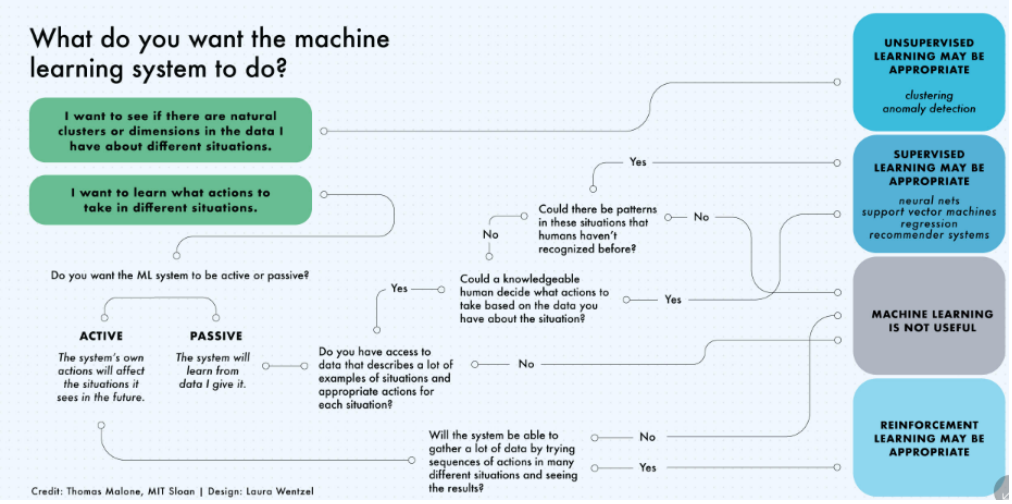
Algo-Selector

Volunteers

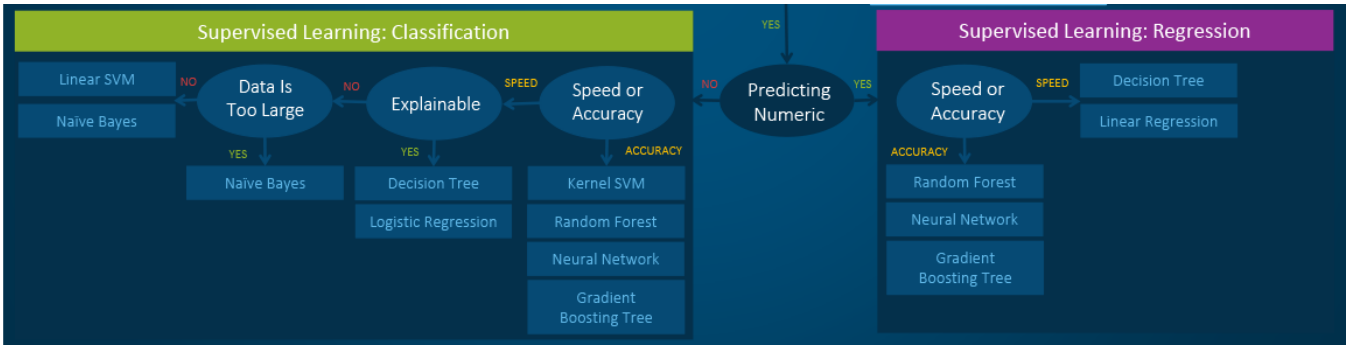
Name	ML Category
Jahanvi	Supervised
Akanksha	Unsupervised
Kanak Raj	Reinforced

Supervised

- Supervised learning algorithms make predictions based on a set of examples
- **Classification:** When the data are being used to predict a categorical variable, supervised learning is also called classification. This is the case when assigning a label or indicator, either dog or cat to an image. When there are only two labels, this is called binary classification. When there are more than two categories, the problems are called multi-class classification.
- **Regression:** When predicting continuous values, the problems become a regression problem.
- **Forecasting:** This is the process of making predictions based on past and present data. It is most commonly used to analyze trends. A common example might be an estimation of the next year sales based on the sales of the current year and previous years.



