# Clustering

One of the classic unsupervised problems is clustering; the idea that items can be put into groups by how similar they are

## Single Linkage Clustering

* Consider each of n object a cluster to start
* Define distance (i.e. distance between two closest points in two clusters)
* merge two closest clusters
* repeat n-k times, to come up with k clusters

SLC runs O(N3) since we do N searches over each N object for N-K times

## K-Means Clustering

* Pick centers at random
* each center claims its closest points
* Recompute centers by averaging the clustered points
* Repeat until it converges

If we define (at iteration t) :

as partition of object x

a set of all points in a cluster of such that , and the

the center of the cluster

then:

That is, the partition chosen is the one that produced the smallest euclidean distance between point x and the center. This partition is used to update the center and the algorithm repeats until convergence

### Optimization

Define the error function as

## Properties

* Each iteration polynomial (K\*n)
* finite iterations O(Kn)
* Error decreases (if ties broken consistently)
* Can get stuck

## Maximum Likelihood Gaussian

the ML Gaussian is the mean of the data

## Expectation Maximization (soft clustering)

Suppose a problem with data set D contains instances generated by a probability distribution of k distinct normal distributions. We would like to find the maximum likelihood hypothesis h such that we maximize the probability of the data given h:

can be given as the product of probabilities of viewing any instance in class :

And since a logarithmic function is monotonic, applying the logarithm to both sides will preserve the order of and yields:

when we eliminate constant terms, or restating with context to the mean, and for a single distribution:

However, when we have multiple distributions producing our feature space, we don’t know from which is produced. That is, each observation can be considered a tuple of the observed value and j hidden variable indicating which distribution it came from (i.e. is 1 if and only if came from distribution j:

If we start with arbitrary and assume this hypothesis holds, we can calculate the expected value of each of the j hidden variables and then recalculate assuming the hidden variables are equal to their expected values of the hidden variables. Iterating, until convergence

The expected values of the hidden variables can be derived for a normal distribution as follows:

.

Assuming a monotonic but unknown distribution, we can simplify the application of EM by defining a function Q, which is used in the estimate step, as follows:

and h, which is used in the maximization step,

From here, a derivation of K-Means is possible, first by defining the probability of seeing the data given a hypothesis:

And then deriving the Q function:

and its argmax over h’ given by:

This equation is minimized by setting each mean of the j samples equal to sample mean, yielding k-means:

### Properties

Monotonically non decreasing

* Does not necessarily converged (but does not diverge)
* Can get stuck at maxima
* Works with any distribution (If E and M are solvable)

# Clustering Algorithm Properties

* Richness
  + For any assignment of clusters, there is some distance matrix that returns that cluster
* Scale Invariance
  + Scaling distances by a positive value should not impact the clusters (i.e. change of units)
* Consistency
  + Shrinking intracluster distance and expanding the intercluster distance does not impact clustering

Impossibility Theorem: No clustering algorithm can do all three

"start with a clustering where all points are in the same cluster. Transform it into any other arrangement by shrinking it into a miniature version of any other arrangement and scaling it up to a full-size version of the other arrangement."

Why you can't have all 3:

https://stats.stackexchange.com/questions/173313/clustering-intuition-behind-kleinbergs-impossibility-theorem

# Feature Selection

Why would we want to do this?

* Knowledge discovery, specifically interoperability and insight
  + i.e.Which features matter?
* Curse of dimensionality
  + The amount of data we need grows exponentially with the amount of features we use
  + If we reduce the number of relevant features required, the amount of data required decreases
  + We also can avoid overfitting

This is an NP-Hard problem O(2N) since we don’t know which is the best transformation from N to M features.

## Filtering and Wrapping

Filtering performs a search through features to create a redacted set (without reference to the learner); typically this is fast, but ignores the learning problem

* ID3 may be an example of this (information gain at each node is the search algorithm)
* Variance, entropy may also be used to inform the search algorithm
* We can search for independent features (as we don’t need the dependent features as they can be reconstructed)
* Slow for isolated features

Wrapping the learner feeds back performance of redacted set; this approach takes into account the model bias, however it’s much slower because it needs to feedback and evaluate each approach

* Randomized Optimization algorithms are examples of algorithms that can be used here because they feedback performance through fitness functions
* Forward search is an approach by which we start with a hypothesized best feature set and add features to determine whether performance improves
* Backward search is an approach by which we try all subsets of features to determine the best one

### Relevance

* Relevance is about information - specifically the impact on a bayes optimal classifier
* Xi is strongly relevant if removing it degrades the bayes optimal classifier
* Xi is weakly relevant if it is not strongly relevant, there exists subset feature if S such that adding that Xi to S improves the bayes optimal classifier. As an example, consider a dependent feature - removing the feature on which it’s dependent would allow the feature in question to improve a bayes optimal classifier
* Xi is irrelevant otherwise

