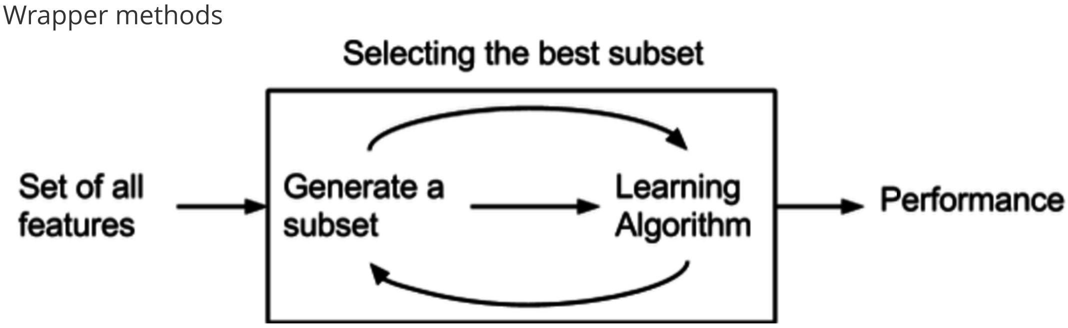
*By Hawk 2018-1-12*

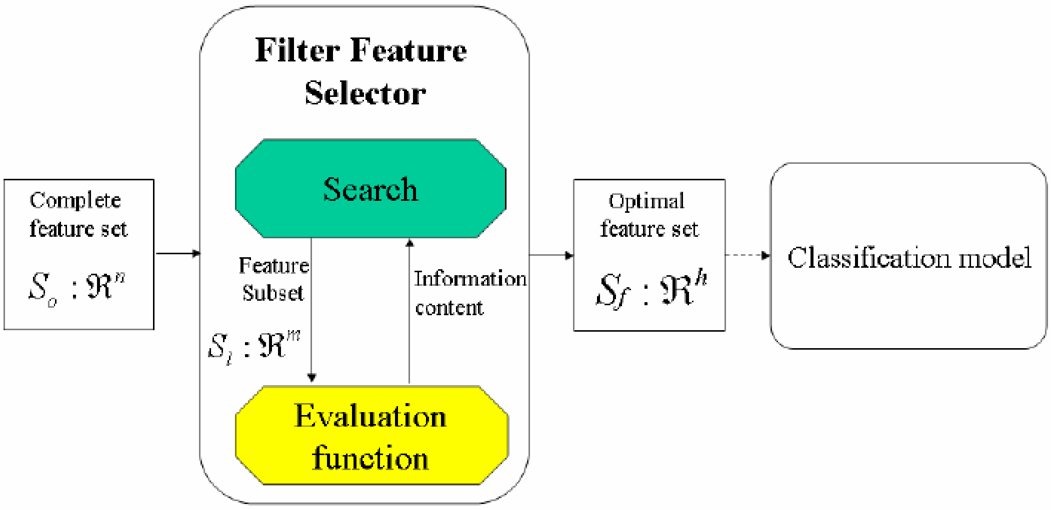
# Feature Engineering

- `Features and Feature Engineering`
  - **Features**: information that describes a problem at hand and is potentially useful for prediction/problem solving
  - **Feature Engineering**: design and process for AI applications
  - Process:
    - understanding the properties of the task and how they may interact with the strengths and limitations of the chosen model
    - design a set of features
    - run experiments and analyse the results on a validation dataset
    - change the feature set
    - go to step 2
- `Feature Explosion`
  - Initial features: an expression of prior knowledge
  - Features combinations
  - Problem:  Storage Cost;  Irrelevant, Redundant or even harmful features; Large number of required training samples; Dysfunctional distance functions
  - Benefits of small features set
- `Tackling Feature Explosion`
  - Feature selection: greedy method
    - Reduce the original feature by throwing out some redundant features
    - Greedy heuristic search for feature selection (**sequential feature selection**)
      - **Forward selection**: add one feature each step
      - **Backward selection**: remove one feature each step
      - **Evaluate method**: information theory; prediction accuracy
      - **Stop criterion**
    - Three typical methods:
      - Wrapper methods