Algorithms



Name	Comments on Applicability	Reference

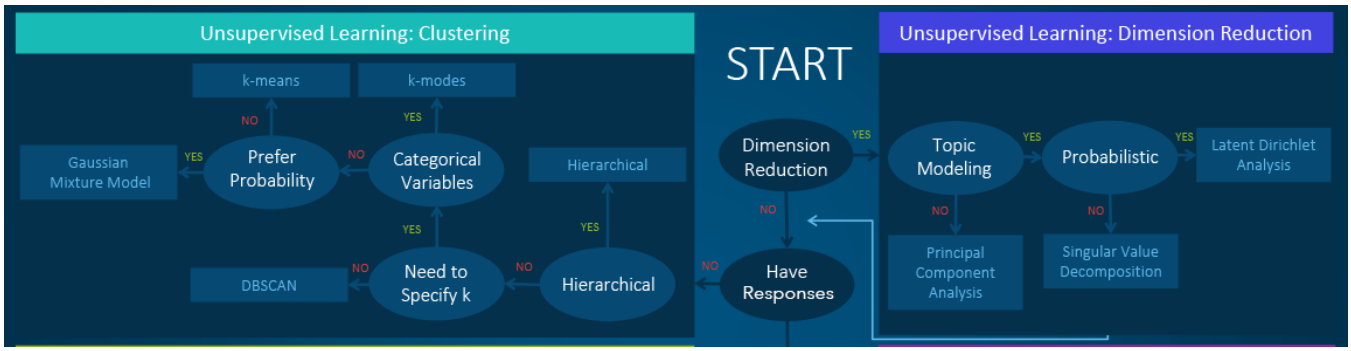
LOGISTIC REGRESSION	1. Logistic regression is kind of like linear regression, but is used when the dependent variable is not a number but something else (e.g., a "yes/no" response). 2. It's called regression but performs classification based on the regression and it classifies the dependent variable into either of the classes. 3. is used for prediction of output which is binary 4. Logistic function is applied to the regression to get the probabilities of it belonging in either class.	
KNN	1. it is used to identify the data points that are separated into several classes to predict the classification of a new sample point. 2. It classifies new cases based on a similarity measure (i.e., distance functions). 3. K-NN works well with a small number of input variables (*p*), but struggles when the number of inputs is very large.	
SUPPORT VECTOR MACHINE	1. Support vector is used for both regression and classification. 2. It is based on the concept of decision planes that define decision boundaries. A decision plane (hyperplane) is one that separates between a set of objects having different class memberships. 3. The learning of the hyperplane in SVM is done by transforming the problem using some linear algebra	
Kernel SVM	1. it takes in a kernel function in the SVM algorithm and transforms it into the required form that maps data on a higher dimension which is separable. 2. Kernel trick uses the kernel function to transform data into a higher dimensional feature space and makes it possible to perform the linear separation for classification.	
RBF Kernel	1. The RBF kernel SVM decision region is actually also a linear decision region. So, the rule thumb is: use linear SVMs for linear problems, and nonlinear kernels such as the RBF kernel for non-linear problems.	
NAIVE BAYES	1. The naive Bayes classifier is based on Bayes' theorem with the independence assumptions between predictors (i. e., it assumes the presence of a feature in a class is unrelated to any other feature). Even if these features depend on each other, or upon the existence of the other features, all of these properties independently. Thus, the name naive Bayes. 2. Based on naive Bayes, Gaussian naive Bayes is used for classification based on the binomial (normal) distribution of data. 3. naive Bayes model is easy to build, with no complicated iterative parameter estimation, which makes it particularly useful for very large datasets.	
DECISION TREE CLASSIFICATION	1. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. 2. Entropy and information gain are used to construct a decision tree. 3. The disadvantage of a decision tree model is overfitting, as it tries to fit the model by going deeper in the training set and thereby reducing test accuracy.	
RANDOM FOREST CLASSIFICATION	1. Random forest classifier is an ensemble algorithm based on bagging i.e bootstrap aggregation. 2. Ensemble methods combines more than one algorithm of the same or different kind for classifying objects (i.e., an ensemble of SVM, naive Bayes or decision trees) 3. The general idea is that a combination of learning models increases the overall result selected. 4. random forests prevent overfitting by creating trees on random subsets. 5. The main reason is that it takes the average of all the predictions, which cancels out the biases.	
GRADIENT BOOSTING CLASSIFICATION	1. Boosting is a way to combine (ensemble) weak learners, primarily to reduce prediction bias. Instead of creating a pool of predictors, as in bagging, boosting produces a cascade of them, where each output is the input for the following learner. 2. Gradient boosting takes a sequential approach to obtaining predictions instead of parallelizing the tree building process. In gradient boosting, each decision tree predicts the error of the previous decision tree—thereby *boosting* (improving) the error (gradient).	