### Usefulness

* Usefulness is about effect on error - specifically the impact on a predictor (model or learner)
* Even if a feature is not relevant, it can reduce the error

## Entropy and Information Gain

### Entropy

Entropy is the measure of order in a given collection of examples. Given a collection, the entropy of can be given by:

Where is the number of different values represented in the distribution and represents the probability of that value in the distribution. By using we can represent entropy as the number of bits required to describe the sample set

### Information Gain

Information gain is defined as the reduction in entropy caused by partitioning a sample set. More precisely, the information gain after partitioning a collection of samples, , for all possible values for attribute is the entropy of the original subset less the sum of the entropies of the partitioned subset, weighted as a fraction of the partitioned subset over the original subset:

## Perceptrons

A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs based on assigned weights and outputs a 1 if the results exceeds a defined threshold, , or a 0 otherwise.

## Bayes Theorem

Typically we want to know if some hypothesis of class is true given an observed data set . If we recall the chain rule, we can represent this probability, which is typically difficult to estimate, with probabilities that are easier to derive:

is the prior probability, or the probability that some hypothesis holds before observing any data. is the probability of observing the data, and p(D|h) is the probability of observing that data given that hypothesis holds

### Optimal Bayes Classifier

GIven we know the probability of a hypothesis given the data, we can derive the probability of a certain classification value belonging to value space over data set as:

and the optimal classifier given by the of this equality

# Feature Transformation

Feature transformation is the preprocessing of a set of features in order to create a new set of features. We expect this new set of features to be smaller or more compact. We also expect this redacted set of features to contain as much information as possible (relevant) as well as be useful.

We can think of Feature Selection as a subset of feature transformation; however, in this section only discussion of linear feature transformations will be discussed such that represents a linear transformation of feature space F such that , where m < n

## Information retrieval

Ad Hoc problem (i.e. the Google problem), we get an unknown query and we retrieve results dependent on a set of predefined features. In a text search, these features may be:

* Words, which have the following issues:
  + means many features, and curse of dimensionality
  + words may mean different thing (false positives)
  + Many words also mean the same thing (false negatives)

## Principal Component Analysis

### Eigenproblems and Eigenvectors[[1]](#footnote-0)

A (non-zero) vector v of dimension N is an eigenvector of a square (N×N) matrix A if and only if it satisfies the linear equation where is a scalar value, Matrices can be decomposed as follows:

Taking a 2 × 2 real matrix, A, as an example to be decomposed into a diagonal matrix through multiplication of a non-singular matrix B\mathbf{B} = \begin{bmatrix} a & b \\ c & d \\ \end{bmatrix} \in \mathbb{R}^{2\times2} \mathbf{A} = \begin{bmatrix} 1 & 0 \\ 1 & 3 \\ \end{bmatrix}

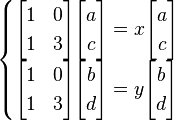
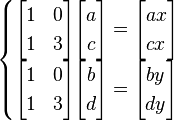
.

Then,for some real diagonal matrix a,d

\begin{bmatrix} a & b \\ c & d \\ \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 \\ 1 & 3 \\ \end{bmatrix} \begin{bmatrix} a & b \\ c & d \\ \end{bmatrix} = \begin{bmatrix} x & 0 \\ 0 & y \\ \end{bmatrix},

such that





Letting \overrightarrow{a} = \begin{bmatrix} a \\ c \end{bmatrix}, \overrightarrow{b} = \begin{bmatrix} b \\ d \end{bmatrix}, this gives us two vector equations: \begin{cases} A \overrightarrow{a} = x \overrightarrow{a} \\ A \overrightarrow{b} = y \overrightarrow{b} \end{cases}

Where  \lambda represents the two eigenvalues x and y

### 

### PCA

PCA is an example of an eigenproblem (rotation) which

* maximizes variance along the components
* is a rotation that produces orthogonal components
* best reconstruction as it’s a linear rotation of original feature space and minimizes L2 (square) error by moving from N to M dimensions
* Each principal component has a prescribed eigenvalue
* We can throw away those with small eigenvalues (if eigenvalue is equal to 0, that value is irrelevant), as they correspond to components that matter less in the reconstruction
* By definition, this means that features are ordered

We can center the problem around the origin (by subtracting the mean of the data).

In effect PCA, produces a set of orthogonal gaussians.

