MPR < 50% indicates test results are better than random guessing



MPR = mean(82, 24, 41,15) lower is better!

- Matrix Factorisation



- Matrix factorisation can be applied to the ratings matrix
- Converts it into two smaller matrices which can be thought of as a model.
- Once we have done the factorisation we make rating predictions using the two factor matrices – not the original ratings matrix
- Concept
  - Assume the items to be recommended have a set of properties.

	Length	Disney	Universal	Action	Comedy	SciFi	T. Hanks	B. Pitt
Movie1	4	10	0	2	8	0	8	2
Movie2	8	0	10	9	1	7	0	9

    - E.g. Movies could have properties: type, actors, directors, film studios, location, length etc. Properties are numerical strengths, e.g. values 0 (low) to 10 (high)
  - Assume users express their preferences using the same properties

	Length	Disney	Universal	Action	Comedy	SciFi	T. Hanks	B. Pitt
User1	2	7	5	4	8	5	5	9
User2	9	3	7	9	1	7	2	5

  - The users rating for any movie can now be computed as a simple dot product of user preferences * movie properties

	Length	Disney	Universal	Action	Comedy	Sci.Fi	T. Hanks	B. Pitt
M1	4	10	0	2	8	0	8	2
M2	8	0	10	9	1	7	0	9

	Length	Disney	Universal	Action	Comedy	Sci.Fi	T. Hanks	B. Pitt
U1	2	7	5	4	8	5	5	9
U2	9	3	7	9	1	7	2	5

    - E.g. User1's rating score for movie 2 is = 2*8 + 7*0 + 5*10 + … (to normalise divide by the max score)
  - BUT we don't have the movie properties or user preferences, we don't even know what the properties should be. We only have a set of ratings from the users (the ratings matrix R). E.g.

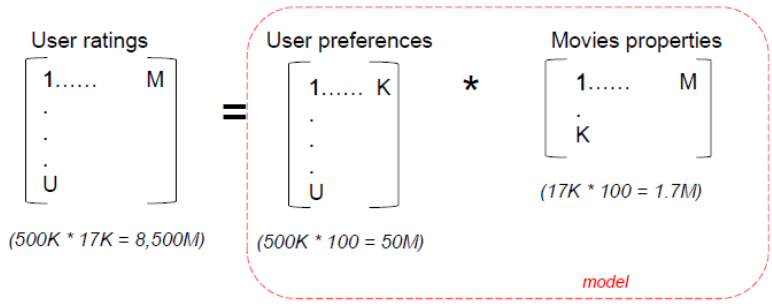
	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	.....	MovieN
U1	-	3	-	-	8	0	.....	3
U2	8	0	10	9	1	7	.....	-
U3							.....	6

- Solution ~ Assume the movies have K properties (e.g. say 100) and use matrix factorisation on R to derive the user preference matrix (U) and the movie properties matrix (M):

$$R = U^T M$$

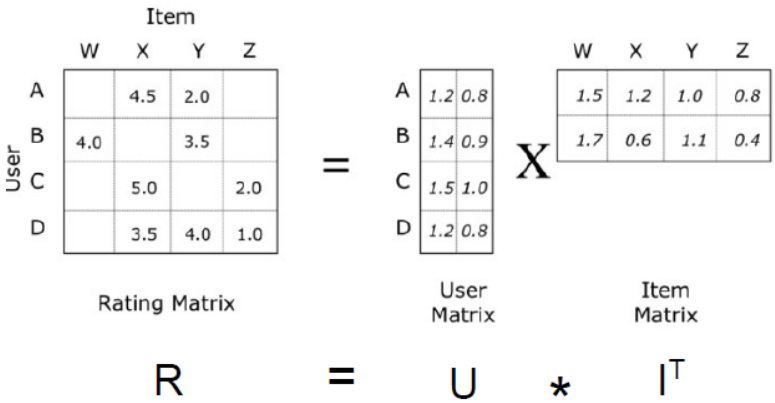
- We don't need to know what the K properties are, we just assume they exist, they are called latent (hidden) variables

- Pipleine



- The resulting matrices form the model ~ usually a much smaller dataset!
- 50 to 100 latent features are often all that is required for good results

Once factorisation is done we can predict the ratings for all users and all items



- Perfoming the Factorisation
  - SVD

    SVD (Singular Value Decomposition) is common method for matrix factorisation

    Issues = much of the ratings matrix is empty (very sparse matrix)

