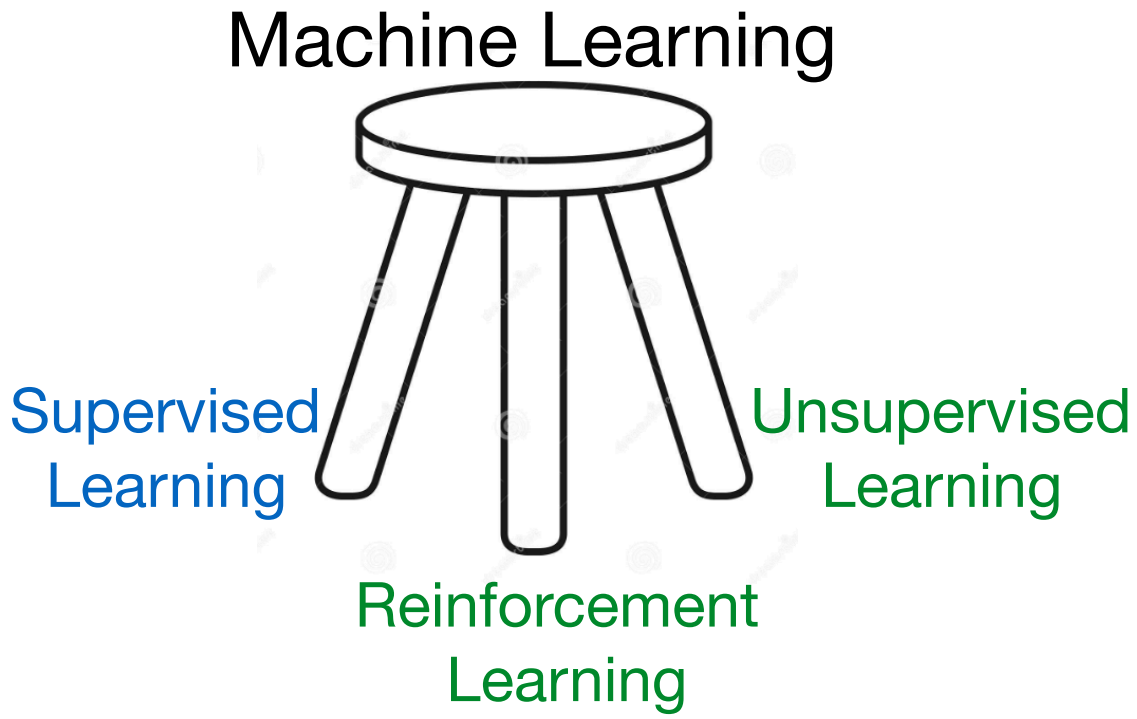
Machine Learning - Beyond Prediction:

Unsupervised Learning

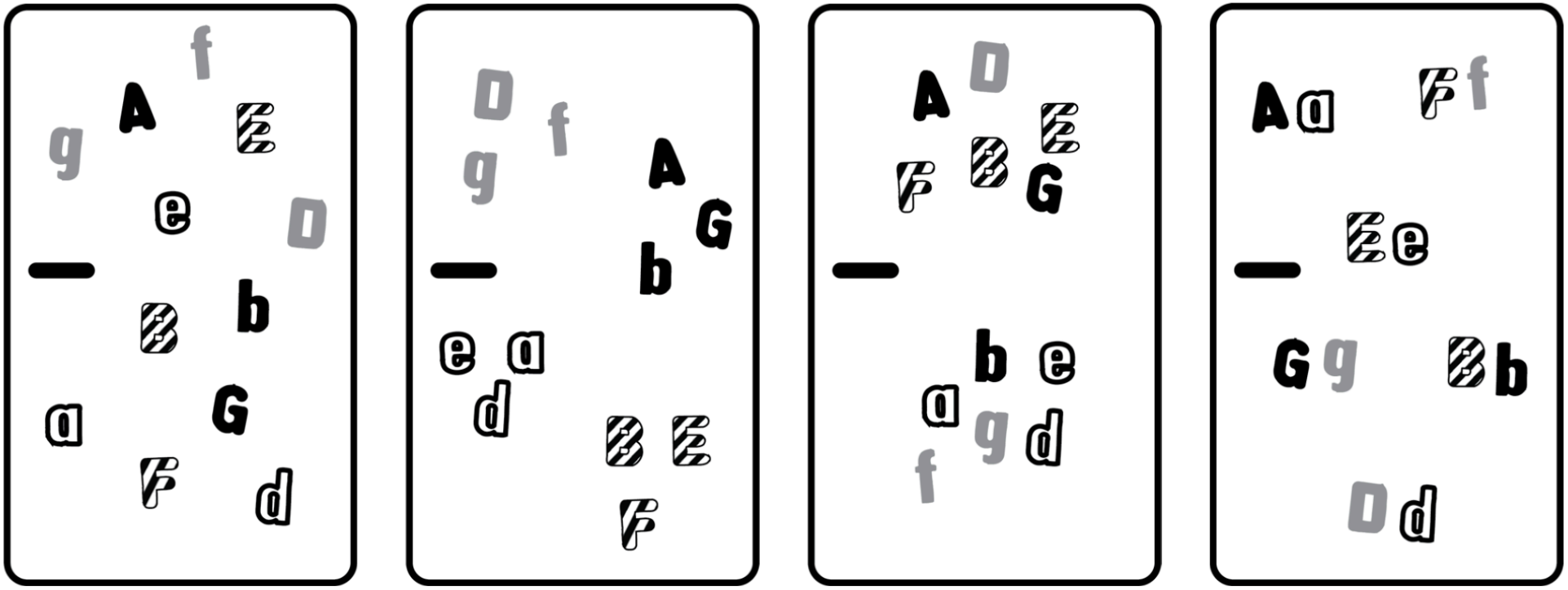
Reinforcement Learning

Sarah Jane Delany

Based on the ML for PDA textbook


Machine Learning
Supervised Learning
Unsupervised Learning
Reinforcement Learning

Unsupervised Learning - Big Idea

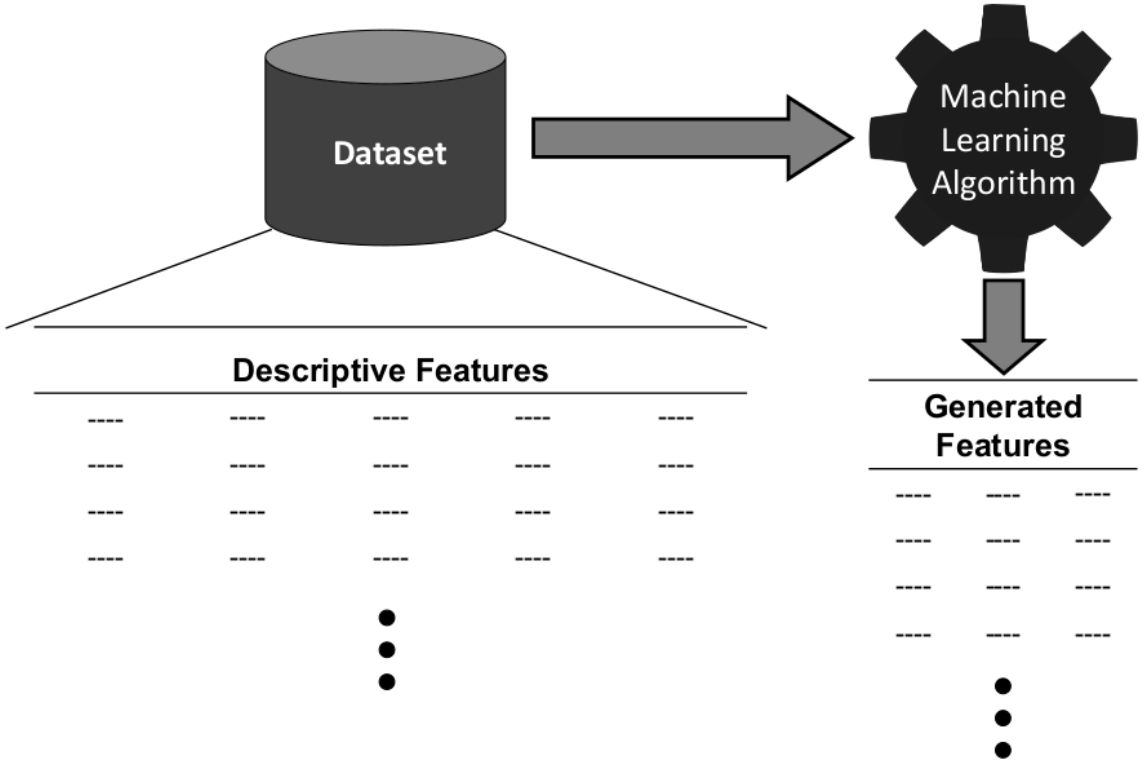


(a) The fridge (b) Abigail (c) Andrew (d) Amalia

- Arrangements of letters on a fridge by three different kids - which is better?
- Difference with supervised learning - no target feature

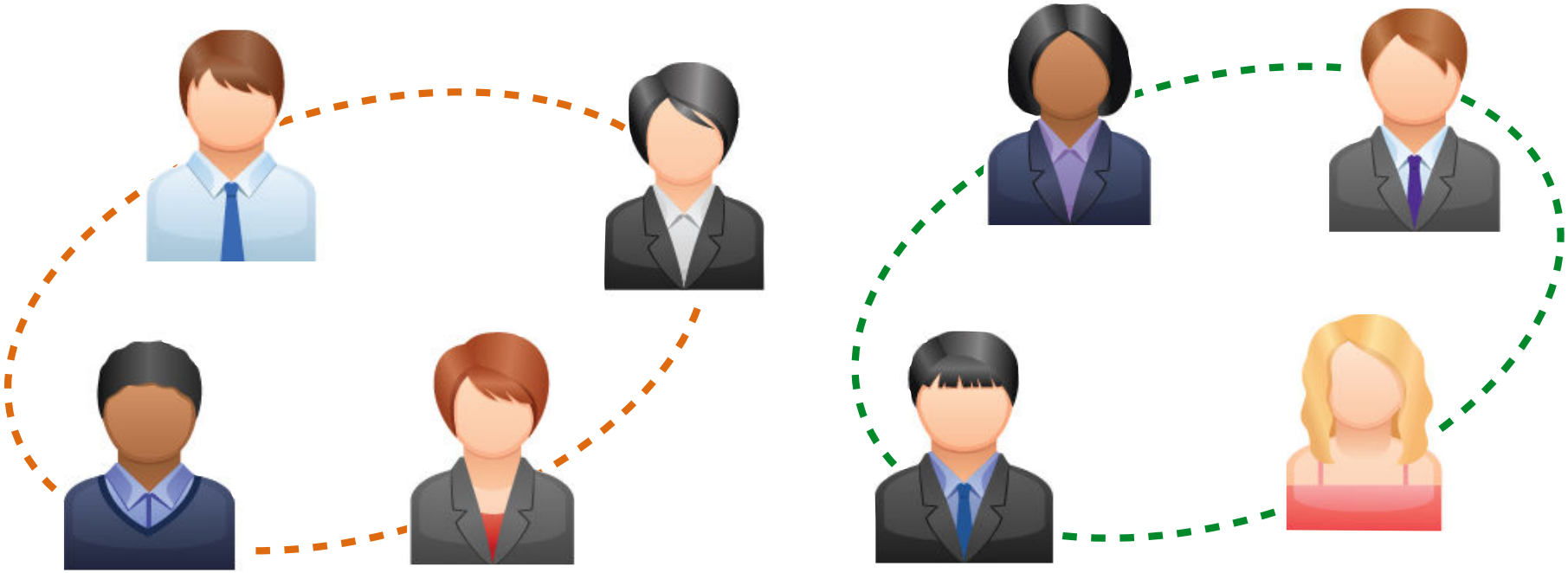
Unsupervised Learning

- Objective: find structure within a set of instances defined by descriptive features alone
- Structure is typically captured in new generated features that can be appended to the original dataset to augment or enrich it



Unsupervised Learning: Applications

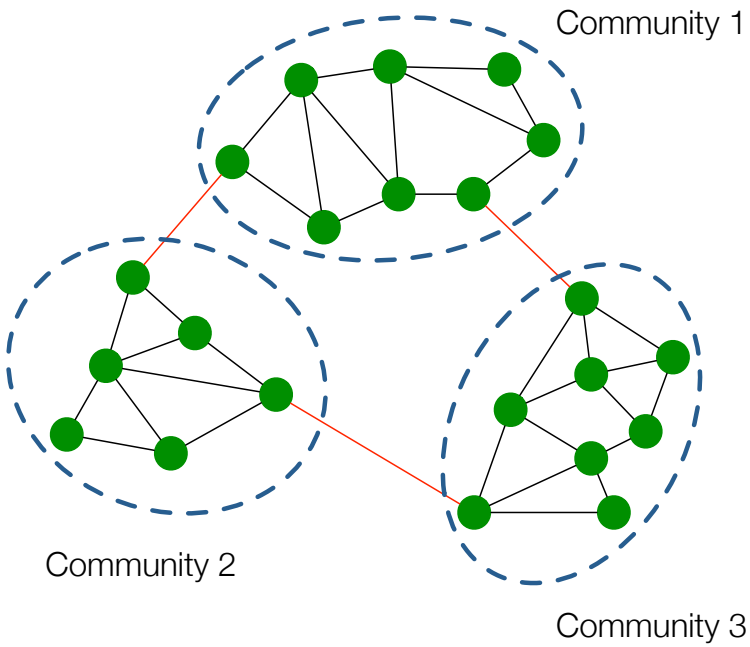
Perform market segmentation by grouping customers into separate clusters, so that customers in the same cluster have similar characteristics.