        highly accurate, computationally expensive, risk of over fitting

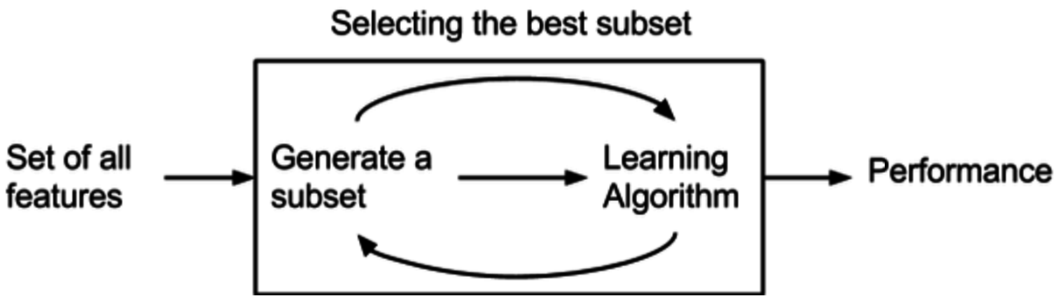
      - Filter methods




fast and simple; not as accurate as wrappers

Examples: correlation-based filters (**Pearson correlation**, **Information gain**)

- Embedded method




similar to wrappers but less computationally expensive; less possible to over fitting

Example: classification and regression trees

- Regularization: (introducing penalty for complexity -> reduce features)
  - Ridge regression (2-order)
  - Lasso regression (1-order)

# Unsupervised Learning

1. `What is clustering`
   - group the points into some number of clusters
   - members of a cluster are close /similar to each other
   - members if different clusters are dissimilar
   - It is hard for clustering high dimension data.
   - Distance measurements:
     - Cosine similarity and distance

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \sqrt{\sum\limits_{i=1}^{n} B_i^2}}$$

$D_C(A,\ B) = 1 - S_C(A,\ B)$ or $distance = \frac{cos^{-1}(similarity)}{\pi}$

- Jaccard similarity and distance (Two sets A and B)

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.\ 0 \le J(A,B) \le 1$$

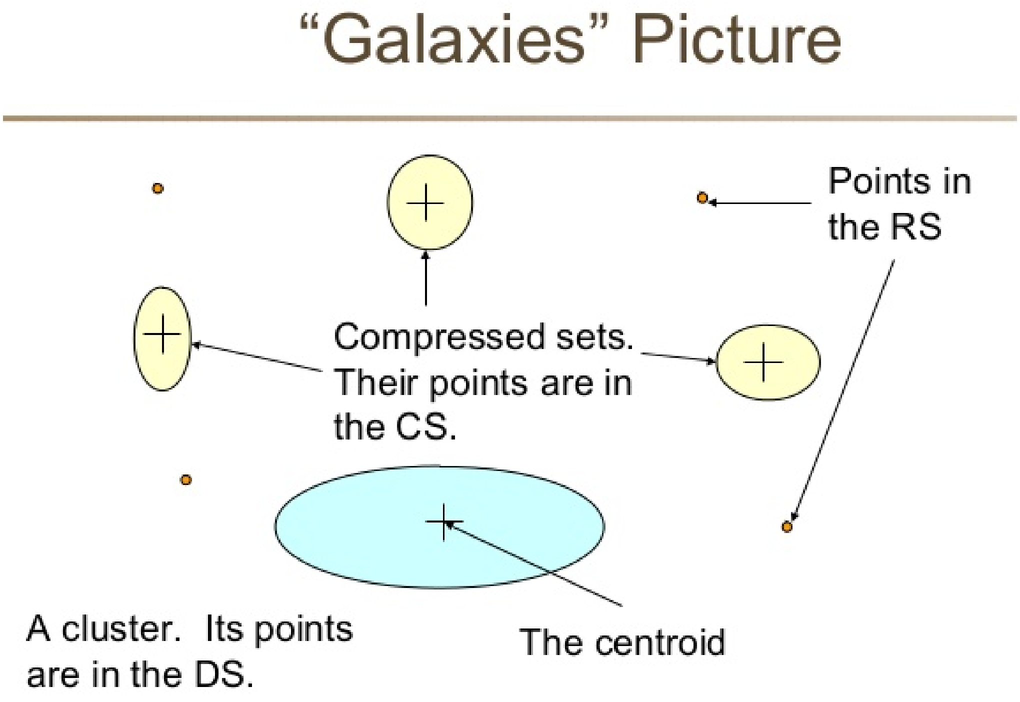
$$d_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}.$$

- Euclidean distance

2. `Hierarchical clustering` (**Repeatedly combine** , **two nearest clusters**)
   - represent a cluster of mangy points: **centroid ( average of its points)**
   - how to determine "nearness" of clusters: **distance of centroids** (and other criterion)
   - when to stop. number criterion, distance criterion
   - Additionally, **in the non-Euclidean space**
3. `k-means clustering` (scan quickly)
4. `The BFR algorithm` Extension of k-means to large data
   - assumes that clusters are normally distributed a centroid in an Euclidean space

```
Select initial k centroids by some approach:
    1. random selection
    2. samll random sample and cluster optimally
    3. take sample one by one, each as far from the previously slected points as possible
```