    Straight SVD doesn't work well with sparse data. Hard to find exact factors

    Instead the Netflix winner used incremental learning solution based on gradient descent by taking derivatives of the approximation error

    E.g. Use stochastic gradient descent to minimise the squared error between the predicted rating and the actual rating:

$$\min_{x_*, y_*} \sum_{r_{u,i} \text{ is known}} (r_{ui} - x_u^T y_i)^2 + \lambda(\|x_u\|^2 + \|y_i\|^2)$$

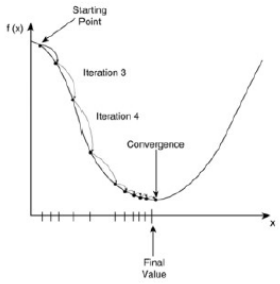
Actual rating — Predicted rating — Regularisation Term

- Gradient Descent

Procedure:
Present the training examples in turn and update the weights after each (similar to NN learning).

Continue until convergence (error stops reducing)

Training examples are triplets: (user, movie, rating)



- An example update rule for the user preference matrix **U** and the item properties matrix **M** is:

$$u_{ki}^{t+1} = u_{ki}^{t} + 2\gamma\,(r_{ij} - p_{ij})\; m_{kj}^{t} - \lambda u_{ki}^{t}$$

$$m_{kj}^{t+1} = m_{kj}^{t} + 2\gamma\,(r_{ij} - p_{ij})\; u_{ki}^{t} - \lambda m_{kj}^{t}$$

u_{ki} = value of kth preference for user I
m_{kj} = value of kth property for item j
r_{ij} = rating of user i on item j
p_{ij} = predicted rating of user i on item j
γ = the learning rate
λ = the regularisation constrant

- Overfitting & Regularisation

Given a high enough degree, a polynomial (or any other model) can be tuned to exactly learn the training data (over-fit). This is bad, the model will not generalise well.

Regularization tries to avoid overfitting by adding an additional penalty term into the error function. The additional term encourages the coefficients to be small so that the model doesn't become over-complex

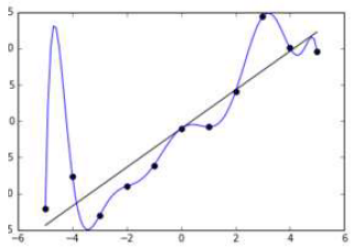
$$L(x,y) = \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2$$

where $h_\theta x_i = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3 + \theta_4 x_4^4$

$$L(x,y) \equiv \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2 + \boxed{\lambda \sum_{i=1}^{n} \theta_i^2}$$

$$f(x_i) = h_\theta x = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \cancel{\theta_3 x_3^3} + \cancel{\theta_4 x_4^4}$$

$$f(x_i) = h_\theta x = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2$$



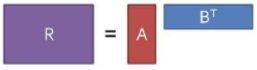
We want the model to represent the training data, not to reproduce the training data!

For MF, keeping the coefficients small means keeping the values in the factorized matrix small – this will be more crucial if the number of latent features is set as a large number (more potential to overfit)

The optimal number of latent features is usually determined by trial-and-error using a test data set, but starting with a rule-of-thumb initial value (e.g. 50→100)

- Alternating Least Squares Algorithm

Model R as product of user and movie feature matrices A and B of size U×K and M×K



We wish to find values for $\mathbf{a}_i$ and $\mathbf{b}_j$ that minimises the total squared error:

$$\text{cost} = \sum_{ij} (r_{ij} - a_i * b_j)^2$$

- If we fix **B** and optimize for **A** alone, the problem is reduced to the problem of linear regression:

$$\mathbf{y = \beta X} \quad y = \beta_1 \cdot x_1 + \beta_2 \cdot x_2 + \beta_3 \cdot x_3 + \ldots$$

- The matrix B corresponds to the training data (the input variables **X**), the matrix R corresponds to the output variables (**y**), and A corresponds to the coefficients **β**

Alternating Least Squares (ALS)
» Start with random A & B
» Optimize user vectors (A) based on movies
» Optimize movie vectors (B) based on users
» Repeat until converged