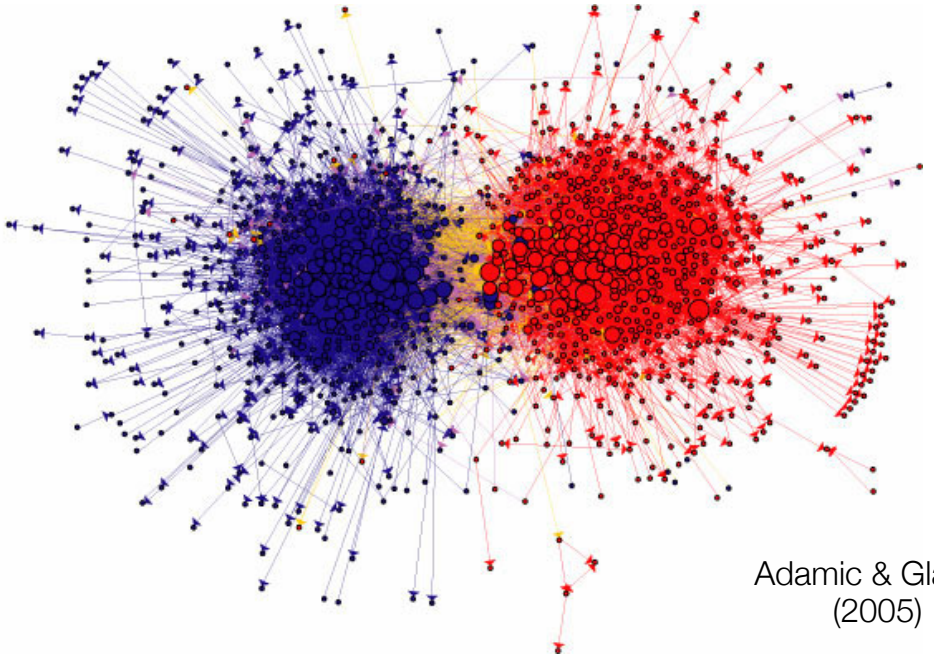


Clustering algorithm output can support targeted marketing and operations for specific sub-groups of customers.

Unsupervised Learning: Applications

Given a social network, apply unsupervised algorithms to identify communities of users who are well-connected to one another, and who are separated from other communities.



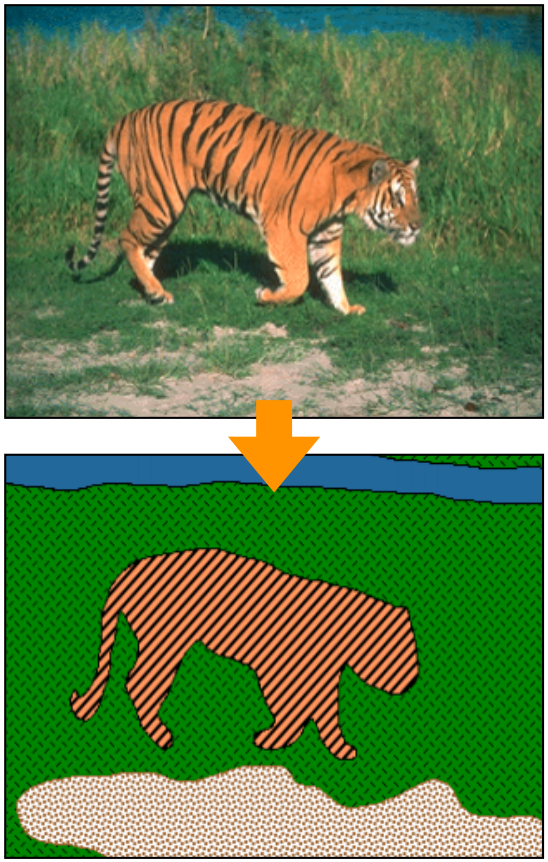


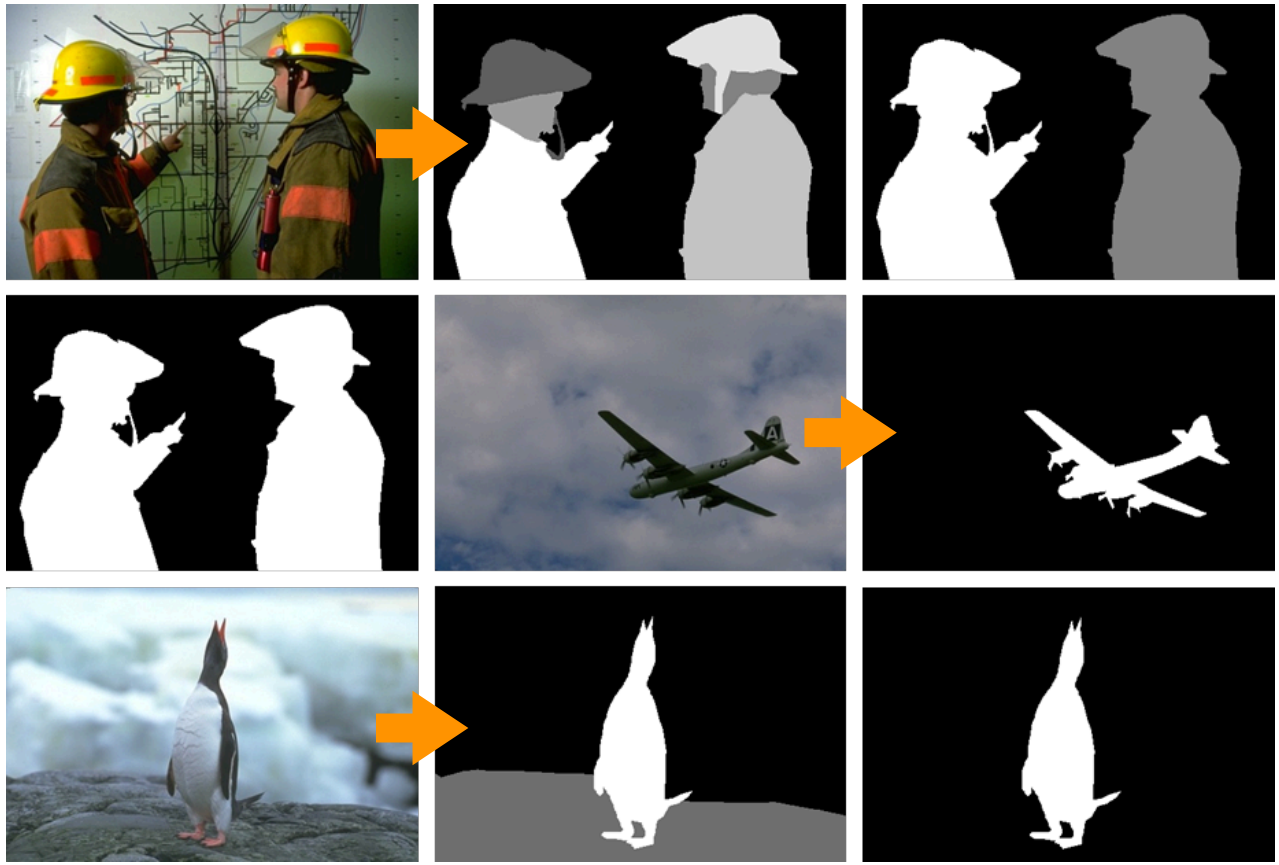
Adamic & Glance (2005)

Users in the network do not need to be manually labeled or annotated in order to apply the community finding algorithm.

Unsupervised Learning: Applications

Image segmentation: Unsupervised task in computer vision that attempts to automatically split an image into regions with similar colour or texture, or both. Aim is to partition the image into its constituent “objects”.



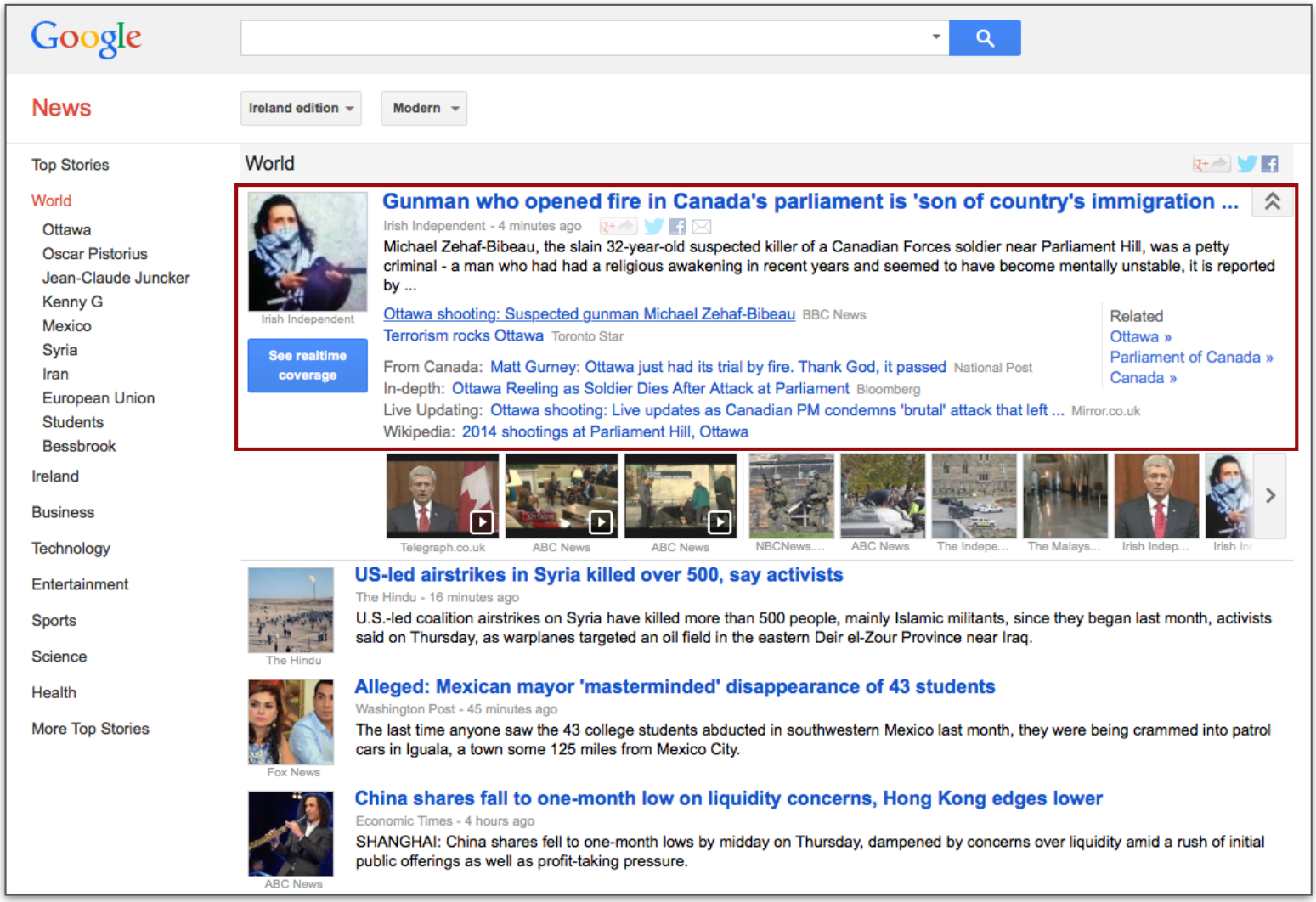


Unsupervised Learning: Applications

Topic modeling: Unsupervised task of discovering the underlying thematic structure in a text corpus - i.e. the key "topics" in the data.

Unsupervised Learning: Applications

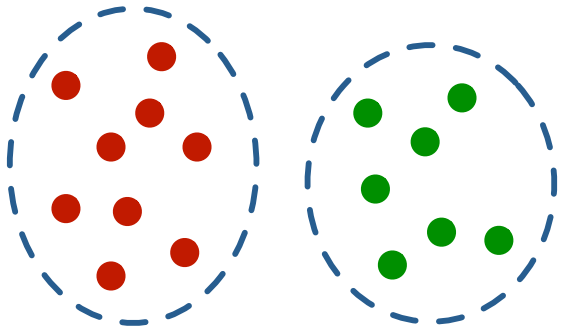
Document clustering: Automatically group related documents together based on similar content (e.g. related articles on Google News).