## Independent Component Analysis

### Information Theory

Joint Entropy is the randomness of two variables given together

Conditional Entropy is the randomness of one variable given the other. If these variables are mutually independent and

#### Mutual Information

When two variables are not independent, their MI is the measure of how related they are to each other: the measure of the reduction of the randomness of a variable given knowledge of the other:

**Conditional entropy**

### ICA

ICA tries to find a linear transformation of the feature space such that each new feature is linearly independent - i.e. their mutual information is 0, but produces maximal mutual information between the old and new feature space. i.e. for some mapping of X to Y,

and

#### Blind Source Separation Problem (cocktail problem)

If you’re in a large room listening to multiple conversations, can we pull out individual sources by analyzing different microphones picking up these sources

* Observables = microphones
* hidden variables = people’s voices

Each microphone has a linear combination of all sources. If we have three observables and three sources, we can solve the linear problem to recover the original sources.

#### Central Limit Theorem

If you attempt to maximize variance of a large enough set of mutually independent variables, you’ll essentially be creating a gaussian

## Random Component Analysis

A randomized projection of the original data, which surprisingly works well but tends to have higher dimensional transformed space

## Linear Discriminant Analysis

Find a projection that discriminates on the label; that is, find a projections of features that ultimately align best to the desired output (wrapping function instead of filtering)

# Markov Decision Processes

Given a state and a model ,with actions with rewards, we are trying to find the set of actions for each state that result in optimal policy, such the is a combination of actions for all s.

For an MDP, we require that:

* Markovian Property holds, meaning that only the current state matters in determining the optimal policy (we can abstract to say any state represents the cumulative impact of visiting past states)
* Stationarity holds, meaning that the state of the problem is stationary (i.e. is matrix T is constant)

## Changes to Rewards

Rewards can be delayed, which yields to the temporal credit assignment problem. This means that it is difficult to understand which action in a specific policy resulted in cumulative or specific rewards at a given state (including at the end of the MDP, or final reward).

Small changes to a reward function matter, especially as the number of states increases

The policy to maximize reward will change depending on whether we have an infinite horizon (unlimited number of steps) or finite horizon. Our policy could then be a function of both our action and time step

### Utility of Sequences

If we have a utility for a stationary MDP following a specific policy beginning at state which is greater than stationary MDP following a specific policy beginning at state , then, for any positive :

The present value of the utility function for some discount factor at time can be given as:

## Finding Policies

We start with a definition of an optimal policy as one which maximizes the expected value of Utility over that policy:

And for any given state , the utility of that state is expected value of the sum of the discounted rewards from there on:

Then, for any given state the optimal policy can be defined as the one which results from an action which maximizes the utility of the next state given that we follow the optimal policy is given by the Bellman equation:

### Value Iteration

Start with an arbitrary policy, and update based on neighbours. That is, start with arbitrary values for for all , and then update recursively by:

At each step, we’re moving closer to the answer. Since our discount factor is less than 1, the remainder of the utility of all other states impacts the current utility being evaluated less, making values better each iteration

### Policy Iteration

Since we’re ultimately interested in finding the optimal policy, calculating all utilities may be wasteful. instead, if we start with an arbitrary policy , and update our policy such that:

and

The latter is a system of linear equations with unknowns, which is solvable

## Q-Learning

Consider that reinforcement learning can be thought of three component parts:

* The model (T,R)
* Transitions <s,a,r,s’>
* The policy (set of actions)

Abstracting the various algorithms in RL, we can categorize each into one of four categories:

* A planner maps models to policy
* A learner maps transitions to policy
* A modeler maps transitions to models
* A simulator maps models to transition

In many practical examples, we don’t know what the reward function may be for any given state. As such, policy and value iteration, which rely on knowledge of the model of the current and successor states, may prove inadequate for us to solve an MDP. We can define a function as

and we can estimate the best policy and the utility at any given state by:

The Q algorithm will converge. given that each state-action pair is visited infinitely often

Initializing the Q value of each state to 0, the algorithm progresses through each state, by choosing states with non-zero rewards and updating adjacent states with new values of Q recursively. Intuitively, we can see that Q will converge as it will incrementally move towards representing the true utility of each state. This is true for all deterministic MDP’s bounded by |r(s,a)|< c, for which the Q algorithm can visit each state infinitely often. Mathematically, if we state that our estimate of Q, QQ:

and there exists some nonzero error:

And for any arbitrary step of the Q algorithm:

Since each subsequent error in the table is smaller than the last, the largest is , the error at the first step. The error converges to 0 for sufficiently large since

For non-deterministic cases, we must account for probability of not following the desired action:

Sarsa vs. Q learning

The reason that Q-learning is off-policy is that it updates its Q-values using the Q-value of the next state s′s′ and the *greedy action* a′a′. In other words, it estimates the *return* (total discounted future reward) for state-action pairs assuming a greedy policy were followed despite the fact that it's not following a greedy policy.

The reason that SARSA is on-policy is that it updates its Q-values using the Q-value of the next state s′s′ and the *current policy's* action a″a″. It estimates the return for state-action pairs assuming the current policy continues to be followed.