   - 3 set of points which we keep track of
     - Discard set (DS): points close enough to a centroid to be summarized
     - Compression set (CS): group of points are close together but not close to any existing centroid
     - Retained set (RS): isolated points waiting to be assigned to a compression set




- Summarizing sets of points
  - the number of points: N
  - the vector SUM: $SUM_i = i^{th}$ component of SUM
  - the vector SUMSQ: $SUMSQ_i$ = sum of squares of $i^{th}$ component
  - represent cluster: $(d, SUM_i, (SUMSQ_i/N) - (SUM_i/N)^2$
- Actual clustering

```
1. Find those points that are “sufficiently close” to a cluster centroid and add those
points to that cluster and the DS
2. Use any main-memory clustering algorithm to cluster the remaining points and the old RS
3. DS set: Adjust statistics of the clusters to account for the new points (Ns,SUMs,SUMSQs)
4. Consider merging compressed sets in the CS
5. If this is the last round, merge all compressed sets in the CS and all RS points into
their nearest cluster
```

**Mahalanobis Distance to decide whether to put a new point into a cluster (and discard)**

**Combine 2 CS sub-clusters if the combined variance is below some threshold.**

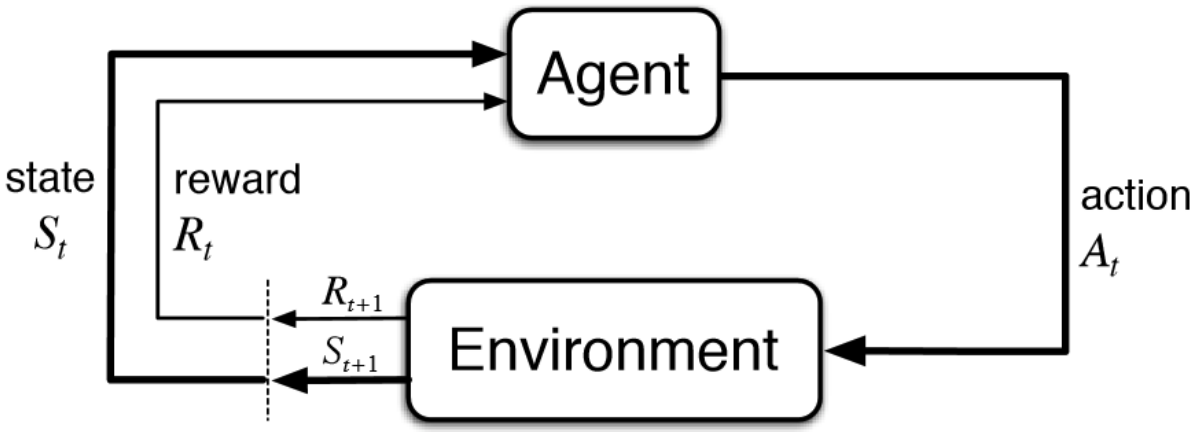
5. `The CURE algorithm`

```
0. Pick a random sample of points that fit in main memory
1. Clustering: group nearest points/clusters
2. Pick representative points:
     For each cluster, pick a sample of points, as dispersed as possible
     From the sample, pick representatives by moving them (say) 20% toward the centroid of
       the cluster
3. Now, rescan the whole dataset and visit each point p in the data set
4. Find the closest representative to p and assign it to representative’s cluster
```

6. `Spectral clustering`

## Markov Decision Process

1. `Basic concepts`




   - receive current state state $S_t$ and reward $R_t$ from the environment
   - Take action $A_t$
   - Environment changes to $S_{t+1}$ and $R_{t+1}$
   - **Objective of interaction** : maximise total reward
   - **Objective if learning**: find out such actions from the information collected during interaction
   - Learn a policy (a mapping from states to actions)
   - $A_t$ affects $S_{t+1}, S_{t+2}, S_{t+3}, \ldots$ and $R_t + 1, R_{t+2}, R_{t+3}, \ldots$ all $R_{t+k}$ contribute to the cumulative reward starting from time t.
   - Reinforcement Learning vs Supervised Learning
     - RL: learn a policy;  SL: learn a classifier
     - No teachers in RL; Feedbacks in SL is instant
     - ...