Clustering

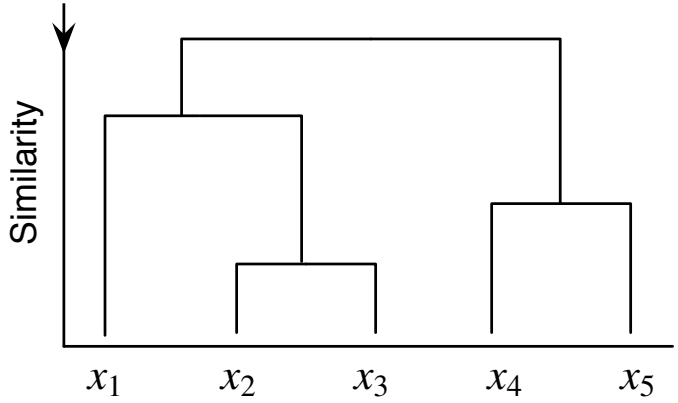
- Clustering partitions a dataset into groups or clusters that are similar to each other
- Algorithms use definitions of similarity/dissimilarity and objective functions for determining a “good” cluster

Partitional Algorithms



Build a “flat” clustering of the data all at once

Hierarchical Algorithms



Gradually build a nested tree structure of clusters

Partitional Clustering

- Attempt to directly decompose a data set into a “flat” grouping consisting of a number of disjoint (non-overlapping) clusters.
- Usually pre-specify number of clusters *k*, although some methods adaptively add/remove clusters.
- Start with an initial set of *k* clusters, often chosen at random.
- Use a heuristic to find the best local solution for an objective function, identified by iteratively improving the initial solution.
- *k*-means clustering is the most well-known approach to partitional clustering

  \+ simple, computationally efficient, quite effective

  \- non deterministic

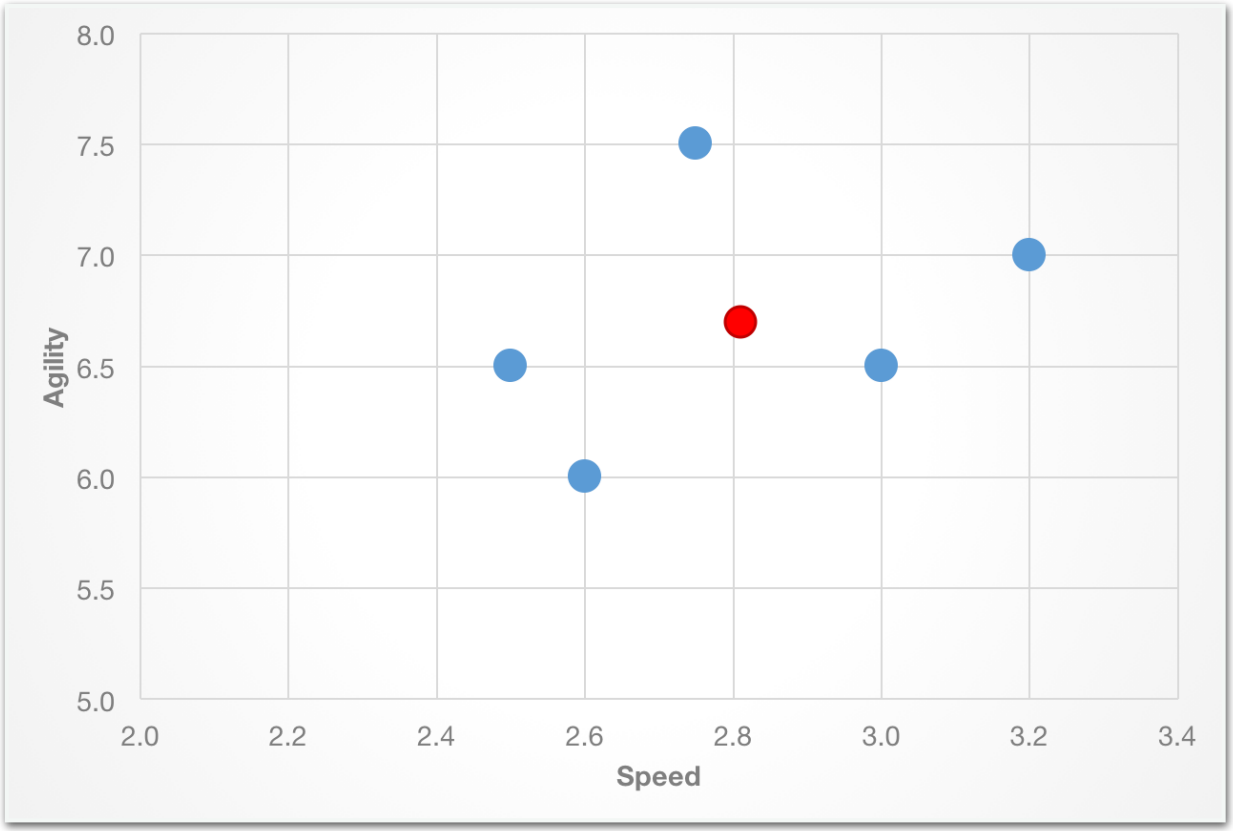
k-Means Clustering

- Centroid: The mean vector of all items assigned to a given cluster (i.e. the mean of their feature vectors).

Athlete	*Speed*	*Agility*
1	2.6	6.0
2	3.0	6.5
3	2.5	6.5
4	3.2	7.0
5	2.8	7.5
Centroid	2.82	6.7

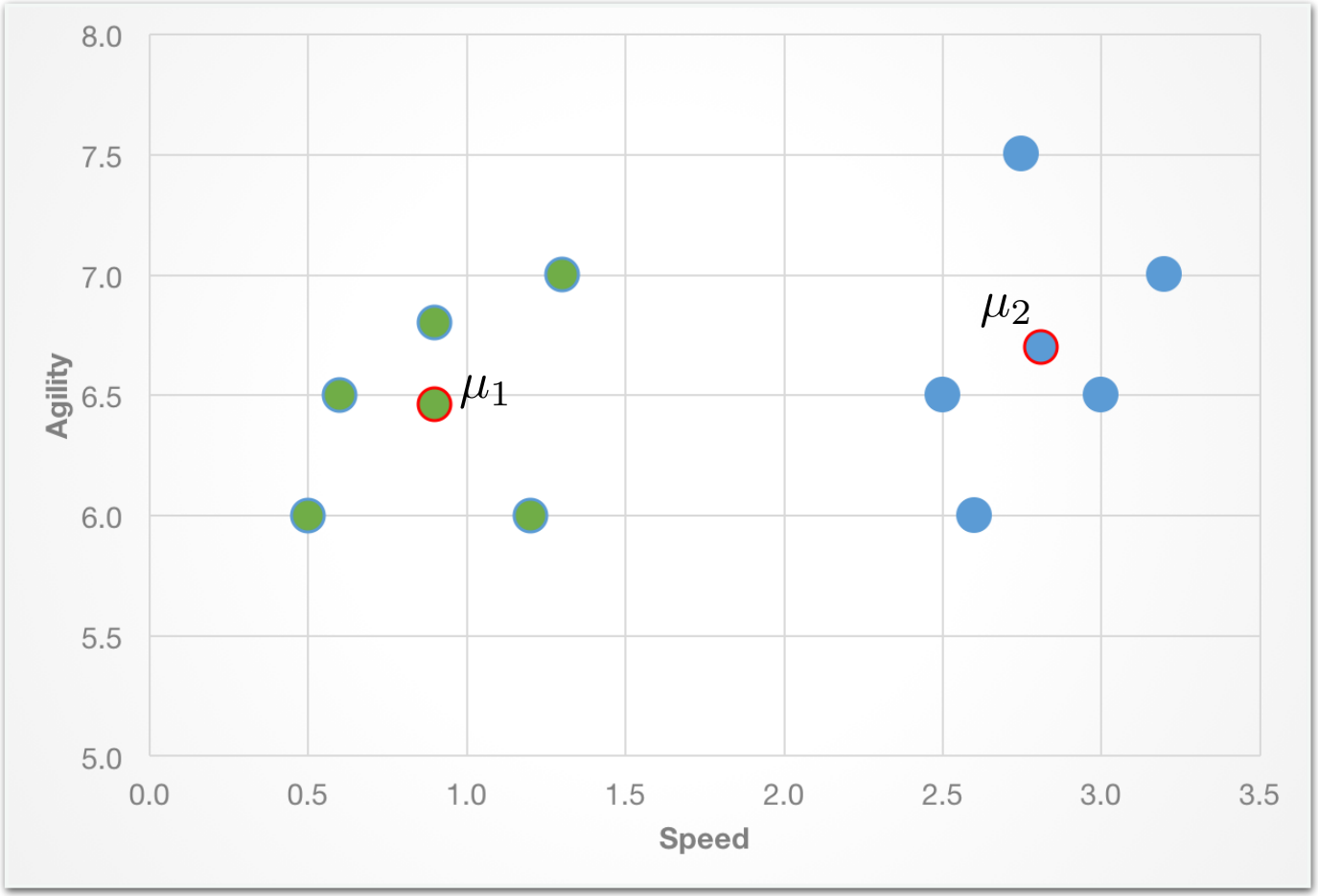
$(2.6 + 3.0 + 2.5 + 3.2 + 2.8)/5 = 2.82$

$(6.0 + 6.5 + 6.5 + 7.0 + 7.5)/5 = 6.7$



k-Means Clustering

- Each of the *k* clusters in a clustering can be represented by its own centroid μ_i
- Example of two clusters, with centroids shown:



k-Means Clustering

- For dataset *D* with instances, $d_1 \ldots d_n$ goal is to minimise the distances between instances and their nearest centroid, i.e. minimising the *sum-of-squared error* (SSE):

$$SSE = \sum_{c=1}^{k} \sum_{d_i \in C_c} Dist(d_i, \mu_c)^2 \quad \text{where} \quad \mu_c = \frac{\sum_{d_i \in C_c} d_i}{|C_c|}$$

- Distance is normally measured using Euclidean distance

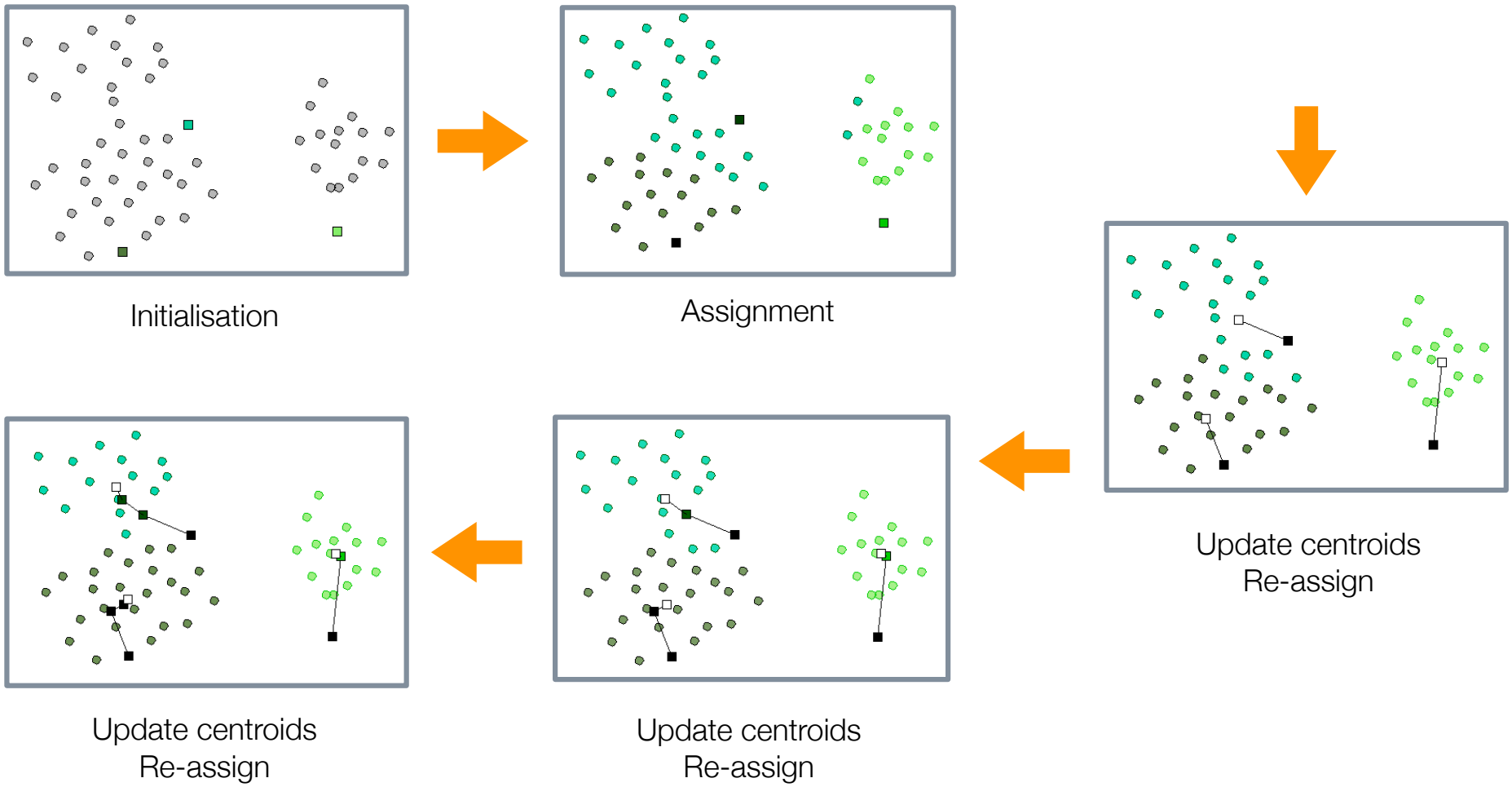
$$Dist(d_i, \mu_c) = \sqrt{\sum_{j=1}^{m} (d_{ij} - \mu_{cj})^2}$$

sum of squared difference over all *m* feature values

- *k*-Means tries to reduce SSE via a two step iterative process:
  1) Reassign items to their nearest cluster centroid
  2) Update the centroids based on the new assignments
- Repeatedly apply these two steps until the algorithm converges (i.e. no cluster reassignments happen)

k-Means Clustering

Simple example of several iterations of *k*-Means for *k*=3…



k-Means Algorithm

Inputs:

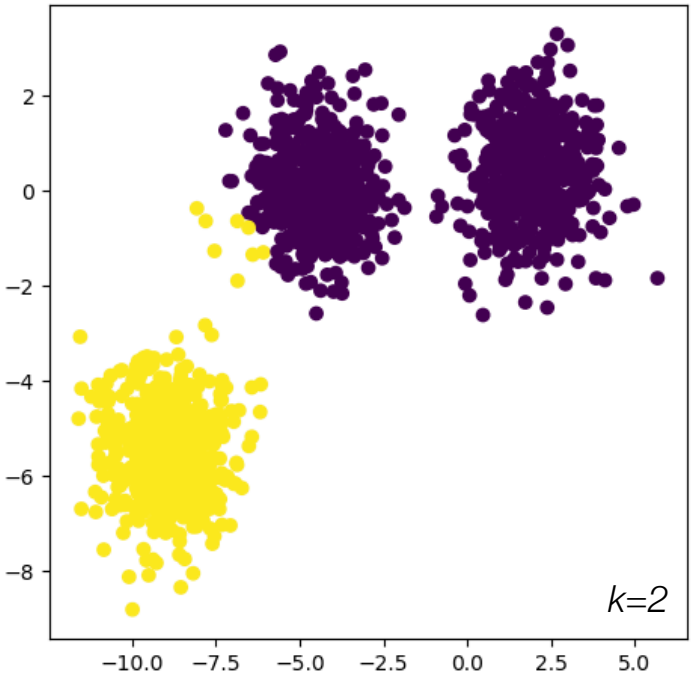
- Data: Set of unlabelled items
- *k*: User-specified target number of clusters
- Maximum number of iterations to run

Algorithm Steps:

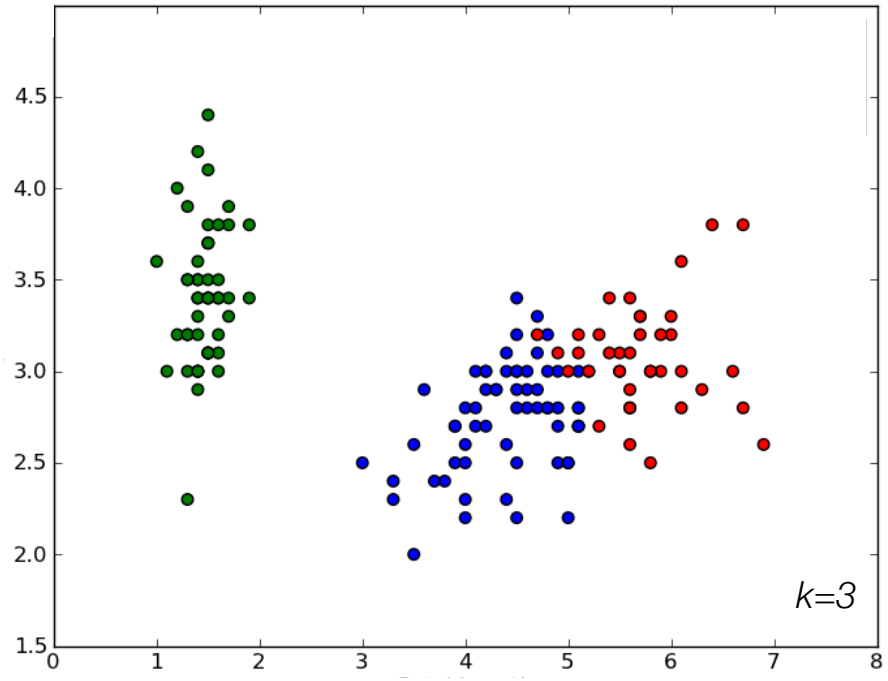
1. *Initialisation*: Select *k* initial cluster centroids (e.g. at random)
2. *Assignment step*: Assign every item to its nearest cluster centroid (e.g. using Euclidean distance).
3. *Update step*: Recompute the centroids of the clusters based on the new cluster assignments, where a centroid is the mean point of its cluster.
4. Go back to Step 2, until when no reassignments occur (or until a maximum number of iterations is reached).

k-Means: How many clusters?

- Key input parameter *k* - how many clusters?
- *k* too low → “smearing” of clusters that should not be merged.
- *k* too high → “over-clustering” of the data into many small, similar clusters.



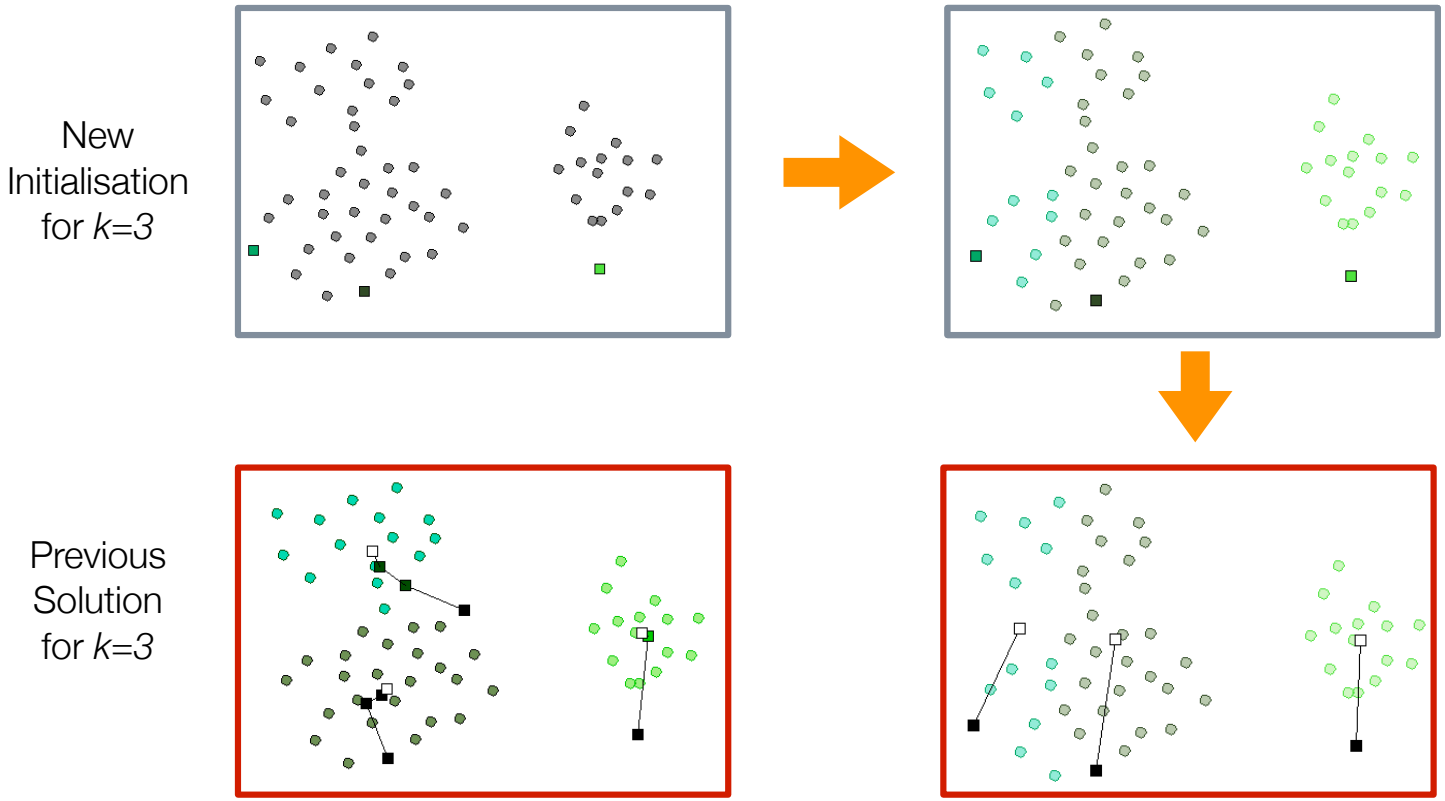
Too few clusters?



Too many clusters?

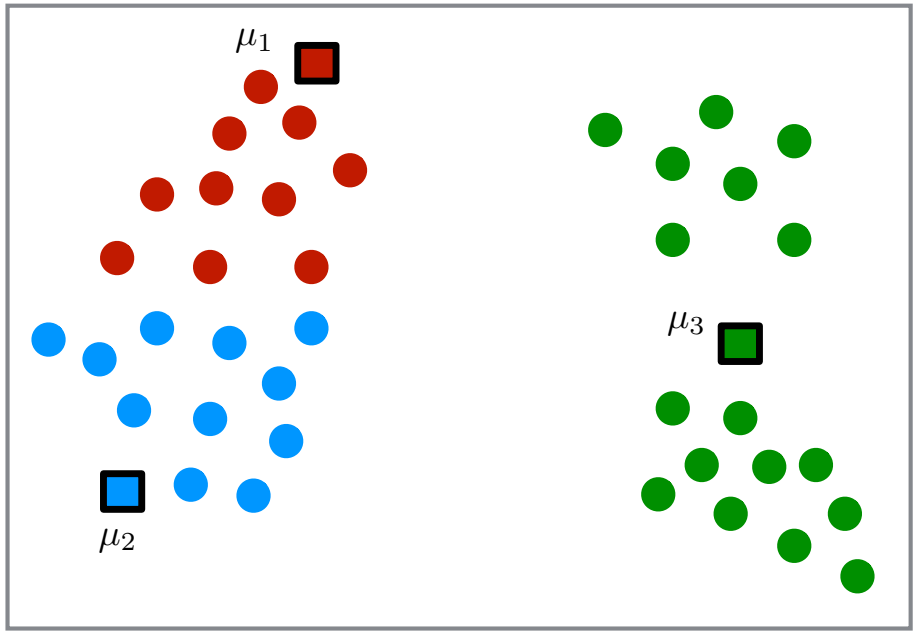
Cluster Initialisation

- Results produced by *k*-Means are often highly dependent on the initialisation.
- Different starting positions can lead to different local minima - i.e. different clusterings of the same data ➡ non-deterministic

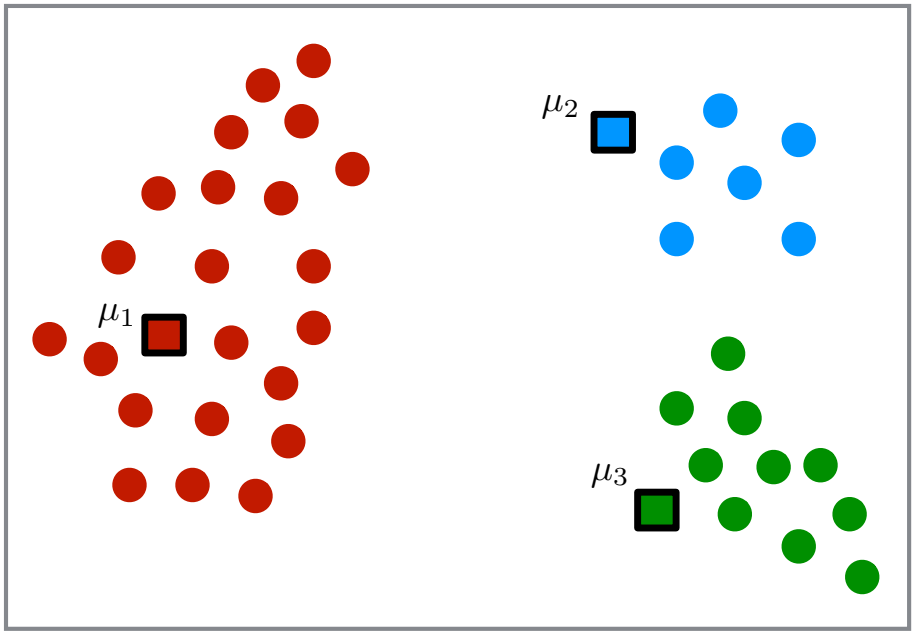


Cluster Initialisation

A poor choice of initial centroids will often lead to a poor clustering that is not useful. A better initialisation will lead to different clusters.