Un-supervised

1. Clustering - hierarchical clustering, k-means, mixture models, DBSCAN, and OPTICS algorithm
2. Anomaly Detection - Local Outlier Factor, and Isolation Forest

3. Dimensionality Reduction - Principal component analysis, Independent component analysis, Non-negative matrix factorization, Singular value decomposition

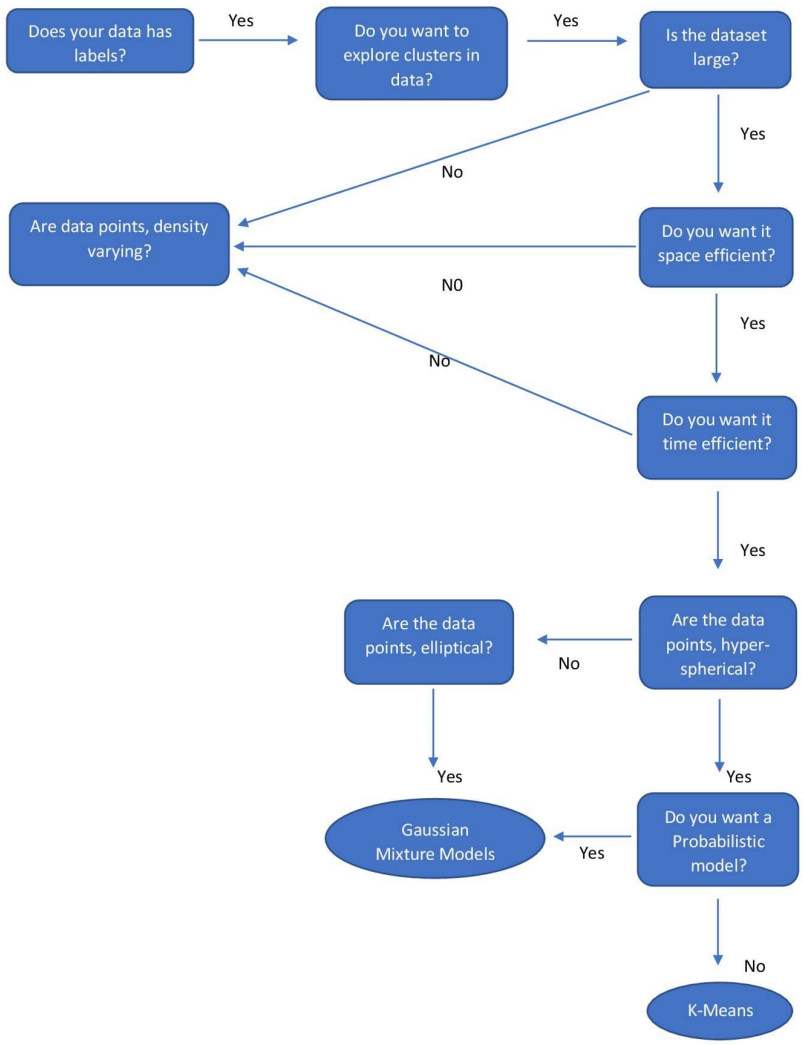
Algorithms

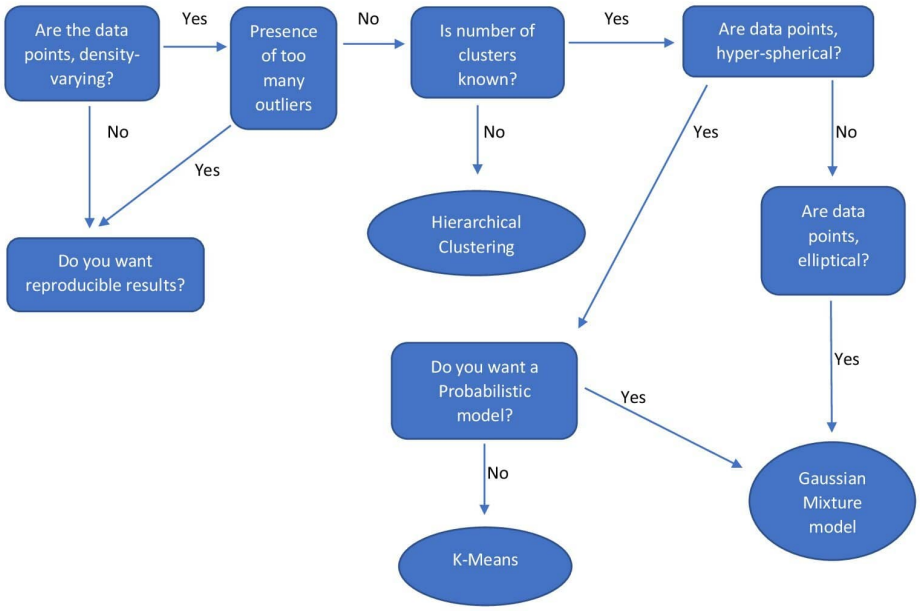


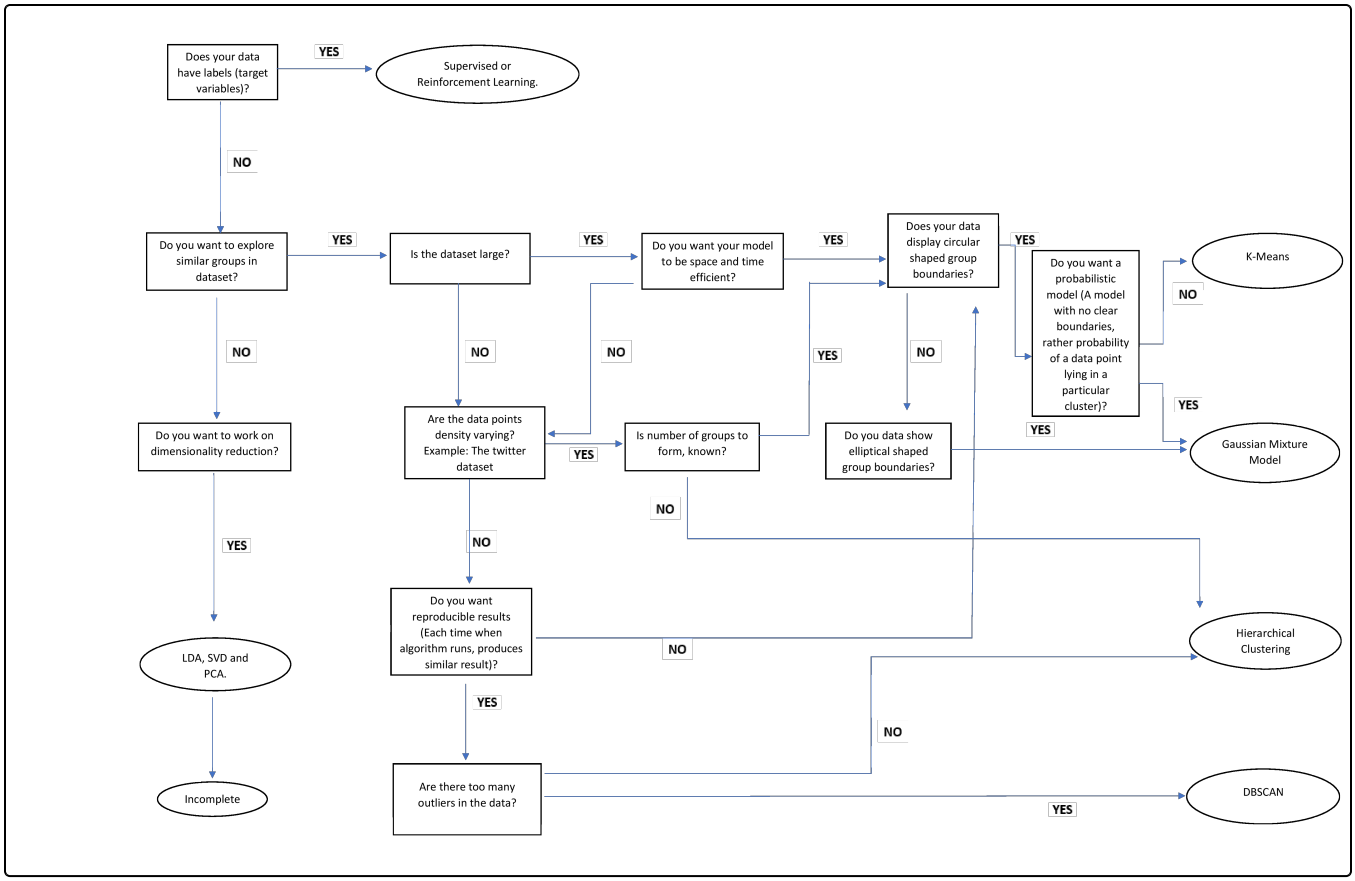
Name	Comments on Applicability	Reference
Hierarchical Clustering	1. (N-1) combination of clusters are formed to choose from. 2. Expensive and slow. n×n distance matrix needs to be made. 3. Cannot work on very large datasets. 4. Results are reproducible. 5. Does not work well with hyper-spherical clusters. 6. Can provide insights into the way the data pts. are clustered. 7. Can use various linkage methods(apart from centroid).	