The distinction disappears if the current policy is a greedy policy. However, such an agent would not be good since it never explores.

[[Source](https://stats.stackexchange.com/questions/184657/difference-between-off-policy-and-on-policy-learning)]

# Game Theory

We’ve been discussing decision making in the context of a single agents trying to maximize rewards, however when conflict arises when there are multiple actors; GT is the mathematics of conflicts

## Von Neumann’s Theorem

In a 2-player, 0-sum, non-deterministic game of perfect information, a strategy is a mapping a possible state to action; however, each player must consider the worst case strategy for the other(s). This leads to a minimax strategy, which translates to the maximal strategy for a certain player is the one that minimizes the other’s, and is equal to maximin. There always exists an optimal strategy for each agent.

This theorem does not hold for games with hidden information

## Pure and Mixed Strategies

For deterministic games, we have a finite set of pure strategies possible

For non-deterministic games, we can have a distribution over strategies; however, the value, in the long run, the expected value of the game is constant for any given probability distribution P- this is known as a mixed strategy.

**Nash Equilibrium**

For each player, assuming the other players do not change their strategy, there is no advantage in changing their own strategy.

* If in an n-player pure strategy game, if elimination of all strictly dominated strategies eliminates in all but one remaining strategy, that strategy is in NE
* Any NE will survive elimination of strictly dominated strategies
* An n-repeated game will result in n-repeated NE

## Iterated Problems

### Relationships to MDP

Given a finite number of steps, we can represent each strategy as a finite state machine: an MDP, where, given an infinite discounted run, the payoff will be discounted by

### Implausible Threats

An agent may choose not to apply a punitive strategy since its application would be detrimental to the agent themselves. As an example, consider Grim Trigger, a strategy which will cooperate if you cooperate, but will defect forever if you defect once. In a scenario where mutual defection leads to lower rewards than mutual cooperation, the threat of application of this strategy is diminished.

#### Subgame Perfect

Regardless of the history of actions before a strategy is applied, the strategy will still lead to an optimal solution. If all agent’s strategy contains a plausible threat, then it will be subgame perfect.

#### 

### Folk Theorem

#### Folk Theorem

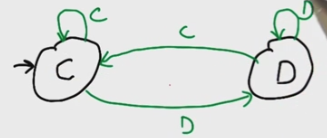
Describes the set of Nash strategies that result from repeated games: Any feasible payoff profile that strictly dominates the minmax/security level profile can be realized as a Nash equilibrium payoff profile, with sufficiently large discount factor.

#### Computational Folk Theorem

We can build a pavlov-like machine for any 2-player, bi-player, average reward game in polynomial time

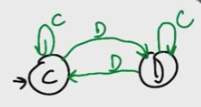
### Machines

#### Tit for Tat



Start with cooperating, and then always respond in kind. e.g. If the opponent cooperated last time, cooperate this time. Not subgame perfect in prisoner's dilemma but is in Nash equilibrium.

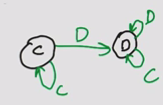
#### Pavlov



Cooperate if agree, defect if disagree. Is both a Nash Equilibrium and is subgame perfect. B

#### Grim trigger

Start of cooperating and cooperate if the other player cooperates. Defect forever as soon as the other player defects, no matter what happens. Can’t do anything better against Grim than to use Grim, therefore Nash Equilibrium!



## Stochastic Games and Multi-game-RL

### Properties

States,

Action for player i, , a,b

Transitions,

Rewards for player i,

Discount,

### Zero-Sum games

Revisiting the bellman equation, we have to account for multiplayer maximization strategies, so we employ a minimax function instead of a max function, resulting in minimax Q:

Properties of the solutions:

* Value iteration works
* minimax-Q converges
* Unique solution to Q, with each policy able to be computed independently and efficiently
* The Q function, as before, is sufficient to solve for the policy

### General Sum Games

Revisiting the bellman equation, we have to account for multiplayer maximization strategies, so we employ a Nash function instead of a max function, resulting in Nash Q:

Properties of the solutions:

* Value iteration does not work
* minimax-Q does not converge
* No Unique solution to Q
* each policy can not be computed independently and efficiently (PPAD)
* The Q function is not sufficient to solve for the policy

Instead of using Q learning, we can use the following ideas:

* Repeated stochastic games (folk theorem)
* Cheaptalk and correlated equilibrium
* Cognitive hierarchy - instead of solving for equilibrium, assume that players have less cognitive resources and attempt best response base on that
* Side payments - bribing other players from your payout

1. http://en.wikipedia.org/wiki/Eigendecomposition\_of\_a\_matrix#Real\_symmetric\_matrices [↑](#footnote-ref-0)