Initialisation 1



Initialisation 2

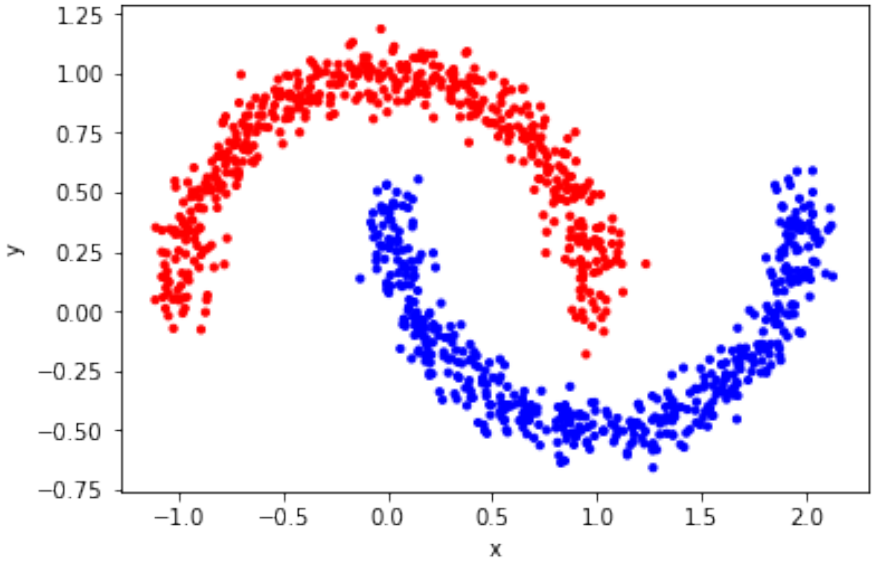
➡ Common strategy: Run the algorithm multiple times, select the solution(s) that scores well according to some validation measure.

Limitations of *k*-means

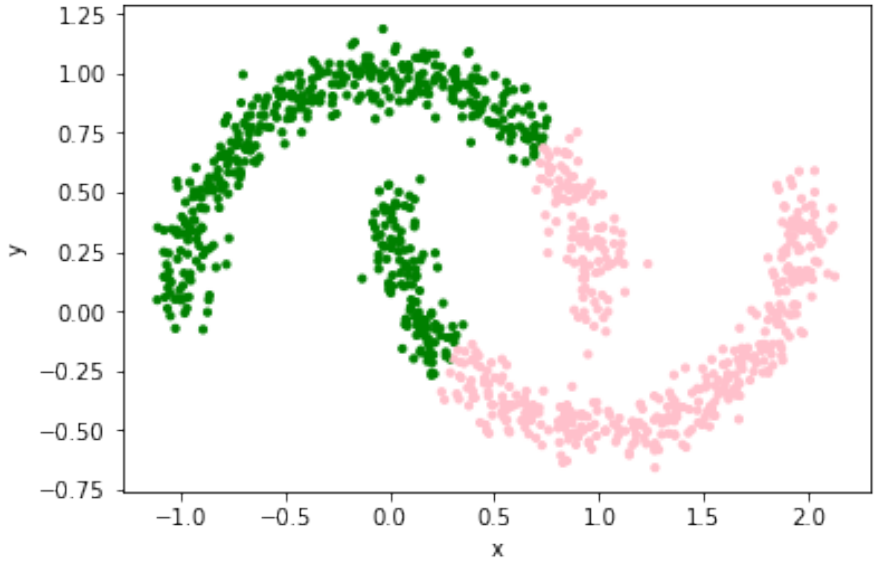
- **Advantages:**
  - Fast, easy to implement.
  - “Good enough” in a wide variety of tasks and domains.
- **Disadvantages:**
  - Must pre-specify number of clusters *k*.
  - Highly sensitive to choice of initial clusters.
  - Assumes that each cluster is spherical in shape and data examples are largely concentrated near its centroid.
  - Traditional objective can give undue influence to outliers.
  - Iterative process can lead to empty clusters, particularly for higher values of *k*.

Limitations of *k*-means

Example: *k*-Means assumes that clusters are spherical in shape and data examples are largely concentrated near its centroid.



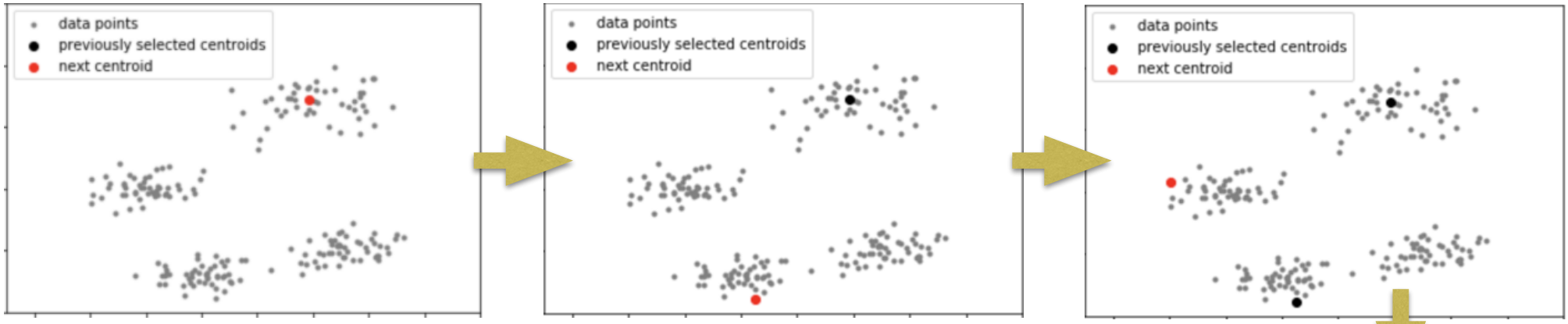
Original “correct” groups in the data



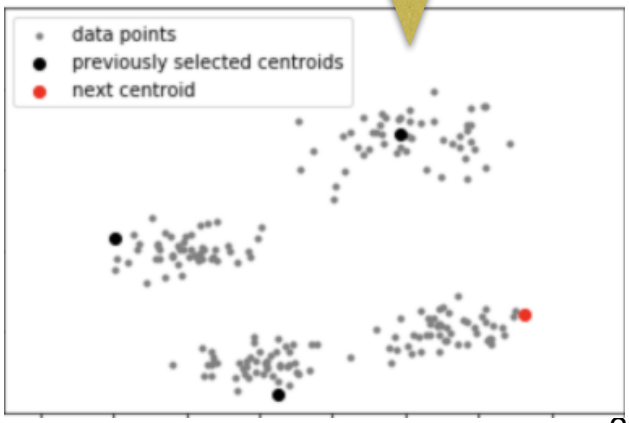
Clusters identified by *k*-means for *k=2*

k-means++

- *k*-means++ is a variation on *k*-means that includes an initialisation strategy that selects initial centroids less likely to lead to sub-optimal clusters
- Instances far away from the already selected centroids are more likely to be selected than those close to the already selected centroids



- Added advantage is that it converges faster
- Still non-deterministic…



k-means++

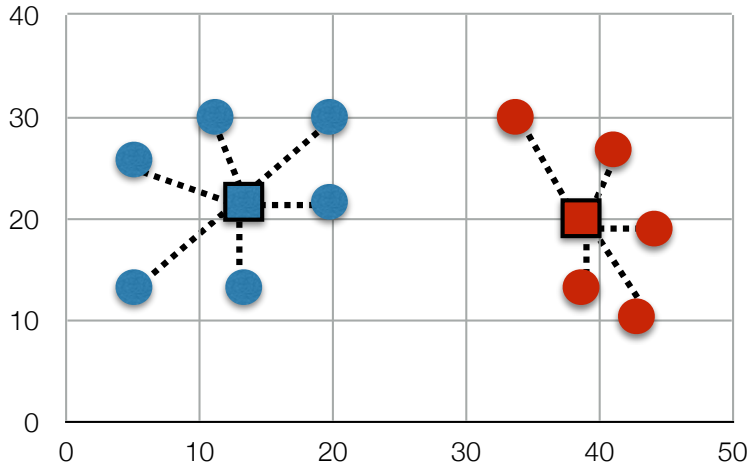
Algorithm:

1. Randomly select first centroid from the dataset $\{d_1 \ldots d_n\}$
2. For each instance d_i in the dataset calculate its distance from its nearest centroid
3. Assign a weight w_i to each instance d_i which is proportional to the square of this distance $$w_i = \frac{Dist(d_i)^2}{\sum_{p=1}^{n} Dist(d_p)^2}$$
4. Select the next centroid randomly from the dataset with each instance weighted by w_i
5. Repeat from 2 until *k* centroids are selected

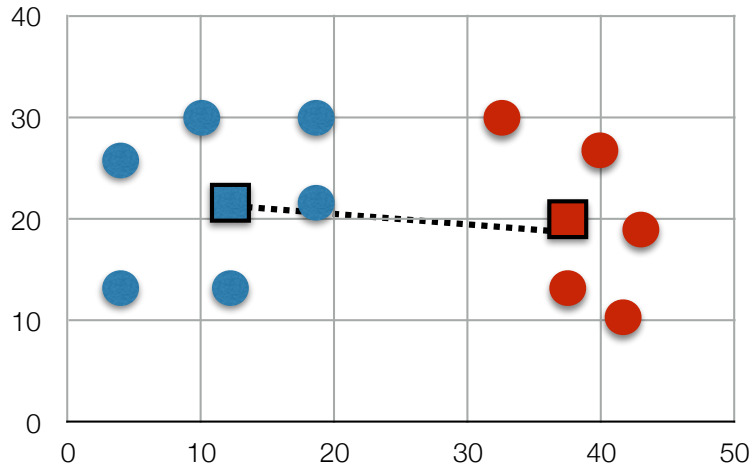
Proceed with *k*-means using selected centroids as the initial centroids

Evaluating Clusters

- Cluster validation: Measures for automatically producing a quantitative evaluation of the quality of a clustering.
- Common motivation - “good” clusters have the property that cluster members are close to each other and far from members of other clusters
- Good clustering minimises intra-cluster distances (aka inertia) and maximises inter-cluster distances



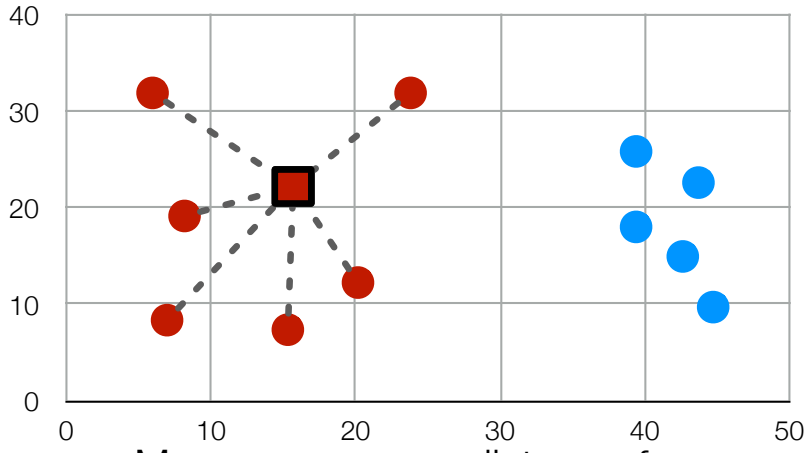
Intra-cluster distance



Inter-cluster distance

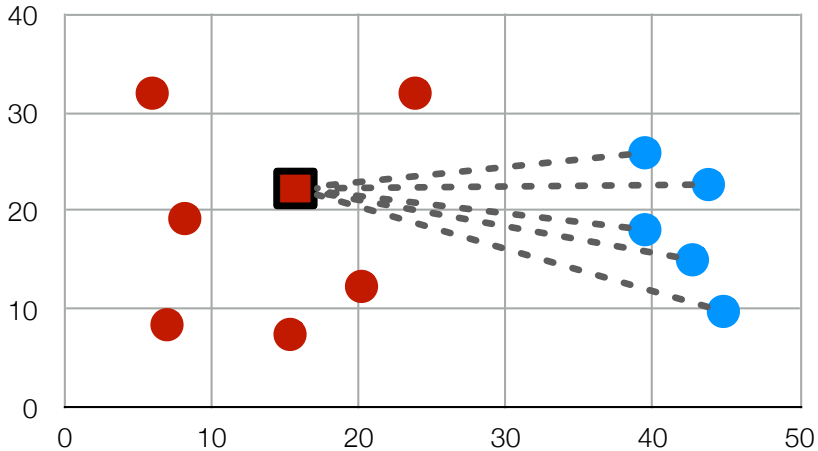
Silhouette measure

- Validation measure which quantifies degree to which each item belongs in its assigned cluster, relative to the other clusters.