- Alternating Least Squares

R =

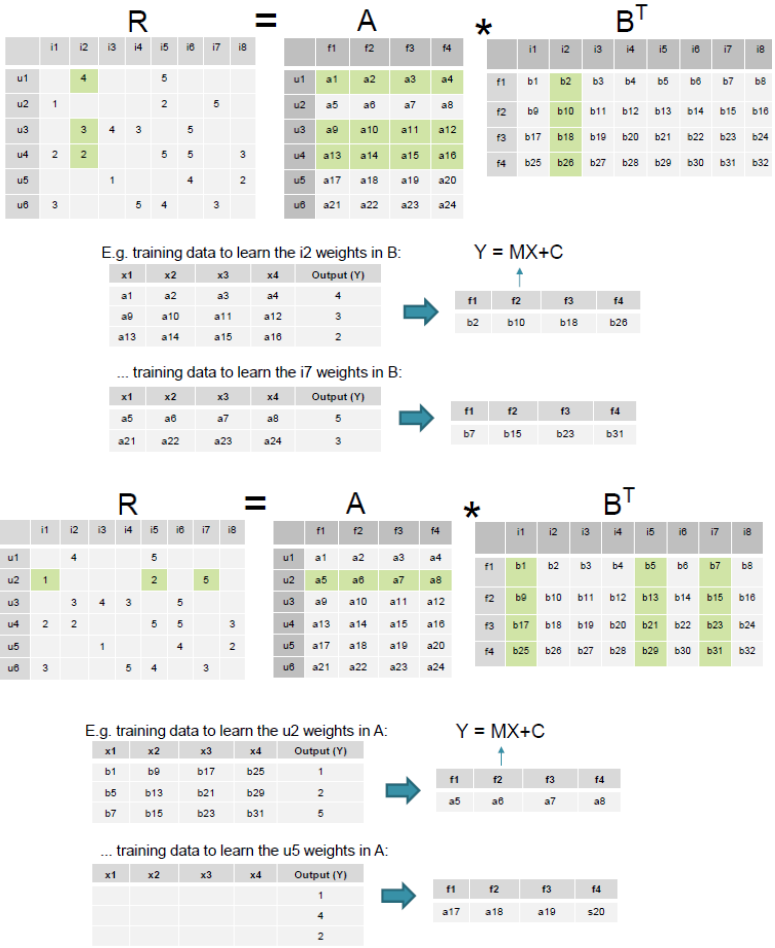
	i1	i2	i3	i4	i5	i6	i7	i8
u1		4			5			
u2	1				2		5	
u3		3	4	3		5		
u4	2	2			5	5		3
u5			1			4		2
u6	3			5	4		3	

A =

	f1	f2	f3	f4
u1	a1	a2	a3	a4
u2	a5	a6	a7	a8
u3	a9	a10	a11	a12
u4	a13	a14	a15	a16
u5	a17	a18	a19	a20
u6	a21	a22	a23	a24

* B^T

	i1	i2	i3	i4	i5	i6	i7	i8
f1	b1	b2	b3	b4	b5	b6	b7	b8
f2	b9	b10	b11	b12	b13	b14	b15	b16
f3	b17	b18	b19	b20	b21	b22	b23	b24
f4	b25	b26	b27	b28	b29	b30	b31	b32

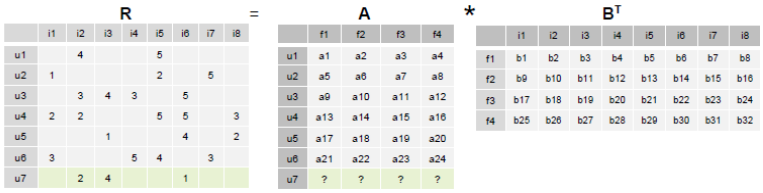


- Step1: Assign random weights to A and B (weights and ratings should be normalised to range 0 to 1)
- Step2 : fix A and use linear regression* to estimate weights in B will require building 8 (#items) regression models
- Step3: fix B and use linear regression* to estimate weights in A will require building 6 (#users) regression models Repeat from step2 until convergence (or time limit etc)
- OVERFIT