2. `Markov decision process`

   - **MDP: M=<S, A, P, R>** S: set of all possible states; A: set of all possible actions
   - $\mathbb{P}(S_{t+1}|S_1, \ldots, S_t, A_1, \ldots, A_t) = \mathbb{P}(S_{t+1}|S_t, A_t)$ the future only correlate with present state
   - P: transition probability $p(s, a, s^{'}) = \mathbb{P}(S_{t+1} = s^{'}|S_t = s, A_t = a)$
   - R: immediate reward function; R: S->R; R: S*A -> R; R: S*A*S -> R

- **Policy**: describe the behaviour of agent
  - Deterministic policy (Discrete): ($\pi(s) = a$: instructs the agent to take action a at state s )
  - Stochastic policy (Continue): $\pi$ is a conditional distribution over actions given states
- **Cumulative reward**
  - total amount of reward rather than just a single-step
  - discounted cumulative reward: $G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k}$
  - $\gamma \in (0,1)$ $\gamma \to 1$ far-sighted; $\gamma \to 0$ myopic
  - $G_t = R_{t+1} + \gamma G_{t+1}$
- **Value functions** (maximise expected cumulative reward)
  - state Value functions ($\boldsymbol{v^\pi}$): $v^\pi(s) := E[G_t|\pi, S_t = s]$
  - action Value functions ($\boldsymbol{q^\pi}$): $q^\pi(s,a) := E[G_t|\pi, S_t = s, A_t = a]$
- **Bellman equation**
  - for state value function:

$$\begin{aligned} v^\pi &= E[G_t|\pi, S_t = s] \\ &= E[R_{t+1} + \gamma G_{t+1}|\pi, S_t = s] \\ &= E[R_{t+1} + \gamma v^\pi(S_{t+1})|\pi, S_t = s] \end{aligned}$$

    since the transition probability is $p(s, \pi(s), s^{'})$

$$v^\pi(s) = \sum_{s^{'} \in S} p(s, \pi(s), s^{'})(R(s, \pi(s), s^{'}) + \gamma v^\pi(s^{'}))$$

    for stochastic $\pi(a|s)$

$$v^\pi(s) = \sum_{a \in A} \pi(a|s) \sum_{s^{'} \in S} p(s, \pi(s), s^{'})(R(s, \pi(s), s^{'}) + \gamma v^\pi(s^{'}))$$

  - for action value function:

$$\begin{aligned} q^\pi(s,a) &= E[G_t|\pi, S_t = s, A_t = a] \\ &= E[R_{t+1} + \gamma G_{t+1}|\pi, S_t = s, A_t = a] \\ &= E[R_{t+1} + \gamma v^\pi(S_{t+1})|\pi, S_t = s, A_t = a] \end{aligned}$$

    since action at the current step is always $a$ regardless of $\pi$ in $q^\pi(s,a)$

$$q^\pi(s,a) = \sum_{s^{'} \in S} p(s, a, s^{'})(R(s, a, s^{'}) + \gamma v^\pi(s^{'}))$$

    compare

$$\begin{cases} v^\pi(s) = \sum_{a \in A} \pi(a|s) \sum_{s^{'} \in S} p(s, \pi(s), s^{'})(R(s, \pi(s), s^{'}) + \gamma v^\pi(s^{'})) \\ q^\pi(s,a) = \sum_{s^{'} \in S} p(s, a, s^{'})(R(s, a, s^{'}) + \gamma v^\pi(s^{'})) \end{cases}$$

    we have

$$v^\pi(s) = \sum_{a \in A} \pi(a|s) q^\pi(s,a)$$

    thus,

$$
\begin{aligned}
q^{\pi}(s,a) &= \sum_{s^{'}\in S} p(s,a,s^{'})(R(s,a,s^{'}) + \gamma v^{\pi}(s^{'})) \\
&= \sum_{s^{'}\in S} p(s,a,s^{'})(R(s,a,s^{'}) + \gamma \sum_{a^{'}\in A} \pi(a^{'}|s^{'}) q^{\pi}(s^{'},a^{'}))) \\
&= \sum_{s^{'}\in S} p(s,a,s^{'})(R(s,a,s^{'}) + \gamma q^{\pi}(s^{'},\pi(s^{'})))
\end{aligned}
$$

3. `Planning in MDPs`
   - Optimality
     - optimal state value function: $v^*(s) = max_{\pi} v^{\pi}(s)$
     - optimal action value function: $q^*(s,a) = max_{\pi} q^{\pi}(s,a)$
     - **optimal policy have the highest expected cumulative reward at any state**
   - Solve Bellman equation
     - Policy iteration algorithm

       Policy iteration (PI) algorithm

       - (0) start from arbitrary $\pi$
       - (1) solve $q^{\pi}$
       - (2) improve $\pi$ by $\pi(s) \leftarrow \text{argmax}_{a\in\mathcal{A}}\, q^{\pi}(s,a)$
       - (3) goto (1) until $\pi$ converges