Measure average distance from instance i to all other items in same cluster C_h.

$$a_i = \frac{1}{|C_h| - 1} \sum_{j \in C_h, j \neq i} d(i,j)$$



Measure average distance from instance i to all other items in nearest competing cluster C_l.

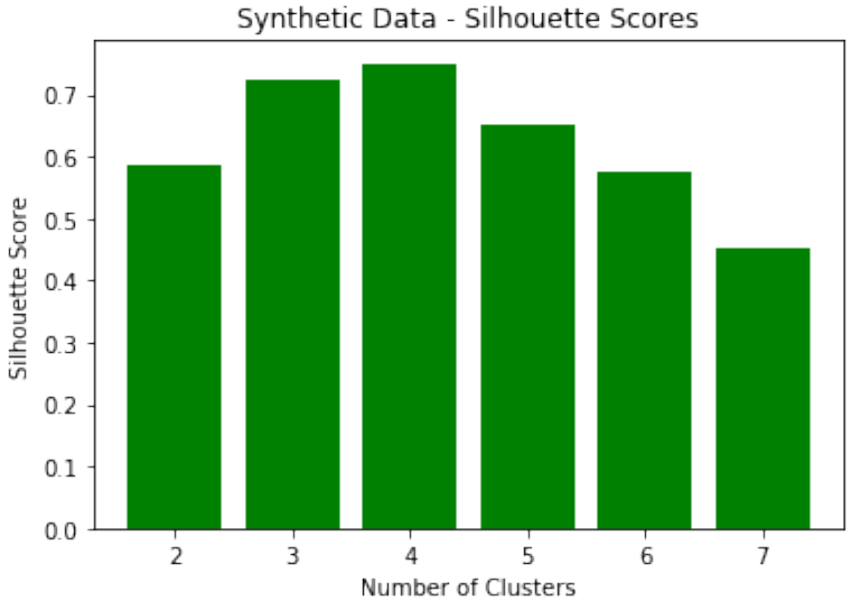
$$b_i = \frac{1}{|C_l|} \sum_{j \in C_l} d(i,j)$$

- Silhouette width for an item x_i is given by s_i Values are in the range [-1,1], a larger value is better.

$$s_i = \frac{b_i - a_i}{\max\{a_i, b_i\}}$$

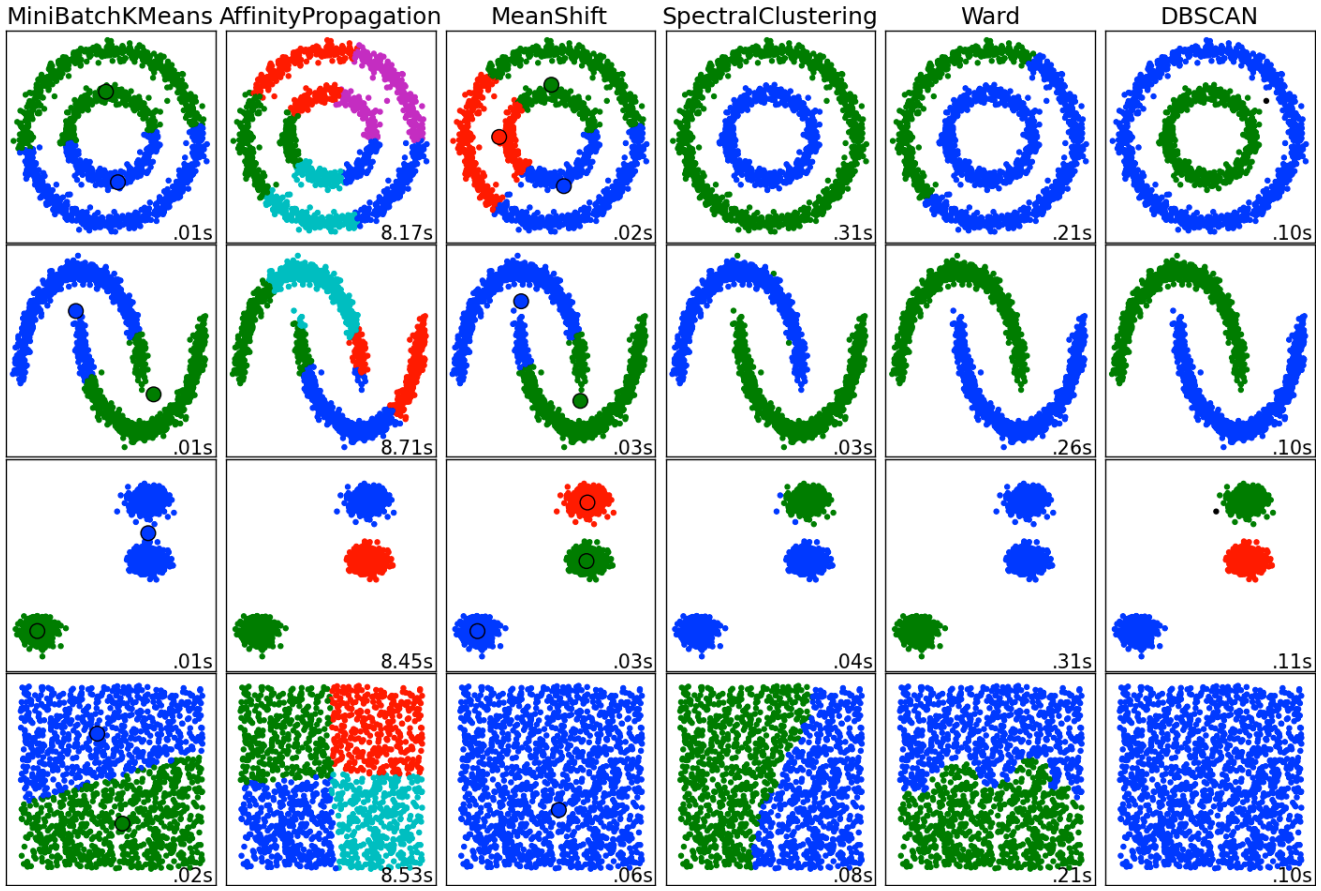
k-Means: How many clusters?

- Cluster validation is often applied for parameter selection - e.g. select an appropriate value *k* for the *k*-Means algorithm.
- **Typical Strategy:**
  1. Apply *k*-Means for each value from k_{min} to k_{max}.
  2. Calculate score for each clustering using a cluster validation measure.
  3. Examine plot of scores to identify a peak for the best value for *k*.



Clustering

There are usually many different ways to cluster the same data. Clustering algorithms differ significantly in their definition of what constitutes a cluster and the approach used to find them.



http://scikit-learn.org/stable/modules/clustering.html

Reinforcement Learning

Most common application: learning to control the behaviours of autonomous systems,
e.g. training robots to perform tasks, automating players to play games

Key Difference: it relies less on using a dataset to drive learning and more on the ability to repeatedly attempt tasks in an environment

Big Idea

- Kate is a young venture scout in training for her pioneering badge
- A challenge is to learn to cross a stream using stepping stones while wearing an electronic blindfold
- Goal is to cross in the fewest steps without getting wet
- Before taking a step, the blindfold is made transparent for 0.5 secs to give the scout a quick view of their environment so they can make a decision about which direction and how far to step

Characteristics:

- Task to achieve which is repeated many times
- Positive feedback
- Negative feedback

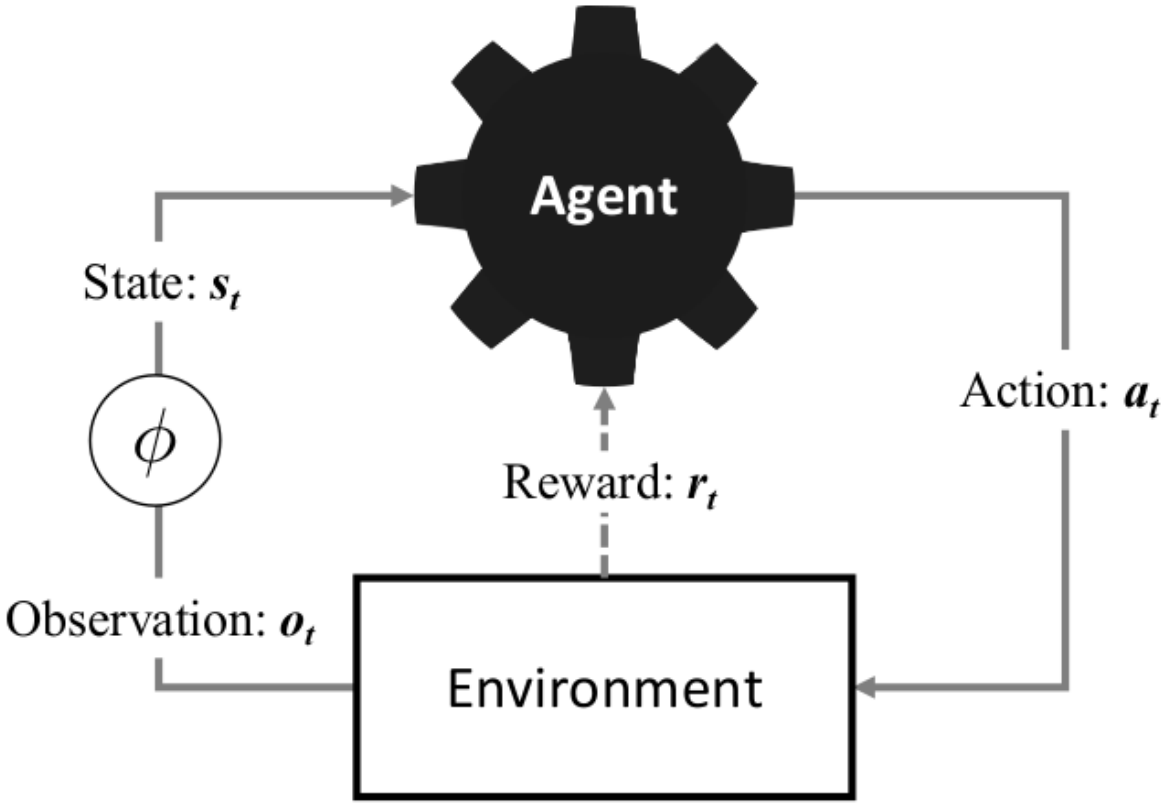
Intelligent Agents

- Agent attempts to complete a task within an environment
- Goal is to complete the task as successfully as possible
- Each attempt is referred to as an episode
- At a point in time *t*
  - the agent observes the current state of the environment o_t
  - selects an action a_t
  - takes this action and receives immediate feedback, reward r_t from the environment about whether this is a good or bad action to take
- An episode is a series of discrete steps through time steps $t = 1, \ldots, e$

$$(o_1, a_1, r_1), (o_2, a_2, r_2), (o_3, a_3, r_3), \ldots, (o_e, a_e, r_e)$$

Reinforcement Learning Cycle

- the state at time-step t, s_t contains info about the current environment, the preceding time-steps and agent



- the state generating function ϕ converts the observations to a state
- designing good state representations is key to reinforcement learning

$$(s_1, a_1, r_1), (s_2, a_2, r_2), (s_3, a_3, r_3), \ldots, (s_e, a_e, r_e)$$

Fundamentals

- Only goal of an agent is to maximise cumulative reward

$$G = r_t + r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_e$$

- The key decision making component of a RL agent is a policy which is a mapping from states to actions $a_t = \pi(s_t)$
  - it tells the agent what action to take when in a given state
  - think of it as a simple lookup table….
  - RL can be considered as how to ‘learn’ this table directly

- Policies can be encoded as a rule used to choose an action from those available in a particular state
  - e.g. greedy action selection policy

Fundamentals

- The action-value function returns the cumulative reward that an agent can expect to earn if it takes action a_t in state s_t and continues to select actions using policy π to the end of the episode
  - i.e. the expected return of pursuing action a_t in state s_t

$$Q_\pi(s_t, a_t) = E_\pi[r_t + r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_e \mid s_t, a_t]$$

- discounted return means paying more attention to the immediate reward rather than rewards expected after actions in the future

$$Q_\pi(s_t, a_t) = E_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots + \gamma^{3-t} r_e \mid s_t, a_t]$$

discount rate [0,1]

Summary

- In RL an agent attempts to achieve a goal by taking a sequence of actions to move between states, receiving a reward after completing each action which indicates whether the outcome was positive or negative
- The goal is measured only by the cumulative rewards received from each action taken in pursuit of the goal
- The agent uses a policy to choose which action to take in a given state
- Policies rely on the expected return of taking an action in a given state which is measured by the action-value function

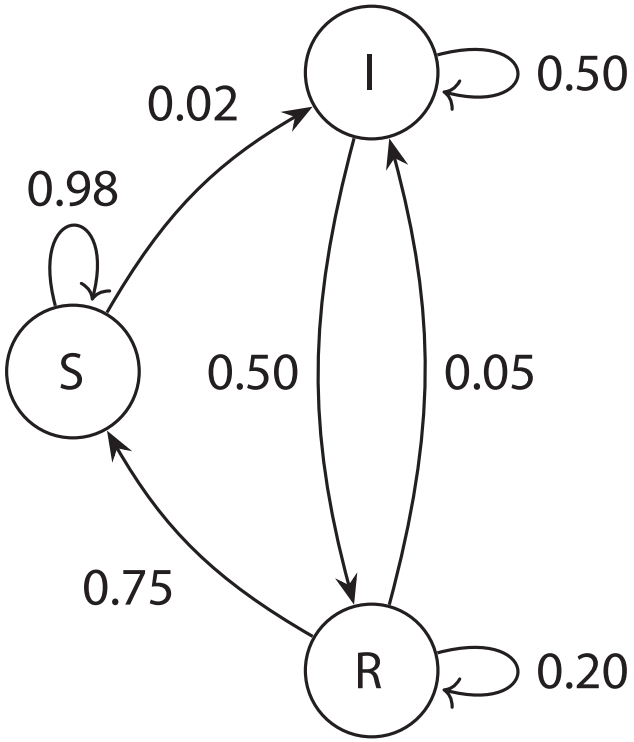
Summary

- In RL an agent attempts to achieve a goal by taking a sequence of actions to move between states, receiving a reward after completing each action which indicates whether the outcome was positive or negative
- The goal is measured only by the cumulative rewards received from each action taken in pursuit of the goal
- The agent uses a policy to choose which action to take in a given state
- Policies rely on the expected return of taking an action in a given state which is measured by the action-value function

So, where's the learning?