k-means	1. Pre-specified number of clusters. 2. Less computationally intensive. 3. Suited for large dataset. 4. Point of start can be random which leads to a different result each time the algorithm runs. 5. K-means needs circular data. Hyper-spherical clusters. 6. K-Means simply divides data into mutually exclusive subsets without giving much insight into the process of division. 7. K-Means uses median or mean to compute centroid for representing cluster.	
Gaussian Mixture Models	1. Pre-specified number of clusters. 2. GMs are somewhat more flexible and with a covariance matrix we can make the boundaries elliptical (as opposed to K-means which makes circular boundaries). 3. Another thing is that GMs is a probabilistic algorithm. By assigning the probabilities to data points, we can express how strong is our belief that a given data point belongs to a specific cluster. 4. GMs usually tend to be slower than K-Means because it takes more iterations to reach the convergence. (The problem with GMs is that they have converged quickly to a local minimum that is not very optimal for this dataset. To avoid this issue, GMs are usually initialized with K-Means.)	
DBSCAN	1. No pre-specified no. of clusters. 2. Computationally a little intensive. 3. Cannot efficiently handle large datasets. 4. Suitable for non-compact and mixed-up arbitrary shaped clusters. 5. Uses density-based clustering. Cannot work well with density varying data points. 6. Not effected by noise or outliers.	

DIMENSIONALITY REDUCTION ALGORITHMS	APPLICABILITY
Linear Discriminant Analysis	It is used to find a linear combination of features that characterizes or separates two or more classes of objects or events. LDA is a supervised LDA is also used for clustering sometimes. And almost always outperforms logistic regression.
Principle Component	

Analysis	It performs a linear mapping of the data from a higher-dimensional space to a lower-dimensional space in such a manner that the variance of the data in the low-dimensional representation is maximized. PCA is unsupervised