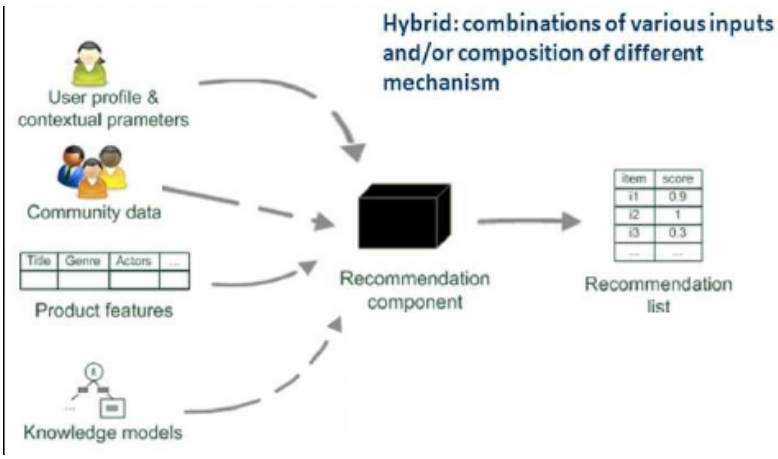
> * To help avoid overfitting we can conceptually substitute the equivalent of:
> - Lasso-regression, or
> - Ridge-regression
>
> (for L1 and L2 regularisation)
>
> $$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$
>
> $$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

- Handling New Users / Items
  - No cold-start: Cannot predict for a user with no existing ratings (or a new item)

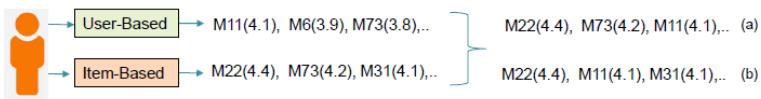
        - Ask the user for sample ratings, else wait until they have done some page-views, transactions etc
    - Must re-factorise (which can take time)…



    - Or factorise just for the new user (reuse the existing item factors matrix)
- Hybrid Recommender Systems



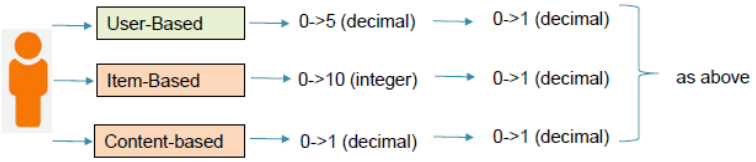
    - Often combining many different systems can generate better results
    - Techniques
        - Mixed: Present all recommender results together
        - Weighted: Numerically combine the scores
        - Switching: Switch between different recommendations according to user profile & context
        - Cascade: Assign recommenders a priority, the low priority ones break ties between the higher ones
        - Feature Combination: Combine features from different sources and input to a single recommender
        - Feature Augmentation: One recommender generates features that form part of the input to another
        - Meta-level: One recommendation technique is applied and produces a model, this is then the input to the next technique.
    - Numerical Combination of Scores
        - If recommender outputs have the same range (e.g. predicted rating 1->5) then they can be numerically combined, e.g.



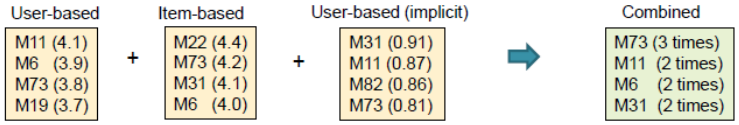
        - a) Pick the topN items with the highest predicted ratings, or

- b) Average the predicted ratings for each item, pick the topN highest

- If the outputs have different ranges then can normalise all to be the same range

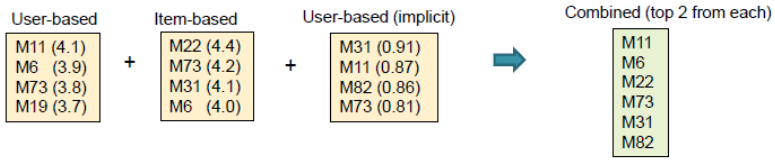


- Pick the most frequent items in the topN , with tie break:

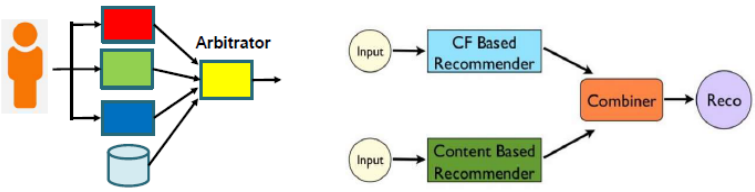


  - For small N and many items its possible (likely) than the top items are all different hence one solution is to make N larger and then select the best K items (K < N)