     - Value iteration algorithm

       Value iteration (VI) algorithm

       - (0) start from random $v$ (can be wrong values like all 0!)
       - (1) update all $v(s)$ by $v(s) \leftarrow \max_{a\in\mathcal{A}} \sum_{s'\in S} \mathcal{P}(s,a,s')\big(\mathcal{R}(s,a,s') + \gamma v(s')\big)$
       - (2) goto (1) until $v$ converges

4. `Extensions to MDPs`
   - POMDPs: *<S, A, O, P ,R, W>* O: set of all observations; W: an observation probability function
   - continue MDPs
   - Semi-MDPs
   - Decentralised POMDPs

## Reinforcement Learning

*what if the agent does not have such full knowledge*

1. `Model-based RL` (Estimate during interaction)
   - $\hat{M} := < \hat{S}, \hat{A}, \hat{P}, \hat{R} >$
     - $\hat{S}, \hat{A}$ set of all **visited** states and actions

- $\hat{R}$: estimated transition probabilities
  - $\hat{R}(s,a,s^{'}) = R_{s,a,s'}/N_{s,a,s'}$
  - $R_{s,a,s'}$ be the sum of rewards received at transition *(s, a, s')* in the whole interaction history
  - $N_{s,a,s'}$ be the number of transition *(s, a, s')*occurred
- $\hat{P}$: estimated immediate rewards
  - $\hat{P}(s,a,s^{'}) = N_{s,a,s'}/N_{s,a}$
  - $N_{s,a}$ be the number of 'taking action *a* at state *s'* in the whole interaction history
  - $N_{s,a,s'}$ be the number of transition *(s, a, s')* occurred

- (Vanilla) model-based RL algorithm
  - (0) Start with an arbitrary policy $\pi$ and an estimated MDP $\widehat{M}$
  - (1) Interact with the environment using $\pi$, record transitions and rewards
  - (2) Update estimated MDP $\widehat{M}$
  - (3) Compute the optimal policy $\hat{\pi}^*$ of $\widehat{M}$ using PI/VI, update $\pi \leftarrow \hat{\pi}^*$
  - (4) Goto (1) until $\pi$ converges

2. `Exploration vs exploitation in RL` **trade off**
   - Exploration: deliberately take actions that are not (seemingly) "optimal" according to the current knowledge
     - $\varepsilon$-greedy: choose a random action with probability $\varepsilon$ ; choose the 'optimal' action with $1-\varepsilon$
     - $\varepsilon$ often set small values like 0.03, 0.01, 0.003 or even smaller
     - R-MAX: Assume $q(s,a)$=$R_{max}$ , unless *a* has been taken at least *m* times at *s*
     - R-MAX forces the agent to try every possible actions many times before making any conclusion
   - Exploitation: gain more information in the hope of discovering better policies
3. `Model-free RL`
   - Monte-Carlo methods
     - Monte-Carlo value estimation:

$$N(s) \leftarrow N(s) + 1$$
$$\hat{G}(s) \leftarrow \hat{G}(s) + G_t$$
$$\hat{v}^{\pi}(s) \leftarrow \hat{G}(s)/N(s)$$

       $G_t$ is the actual cumulative reward $G_t = R_{t+1} + \gamma Rt + 2 + \cdots + \gamma^k R_{t+k+1} + \ldots$
     - MC reinforcement learning

## Monte-Carlo reinforcement learning

- (0) Start with arbitrary policy $\pi$
- (1) Interact with the environment, record all cumulative rewards
- (2) Update $\hat{v}^\pi$ or $\hat{q}^\pi$ with the MC value estimation algorithm
- (3) Improve $\pi$ by argmaxing $\hat{v}^\pi$ or $\hat{q}^\pi$
- (4) Goto (1) until $\pi$ converges

Incremental version of MC estimation

$$\hat{v}^\pi(S_t) \leftarrow \hat{v}^\pi(S_t) + \frac{1}{N(S_t)}(G_t - \hat{v}^\pi(S_t))$$
$$\hat{v}^\pi(S_t) \leftarrow \hat{v}^\pi(S_t) + \alpha(G_t - \hat{v}^\pi(S_t)) \quad 0 < \alpha < 1$$

$\alpha$ is the update rate

- Temporal difference terminology

$$\hat{v}^\pi(S_t) \leftarrow \hat{v}^\pi(S_t) + \alpha(G_t - \hat{v}^\pi(S_t))$$
$$G_t = R_{t+1} + \gamma \hat{v}^\pi(S_{t+1})$$
$$\hat{v}^\pi(S_t) \leftarrow \hat{v}^\pi(S_t) + \alpha(R_{t+1} + \gamma \hat{v}^\pi(S_{t+1}) - \hat{v}^\pi(S_t))$$