Markov Process

- A Markov process models a discrete random process that transitions through a set of states



$$P(\mathrm{S} \rightarrow \mathrm{I}) = P(S_{t+1} = \mathrm{I} \mid S_t = \mathrm{S})$$

$$\mathcal{P} = \begin{array}{c} \\ \mathrm{S} \\ \mathrm{I} \\ \mathrm{R} \end{array} \begin{array}{c} \begin{array}{ccc} \mathrm{S} & \mathrm{I} & \mathrm{R} \end{array} \\ \begin{bmatrix} 0.98 & 0.02 & 0.00 \\ 0.00 & 0.50 & 0.50 \\ 0.75 & 0.05 & 0.20 \end{bmatrix} \end{array}$$

transition matrix

- An simple SIR model: a Markov process to model the evolution of an infectious disease in individuals during a epidemic

Markov Decision Process (MDP)

$$P(s_1 \to s_2) = P(S_{t+1} = s_2 \mid S_t = s_1)$$ Markov Assumption

- MDP extends the Markov process by adding decision making and rewards

$$P(s_1 \xrightarrow{a} s_2) = P(S_{t+1} = s_2 \mid S_t = s_1, A_t = a)$$

$$R(s_1 \xrightarrow{a} s_2) = E(r_t \mid S_t = s_1, S_{t+1} = s_2, A_t = a)$$

MDP example

- **TwentyTwos** a simplified version of Blackjack
- Aim: collect cards that have a higher value than the dealer but not exceeding 22
  - Cards are worth their value, picture cards are 10, Ace is 11
- Play:
  - Two cards dealt to player & dealer, one face up
  - Player can *twist* repeatedly or *stick,* but can go bust
  - Dealer then turns over until their total is >= 17
- Outcome:
  - Player wins €1 if player total > dealer total or dealer goes bust
  - Player and dealer can tie if both have same total <22
  - Otherwise player loses €1
- Special case - player wins €2 if dealt two Aces

Iter	Player Hand		Dealer Hand		Action	Reward
1	2♥ 7♣	(9)	8♥	(8)	*Twist*	0
2	2♥ 7♣ *K*♣	(19)	8♥	(8)	*Stick*	+1
3	2♥ 7♣ *K*♣	(19)	8♥ *Q*♦	(18)		
1	4♠ *A*♥	(15)	*Q*♥	(10)	*Twist*	-1
2	4♠ *A*♥ 9♣	(24)	*Q*♥	(10)		
1	2♦ 4♦	(6)	3♥	(3)	*Twist*	0
2	2♦ 4♦ 3♥	(9)	3♥	(3)	*Twist*	0
3	2♦ 4♦ 3♥ 6♣	(15)	3♥	(3)	*Twist*	0
4	2♦ 4♦ 3♥ 6♣ 6♦	(21)	3♥	(3)	*Stick*	0
5	2♦ 4♦ 3♥ 6♣ 6♦	(21)	3♥ 7♥ *A*♠	(21)		
1	*Q*♦ *J*♣	(20)	*A*♥	(11)	*Stick*	+1
2	*Q*♦ *J*♣	(20)	*A*♣ 5♣ *Q*♠	(26)		
1	*A*♦ *A*♥	(22)	2♥	(2)	*Stick*	+2
2	*A*♦ *A*♥	(22)	2♥	(2)		

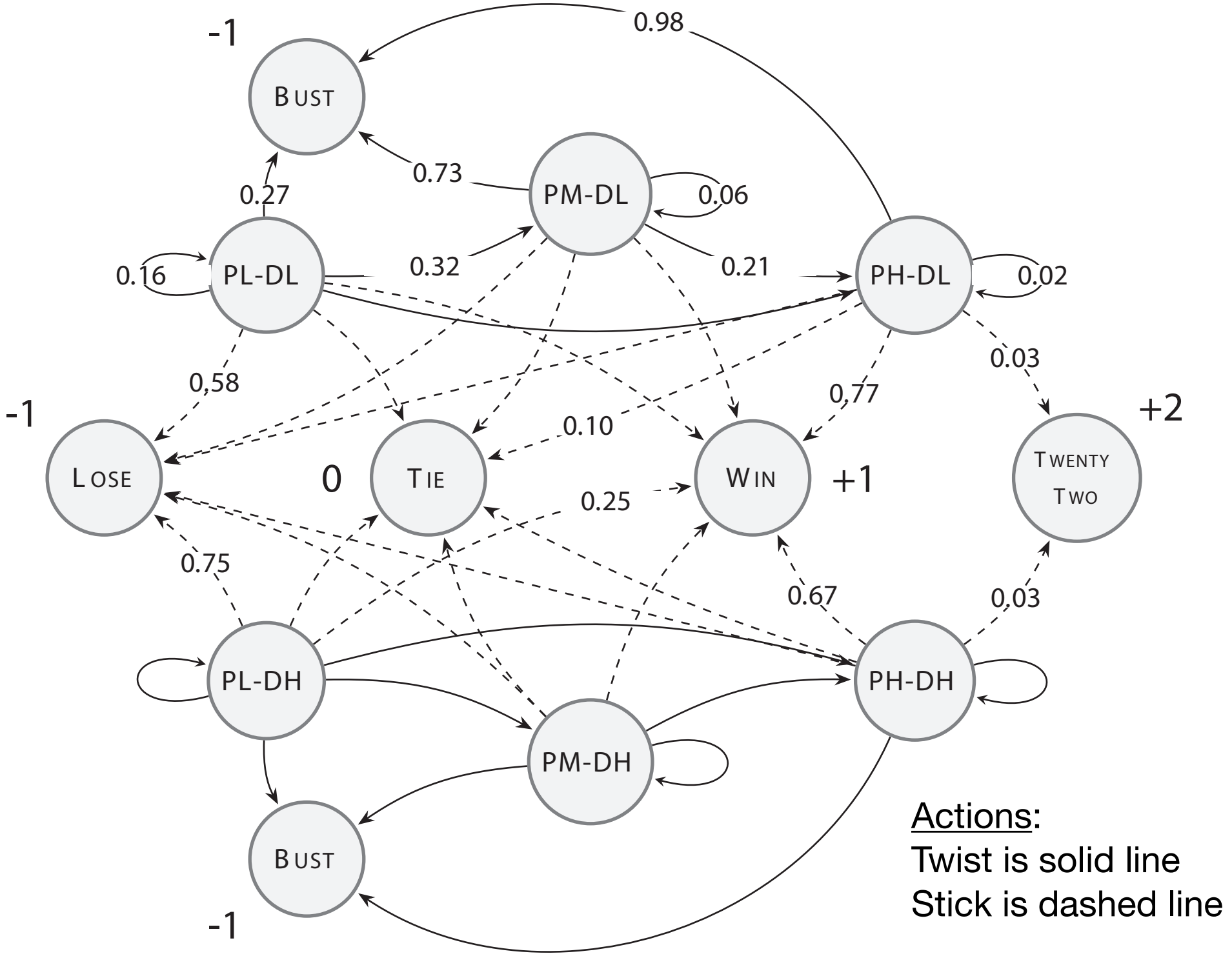
Iter	Player Hand		Dealer Hand		Action	Reward
1	2♥ 7♣	(9)	8♥	(8)	*Twist*	0
2	2♥ 7♣ *K*♣	(19)	8♥	(8)	*Stick*	+1
3	2♥ 7♣ *K*♣	(19)	8♥ *Q*♦	(18)		
1	4♠ *A*♥	(15)	*Q*♥	(10)	*Twist*	-1
2	4♠ *A*♥ 9♣	(24)	*Q*♥	(10)		
1	2♦ 4♦	(6)	3♥	(3)	*Twist*	0
2	2♦ 4♦ 3♥	(9)	3♥	(3)	*Twist*	0
3	2♦ 4♦ 3♥ 6♣	(15)	3♥	(3)	*Twist*	0
4	2♦ 4♦ 3♥ 6♣ 6♦	(21)	3♥	(3)	*Stick*	0
5	2♦ 4♦ 3♥ 6♣ 6♦	(21)	3♥ 7♥ *A*♠	(21)		
1	*Q*♦ *J*♣	(20)	*A*♥	(11)	*Stick*	+1
2	*Q*♦ *J*♣	(20)	*A*♣ 5♣ *Q*♠	(26)		
1	*A*♦ *A*♥	(22)	2♥	(2)	*Stick*	+2
2	*A*♦ *A*♥	(22)	2♥	(2)		

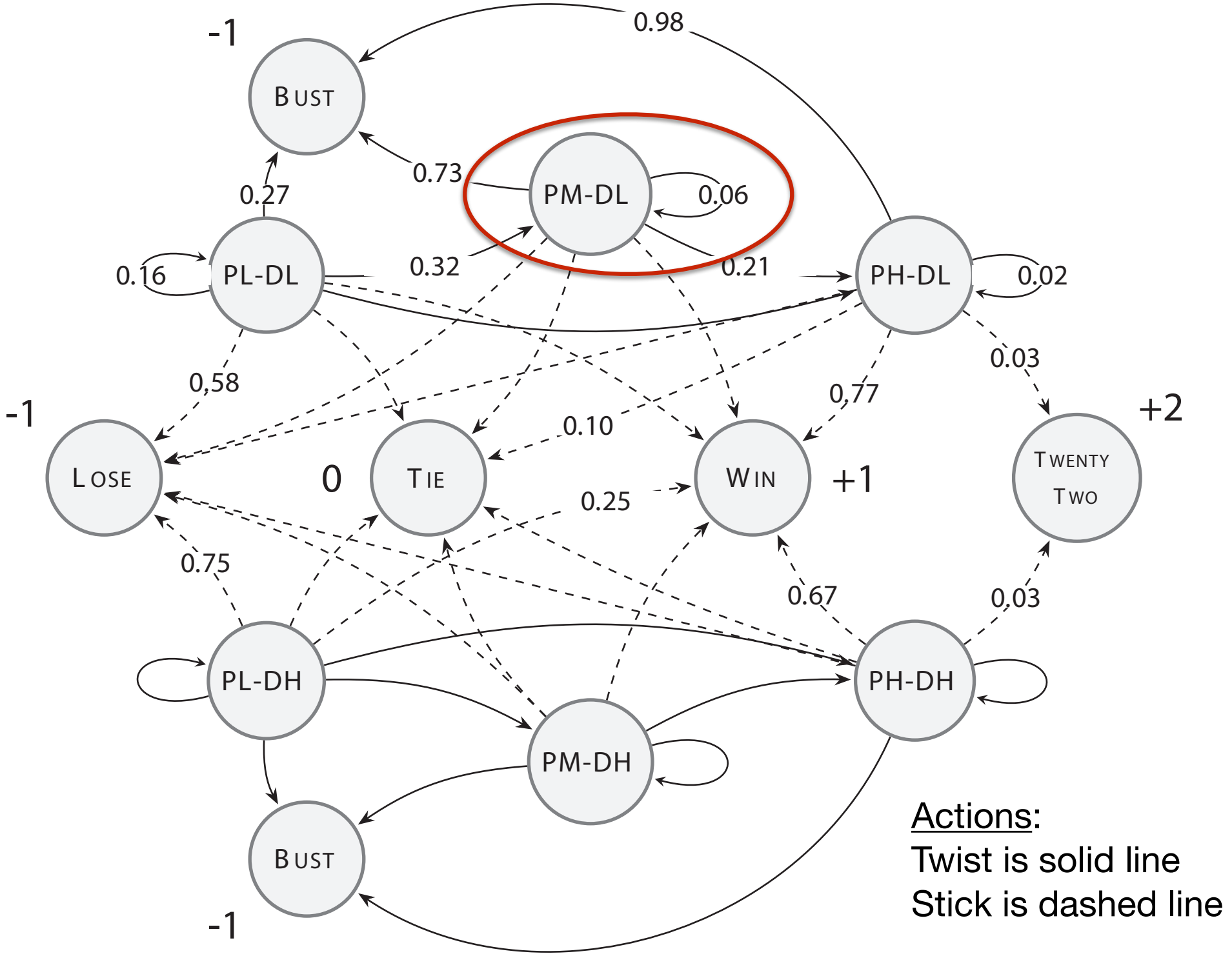
- How do we model this?
  - State representation? Actions? Rewards?