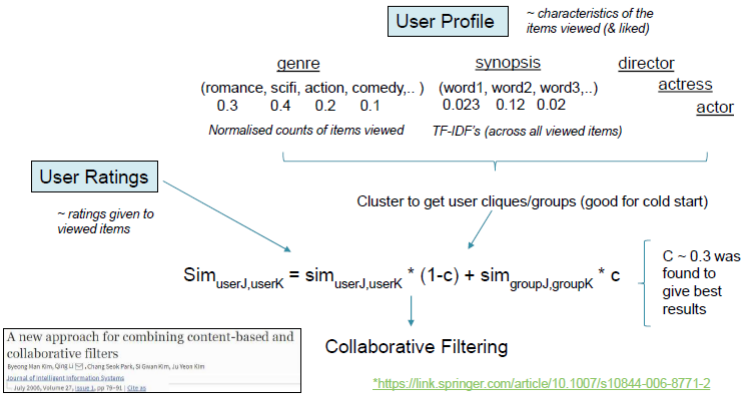
- pick a selection from the topN of each recommender:



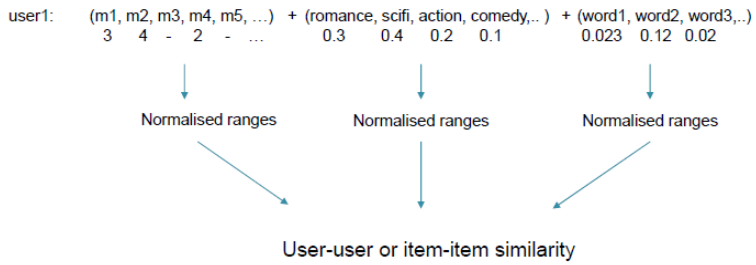
- Weighted Combination



- Feature Weighted Stacking
  - Many different recommenders (models) are stacked up
  - Another model is used to weight their votes
  - Can select weights manually via experimentation
  - Or, use a ML algorithm to learn the weights
    - E.g. linear regression
    - Training signal could be rating prediction error
- Weighted Combination of Similarities
  - Combining User Ratings with Content-Based Filtering (example)*

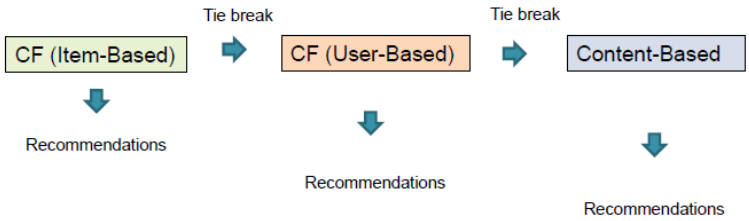


- Feature Combination

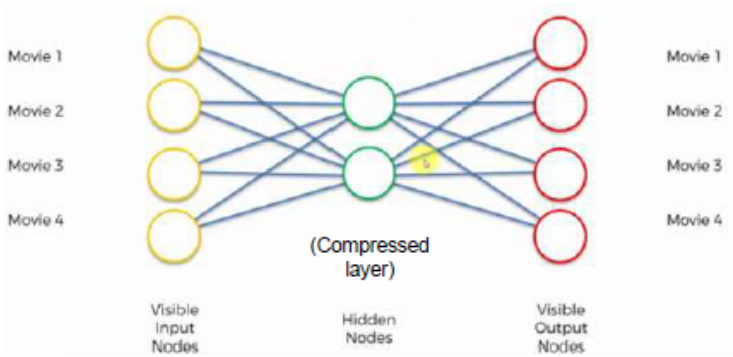


  - Combine the features together into one large user-record and input into a single recommender

- Cascading Example
  - Assign recommenders a priority, the low priority ones break ties between the higher ones



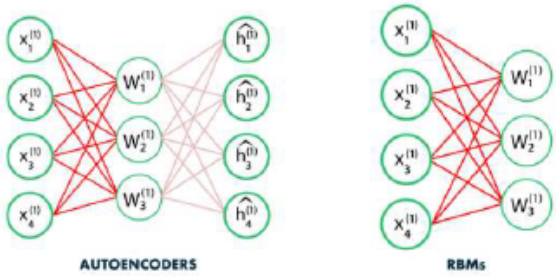
- Recommendations using Deep Neural Networks
  - Auto-Encoders