  - "TD target": $R_{t+1} + \gamma \hat{v}^\pi(S_{t+1})$
  - "TD error": $R_{t+1} + \gamma \hat{v}^\pi(S_{t+1}) - \hat{v}^\pi(S_t)$

## (Vanilla) temporal difference RL

- (0) Start with an arbitrary policy $\pi$
- (1) Execute $A_t \leftarrow \pi(S_t)$, get $R_{t+1}$ and $S_{t+1}$
- (2) Update $\hat{v}^\pi(S_t)$ or $\hat{q}^\pi(S_t, A_t)$ with TD estimation
- (3) Improve $\pi(S_{t+1})$ by argmaxing $\hat{v}^\pi(S_{t+1})$ or $\hat{q}^\pi(S_{t+1}, a)$
- (4) Goto (1) until $\pi$ converges

- MC vs TD
  - MC: unbiased, but usually has a higher variance
  - TD:  biased, but usually has a lower variance
- Other version of TD algorithms
  - Sarsa algorithm

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Repeat (for each step of episode):
    Take action $a$, observe $r$, $s'$
    Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma Q(s',a') - Q(s,a)\big]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until $s$ is terminal

- Q-learning algorithm

The learning rate, i.e. that extent to which new information overrides old information. This is a number between 0 and 1.

The $Q$ function we are updating, based on state s and action $a$ at time $t$.

The reward earned when transitioning from time $t$ to the next next turn, time $t$+1.

The value of the action that is estimated to return the largest (i.e. maximum) total future reward, based on all possible actions that can be made in the next state.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_{t+1} + \lambda \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\big]$$

The arrow operator means update the $Q$ function to the left. This is saying, add the stuff to the right (i.e. the difference between the old and the new estimated future reward) to the existing $Q$ value. This is equivalent in programming to A = A+B.

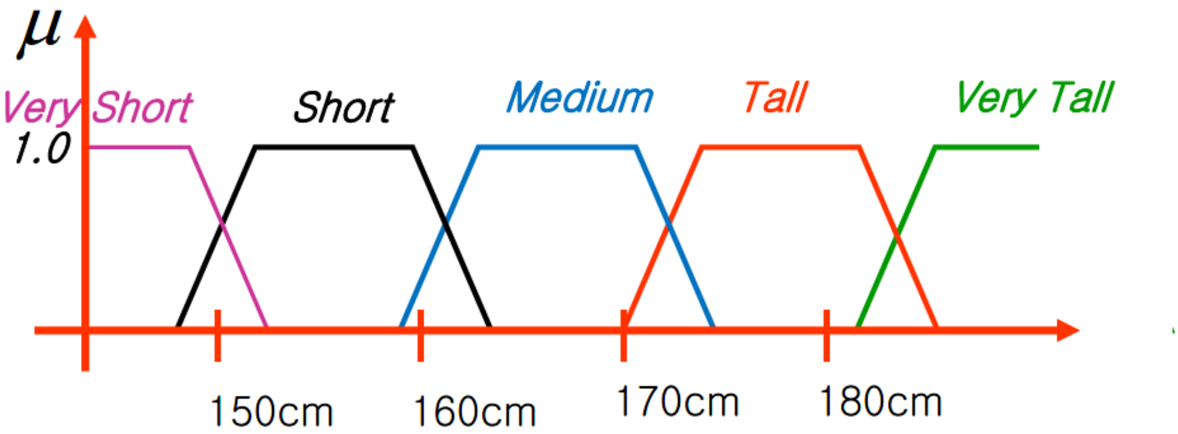
The discount rate. Determines how much future rewards are worth, compared to the value of immediate rewards. This is a number between 0 and 1.

The existing estimate of the $Q$ function, (a.k.a. current the action-value).