Modelling

- State representation - go for the simplest one that gives sufficient flexibility to model the important aspects of the environment and task
- Discretise the representation of the value of the player and dealer cards
  - PL - Player Low: 4 - 14
  - PM - Player Medium: 15 - 18
  - PH - Player High: 19 - 22
  - DL - Dealer Low: 2 - 7
  - DH - Dealer High: 8 - 11
- 6 states + 5 terminal states

Iter	Player Hand		Dealer Hand		State	Action	Reward
1	2♥ 7♣	(9)	8♥	(8)	PL-DH	*Twist*	0
2	2♥ 7♣ *K*♣	(19)	8♥	(8)	PH-DH	*Stick*	+1
3	2♥ 7♣ *K*♣	(19)	8♥ *Q*♦	(18)	WIN		
1	4♠ *A*♥	(15)	*Q*♥	(10)	PM-DH	*Twist*	-1
2	4♠ *A*♥ 9♣	(24)	*Q*♥	(10)	BUST		
1	2♦ 4♦	(6)	3♥	(3)	PL-DL	*Twist*	0
2	2♦ 4♦ 3♥	(9)	3♥	(3)	PL-DL	*Twist*	0
3	2♦ 4♦ 3♥ 6♣	(15)	3♥	(3)	PM-DL	*Twist*	0
4	2♦ 4♦ 3♥ 6♣ 6♦	(21)	3♥	(3)	PH-DL	*Stick*	0
5	2♦ 4♦ 3♥ 6♣ 6♦	(21)	3♥ 7♥ *A*♠	(21)	TIE		
1	*Q*♦ *J*♣	(20)	*A*♥	(11)	PH-DH	*Stick*	+1
2	*Q*♦ *J*♣	(20)	*A*♣ 5♣ *Q*♠	(26)	WIN		
1	*A*♦ *A*♥	(22)	2♥	(2)	PH-DL	*Stick*	+2
2	*A*♦ *A*♥	(22)	2♥	(2)	TWENTYTWO		


-1
0.98
BUST
0.73
PM-DL
0.06
0.27
0.32
0.21
0.16
PL-DL
PH-DL
0.02
0.03
0,58
0,77
-1
0.10
+2
LOSE
0
TIE
WIN
+1
TWENTY TWO
0.25
0.75
0.67
0,03
PL-DH
PH-DH
PM-DH
BUST
-1
Actions:
Twist is solid line
Stick is dashed line


-1
BUST
0.98
0.73
PM-DL
0.06
0.27
0.32
0.21
0.16
PL-DL
PH-DL
0.02
0.03
0,58
0,77
-1
0.10
+2
LOSE
0
TIE
WIN
+1
TWENTY TWO
0.25
0.75
0.67
0,03
PL-DH
PH-DH
PM-DH
BUST
-1
Actions:
Twist is solid line
Stick is dashed line

Back to RL…

- Express the action-value function in terms of an MDP

$$Q_\pi(s_t, a_t) = E_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots + \gamma^{3-t} r_e \mid s_t, a_t]$$

$$Q_\pi(s_t, a_t) = \sum_{s_{t+1}} P(s_t \xrightarrow{a_t} s_{t+1}) \left[R(s_t \xrightarrow{a_t} s_{t+1}) + \gamma \max_{a_{t+1}} Q_\pi(s_{t+1}, a_{t+1}) \right]$$

- For the optimal policy, this is known as the Bellman optimality equation which states that the maximal cumulative return following an action a_t taken in state s_t is earned by continuing to take the action that will return the maximal return
- This gives a set of equations for each action and state and solving these is complex…
- RL uses iterative approaches that calculate approximate solutions - one of the most important is temporal-difference learning

Temporal Difference Learning

- A simple, iterative, tabular approach to learning the action-value function $Q_\pi(s_t, a_t)$
- The action-value table is used to store estimates of the expected return from taking each available action in each possible state
  - entries are initialised randomly
  - terminal states always have a value of zero for all actions
- The agent is deployed in the environment
- The entries in the table are updated on the basis of the performance of the agent after each action is taken
- A learning rate is used to control the size of the changes

Action-value table for TwentyTwos

State	Action	Value	State	Action	Value	State	Action	Value
PL-DL	*Twist*	0.039	PH-DL	*Twist*	−0.666	PM-DH	*Twist*	−0.668
PL-DL	*Stick*	−0.623	PH-DL	*Stick*	0.940	PM-DH	*Stick*	−0.852
PM-DL	*Twist*	−0.597	PL-DH	*Twist*	−0.159	PH-DH	*Twist*	−0.883
PM-DL	*Stick*	−0.574	PL-DH	*Stick*	−0.379	PH-DH	*Stick*	0.391
BUST	*Twist*	0.000	TIE	*Twist*	0.000	WIN	*Twist*	0.000
BUST	*Stick*	0.000	TIE	*Stick*	0.000	WIN	*Stick*	0.000
LOSE	*Twist*	0.000				TWENTYTWO	*Twist*	0.000
LOSE	*Stick*	0.000				TWENTYTWO	*Stick*	0.000

Update Rule:

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha(\underbrace{G(s_t, a_t) - Q_\pi(s_t, a_t)}_{\text{difference between actual and expected returns}})$$

Actual return from the point of taking action a_t to the end of the episode

Update Rule

- But we don't know the actual return after each action until the end of the episode…
- Bootstrapping uses existing estimates of expected returns to make updates rather than waiting for the episode to complete

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha(\underbrace{r_t + \gamma Q_\pi(s_{t+1}, a_{t+1})}_{\text{actual return}} - \underbrace{Q_\pi(s_t, a_t)}_{\text{expected return}})$$

immediate reward

discount rate

estimate of return in the next state

Update Rule

- But we don't know the actual return after each action until the end of the episode…
- Bootstrapping uses existing estimates of expected returns to make updates rather than waiting for the episode to complete

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha(\underbrace{r_t + \gamma Q_\pi(s_{t+1}, a_{t+1})}_{\text{actual return}} - \underbrace{Q_\pi(s_t, a_t)}_{\text{expected return}})$$

immediate reward

discount rate

estimate of next state

New estimate = current estimate + scaling term x error

……does this sound familiar?

Exploration vs Exploitation

- The dilemma for decision making systems that have incomplete knowledge of the world is whether to repeat decisions have worked well so far, exploit, or to make novel decisions, explore, hoping to do better
  - e.g. in applications such as recommender systems or online advertising and in RL

- The policy when training in reinforcement learning needs a balance between exploitation and exploration

Policies

- A greedy strategy takes the decision that seems to be the best with respect to the current knowledge
  - take the action with the highest reward
  - exploitation only with no opportunity for learning
- A random strategy takes a random decision
  - exploration only with no opportunity to use existing knowledge
- An ϵ-greedy policy takes the best action most of the time and chooses a random action with a probability of ϵ from those available

- Note that different policies can be used for training and deployment
  - a behaviour policy is used for training
  - a target policy is used for deployment

Q-learning algorithm for RL

Require: a behavior policy, π, that chooses actions
Require: an action-value function Q that performs a lookup into an action-value table with entries for every possible action, a, and state, s
Require: a learning rate, α, a discount-rate, γ, and a number of episodes to perform

1: initialize all entries in the action-value table to random values (except for terminal states which receive a value of 0)
2: **for** each episode **do**
3: reset s_t to the initial agent state
4: **repeat**
5: select an action, a_t, based on policy, π, current state, s_t, and action-value function, Q
6: take action a_t observing reward, r_t, and new state s_{t+1}
7: update the record in the action-value table for the action, a_t, just taken in the last state, s_t, using:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right) \quad (23)$$

8: let $s_t = s_{t+1}$
9: **until** agent reaches a terminal state
10: **end for**

Q-Learning

- Q-learning is referred to as off-policy
  - the value-action table is updated using a greedy policy regardless of the behaviour policy selected in the algorithm
  - an optimistic strategy

TwentyTwos example

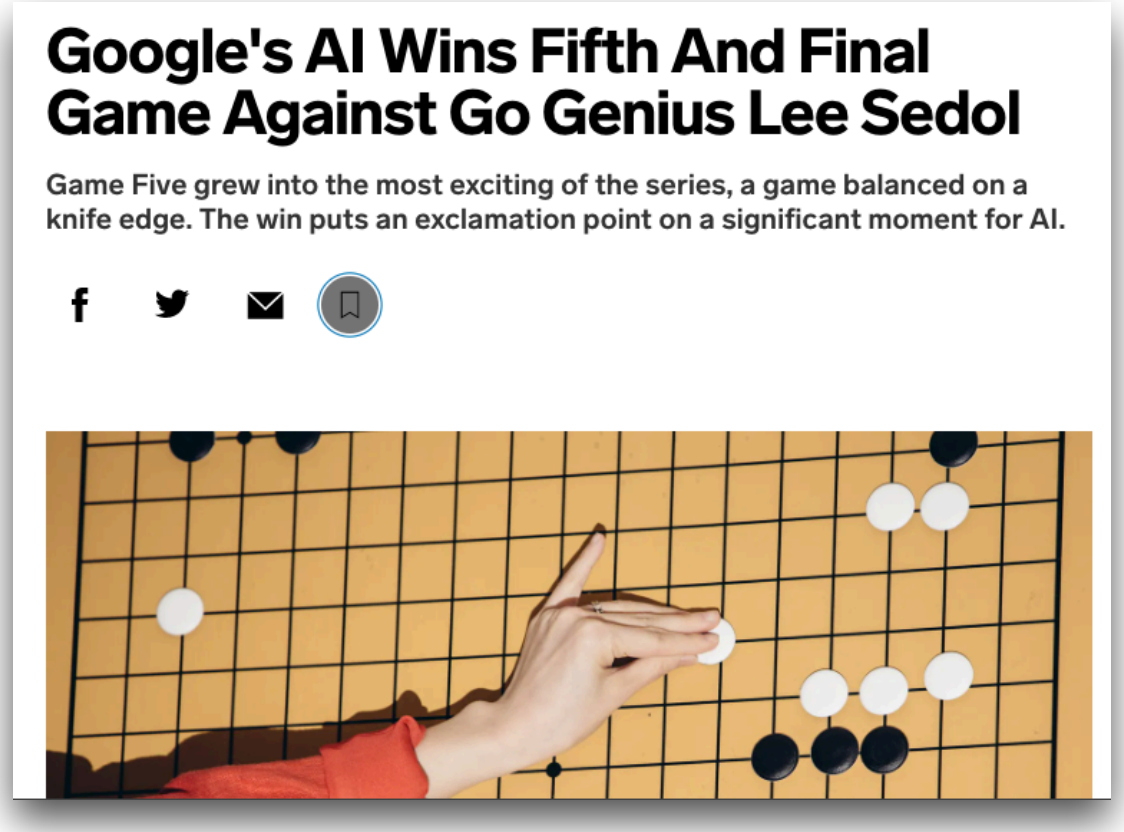
State	Action	Value	State	Action	Value	State	Action	Value
PL-DL	*Twist*	0.039	PH-DL	*Twist*	−0.666	PM-DH	*Twist*	−0.668
PL-DL	*Stick*	−0.623	PH-DL	*Stick*	0.940	PM-DH	*Stick*	−0.852
PM-DL	*Twist*	−0.597	PL-DH	*Twist*	−0.159	PH-DH	*Twist*	−0.883
PM-DL	*Stick*	−0.574	PL-DH	*Stick*	−0.379	PH-DH	*Stick*	0.391
BUST	*Twist*	0.000	TIE	*Twist*	0.000	WIN	*Twist*	0.000
BUST	*Stick*	0.000	TIE	*Stick*	0.000	WIN	*Stick*	0.000
LOSE	*Twist*	0.000				TWENTYTWO	*Twist*	0.000
LOSE	*Stick*	0.000				TWENTYTWO	*Stick*	0.000

- Learned from 100,000 episodes of Q-learning
- A cautious strategy
- An evaluation of 100,000 runs of 1000 hands
  - Q-learning agent wins $\$198 \pm 24$ out of 1000 hands
  - Random agent loses $\$197 \pm 25$ out of 1000 hands

Summary

- A reinforcement learning agent is deployed into an environment and learns from experimenting in the environment
- Suitable for control tasks, e.g. robotics or game playing

- Supervised learning can be used for such tasks but RL is
  - cheaper and less time consuming
  - can find exceptional or surprising solutions that humans are unaware of
  - requires large computational resources



https://www.wired.com/2016/03/googles-ai-wins-fifth-final-game-go-genius-lee-sedol/