    - Auto-Encoders offer one way to learn how to predict ratings for unseen items
    - In Auto-Encoders, the input and output layers are identical. During training the hidden layer weights are adjusted so that the output matches the input (with minimum error).
    - Often used for image compression and image reconstruction. For recommender systems the input is the user ratings vector
    - The hidden (compressed) layer can be though of as analogous to the latent features in MF
    - After training, we can input a user with a few ratings and the output will contain the predicted ratings for that user for all of the movies
  - Restricted Bolzman Machine

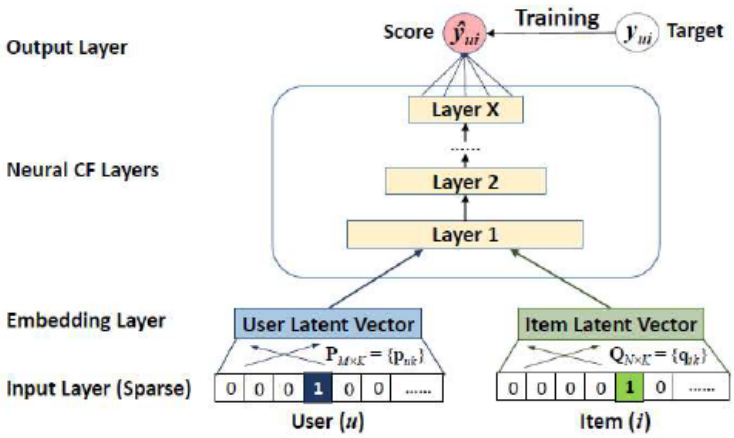
Difference between Autoencoders & RBMs

Autoencoder is a simple 3-layer neural network where output units are directly connected back to input units. Typically, the number of hidden units is much less than the number of visible ones. The task of training is to minimize an error or reconstruction, i.e. find the most efficient compact representation for input data.



RBM shares a similar idea, but it uses stochastic units with particular distribution instead of deterministic distribution. The task of training is to find out how these two sets of variables are actually connected to each other.

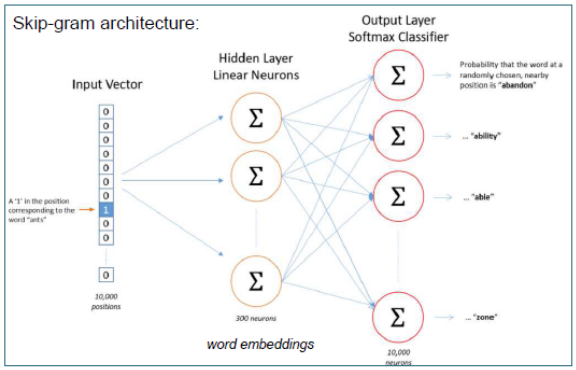
- CNN



  - When using matrix factorisation, ratings are predicted by taking the inner (dot) product of the user latent features and the item latent features
  - In the NCF* system, the inner product is replaced with a neural architecture that can learn an arbitrary function from data
- Leveraging NLP methods: Prod2Vec

Many NLP methods exist for mapping words into vector space such that similar words are near each other. The word vectors (embeddings) represent how a word interacts with other words in a corpus

Word2Vec (Google) trains words against other words that neighbour them in the input corpus. Skip-Gram is one architecture. Can be used to predict the next word in a sentence (language model)



http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

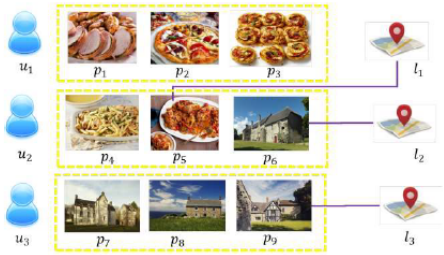
Prod2Vec: Treats products as word tokens and uses user sequences of products (analogical to sentences) to learn product embeddings. The user sequence of products is the sequence of placing items into a basket

After training, supply a product as input to get likelihoods of other products being nearby.

Or use the vectors (embeddings) to generate a product-product similarity matrix for content-based filtering

- Visual 'Likes': Handling Product Images

  - E.g. Point of Interest Recommendation
  - With location-based social networks such as Yelp and Instagram, users can upload photos of Points of Interest. These reflect user interests and also provide informative descriptions about locations. E.g. a user who posts architecture photos is more likely to visit famous landmarks; a user who posts images about food is more likely to visit restaurants.