4. `Issues with terminology` (details in slides)

## Fuzzy Logic

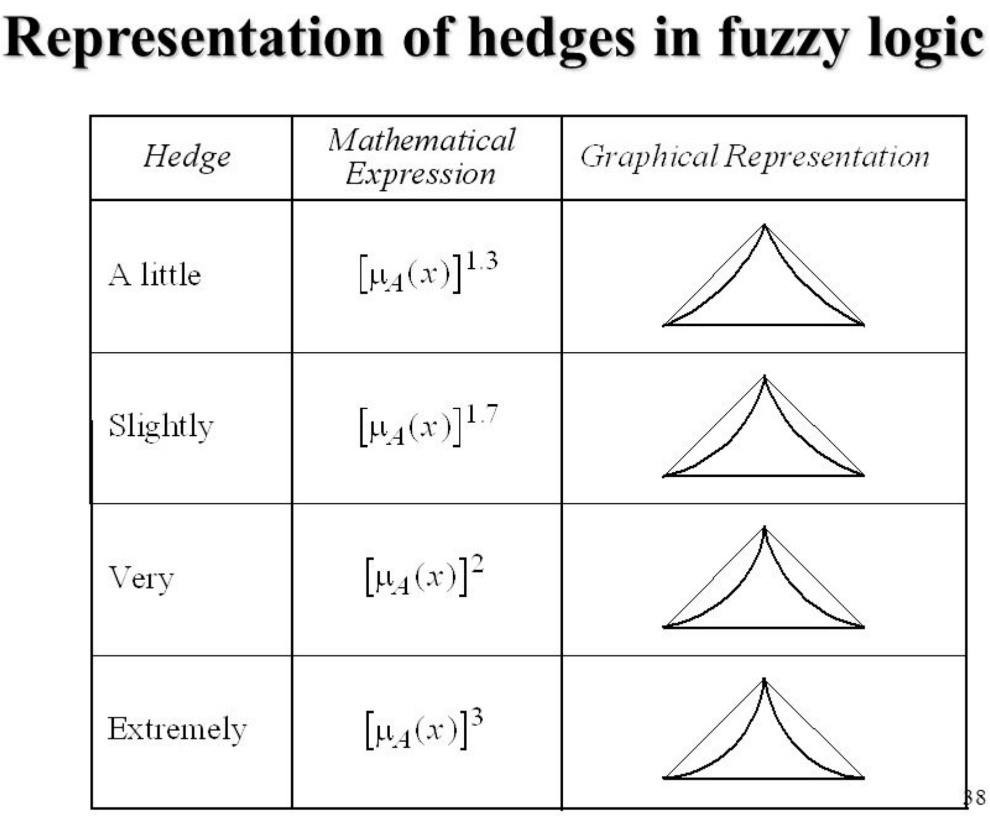
1. `Introduction` details in slides
2. `Fuzzy Sets anf Membership Functions`
   - fuzzy set: fundamental to mathematics. Example: "tall man" is a fuzzy set
   - Membership Function: the degree of an element of universe X belong to a fuzzy set.
     - $\mu_A(x) = 1$ if x is totally in A;
     - $\mu_A(x) = 0$ if x is not in A;
     - $0 < \mu_A(x) < 1$ if x is partly in A;
   - Example:




5 Fuzzy Set: Very Short, Short, Medium, Tall, Very Tall

Membership Function: curve in the coordinate system

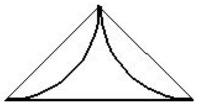
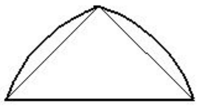
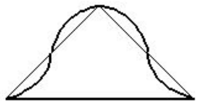
3. `Fuzzy Linguistic Variables`
    - example of FLV:
        - Colour:  red, blue, green,...
        - age: young, middle-aged, old, very-old,
        - size: small, big, very big, ...
    - representation of hedges in fuzzy logic



# Representation of hedges in fuzzy logic

| *Hedge* | *Mathematical Expression* | *Graphical Representation* |
|---|---|---|
| A little | $[\mu_A(x)]^{1.3}$ | |
| Slightly | $[\mu_A(x)]^{1.7}$ | |
| Very | $[\mu_A(x)]^{2}$ | |
| Extremely | $[\mu_A(x)]^{3}$ | |

## Representation of hedges in fuzzy logic (continued)

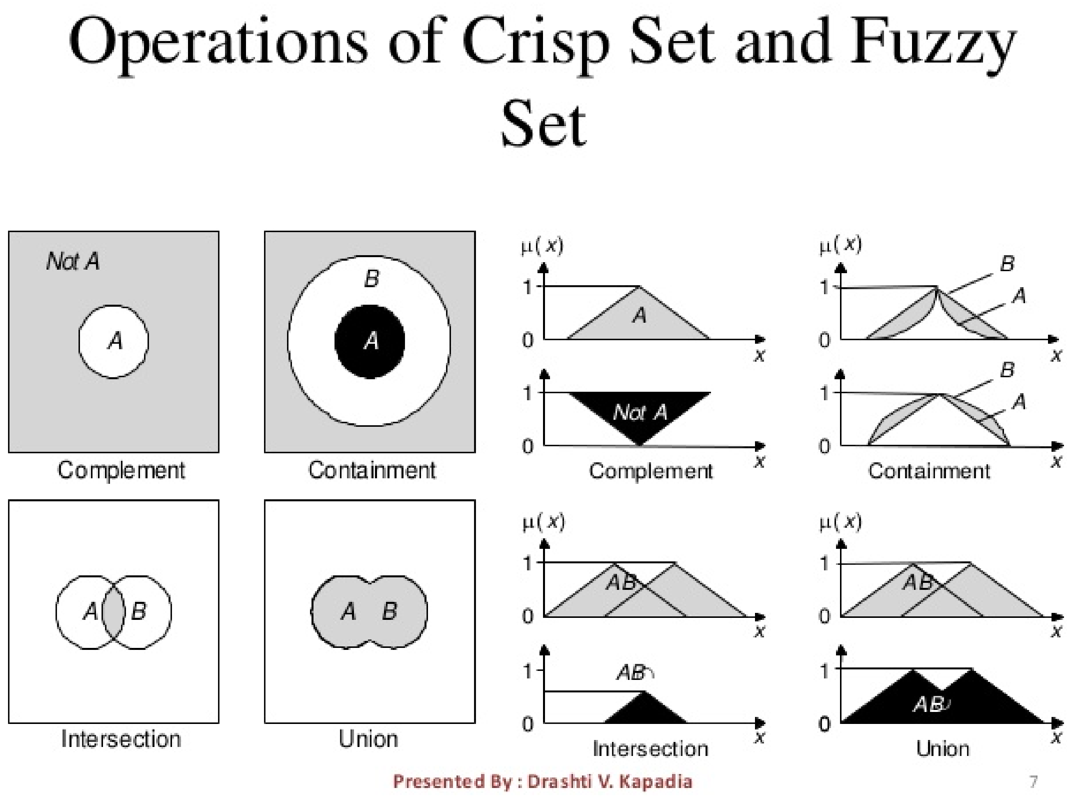
| *Hedge* | *Mathematical Expression* | *Graphical Representation* |
|---|---|---|
| Very very | $[\mu_A(x)]^4$ |  |
| More or less | $\sqrt{\mu_A(x)}$ |  |
| Somewhat | $\sqrt{\mu_A(x)}$ |  |
| Indeed | $2\,[\mu_A(x)]^2$ if $0 \le \mu_A \le 0.5$<br>$1 - 2\,[1 - \mu_A(x)]^2$ if $0.5 < \mu_A \le 1$ |  |

39

4. `Fuzzy Set Operations and Properties` 🐻 ☆☆☆

    Assume a Fuzzy set: $A = \mu_A(x_i)/x_i + \ldots\ldots\ldots\ldots + \mu_A(x_n)/x_n$

    - Operations
        - **Complement**: $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$
        - **Containment**: all elements have the smaller membership value compare to another fuzzy set
        - **Insertion**: $\mu_{A \cap B}(x) = min[\mu_A(x), \mu_B(x)] = \mu_B(x) \cap \mu_B(x)$
        - **Union**: $\mu_{A \cup B}(x) = max[\mu_A(x), \mu_B(x)] = \mu_B(x) \cup \mu_B(x)$




- Properties
  - **Equality**: $\mu_A(x) = \mu_B(x), \forall x \in X$
  - **Inclusion**: $\mu_A(x) \leq \mu_B(x), \forall x \in X$
  - **Cardinality**: $card_A = \mu_A(x_1) + \mu_A(x_2) + \ldots + \mu_A(x_n) = \sum_{\mu_A}(x_i)$
  - **Empty Fuzzy Set**: $\mu_A(x) = 0, \forall x \in X$
  - **Alpha-cut**: $A_\alpha = \{\mu_A(x) \geq \alpha, \forall x \in X\}$
  - **Fuzzy Set Normality** : a fuzzy sunset is normal if there exits at least one element $\mu_A(x) = 1$
  - **Height**: $height(A) = max_x(\mu_A(x))$
  - **Core**: $core(A) = \{x | \mu_A(x) = 1 \ \ and \ \ x \in X\}$
  - **Support**: $supp(A) = \{x | \mu_A(x) > 0 \ \ and \ \ x \in X\}$

5. `Fuzzy Rules`
   - If .... Then.... (FLV are used in fuzzy rules)
   - Example: If height is tall, Then weight is heavy.

6. `Fuzzy Inference System`
   - Step1: Input Fuzzification
   - Step2: Fuzzy Rules Evaluation
   - Step3: Calculate Membership
   - Step4: Activate Fuzzy Rules
   - Step5: Compute Decision Function
   - Step6: Compute Final Decision

   **Specific Example could see slides**

7. `Summary`