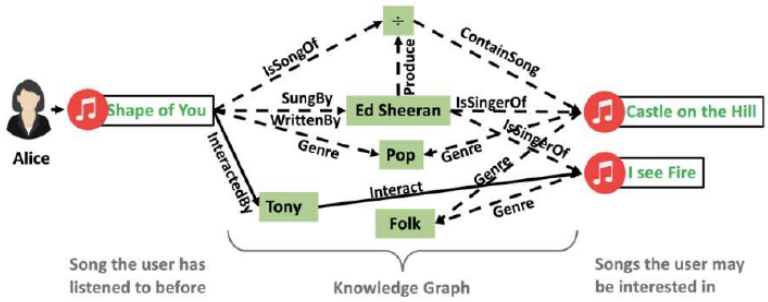


Example of Images Posted by Users.

A CNN (Convolutional Neural Network) is used to extract features from the uploaded images, these are then used as inputs by a probabilistic matrix factorisation engine

http://www.public.asu.edu/~skai2/files/WWW_suhang.pdf

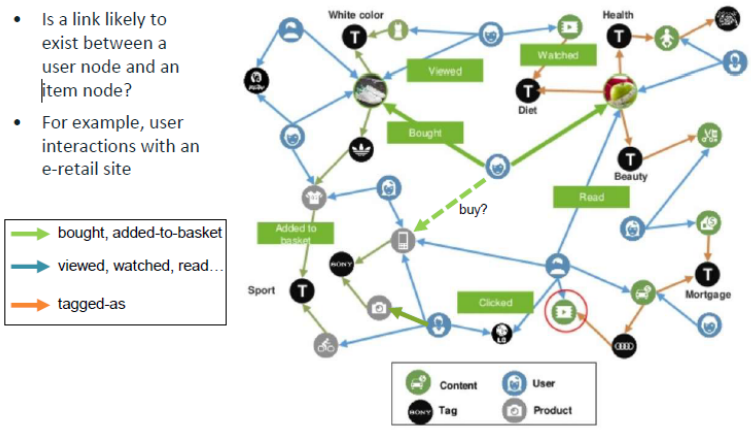
- Recommendation as a Graph Problem



  - By exploring the interlinks within a knowledge graph, the connectivity between users and items can be discovered as paths, which provide rich and complementary information to user-item interactions

- Recommendation using Link Prediction

- Is a link likely to exist between a user node and an item node?
- For example, user interactions with an e-retail site



- Predictive Modelling Approach
  - Generate training examples by considering all possible links ~ (a,b) pairs in the graph
  - Model inputs = node properties and graph topology (relationship) features
  - Model output = T/F does a link exist

node1					node2					Topology ??				
sex	age	income	status	…	sex	age	income	status	…					link
M	45	12000	single	…	M	23	5000	married	…					Y
M	45	12000	single		F	51	7200	married						N

Model inputs Model output

- Instead of explicitly creating/selecting the features (model inputs) , we can use ML to learn a lower dimensional vector representation of the graph nodes and topology – these representations are called embeddings

- Challenges and Issues – User Experience
  - The success of a recommender systems depends on more than just accuracy:
    - Diversity - users tend to be more satisfied with recommendations when there is a higher diversity, e.g. items from different artists. There is unlikely to be a single best recommendation, allow the user to treat the RS as a knowledge discovery tool
    - Serendipity - how surprising are the recommendations?
    - Avoiding bad recommendations - assigning a cost to them
    - Long versus short term recommendations. How far ahead can the recommendation be? e.g. associations found between items over longer time
    - Repeat Recommendations - sometimes it may be more effective to re-show recommendations or let users re-rate items, than showing new items. Users may ignore a recommendation first time but still like the item, e.g. if short of time
    - Recommending Sequences – sometimes the sequence is important e.g. a compilation of musical tracks from slow to fast, episodes in Game of Thrones etc.
  - Privacy - push-back by users if they feel the RS is collecting too much information about them.
  - Trust can be built by explaining how the recommendations are generated and why an item is being recommended
    - Fake reviews and fake ratings erode trust
  - Computability
    - Scalability – handling very large numbers of users and items

- Cold-start – making recommendations to a new user
- Sparsity - recommending items in the long-tail is hard since there are very few ratings / purchases for them