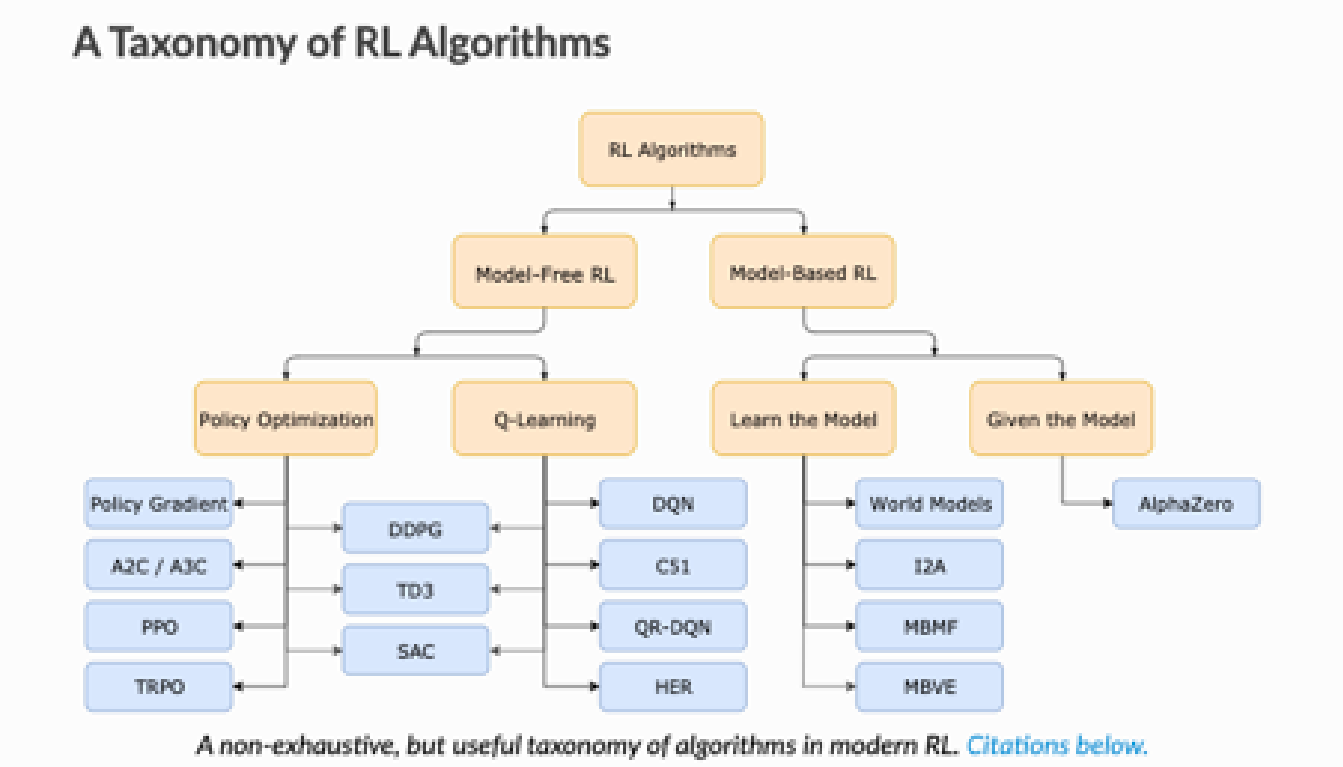



Are the data points, density-varying?
Yes
Presence of too many outliers
No
Is number of clusters known?
Yes
Are data points, hyper-spherical?
No
Yes
No
Yes
No
Hierarchical Clustering
Do you want reproducible results?
Are data points, elliptical?
Yes
Do you want a Probabilistic model?
Yes
No
Gaussian Mixture model
K-Means



Reinforcement Learning

1. Active Learning
2. No labeled data
3. No supervisor, only reward
4. Actions are sequential
5. Feedback is delayed, not instantaneous.
6. Can afford to make mistakes?
7. Is it possible to use a simulated environment for the task?
8. Lots of time
9. Think about the variables that can define the state of the environment.
    a. State Variables and Quantify them
    b. The agent has access to these variables at every time step
    c. Concrete Reward Function and Compute Reward after action
    d. Define Policy Function



Model-Free vs Model-Based RL

Whether the agent has access to (or learns) a model of the environment(a function that predicts state transitions and rewards)

Model Free	Model-Based
forego the potential gains in sample efficiency from using a model	Allows to plan ahead and look in possible results for a range of possible choices.
easier to implement and tune.	Ground Truth Model for any task is generally not available.
	If agents want to use a model then it has to prepare it purely from experience
	fundamentally hard
	being willing to throw lots of time
	High computation
	Can fail off due to over-exploitation of bias

When would model-free learners be appropriate?

If you don't have an accurate model provided as part of the problem definition, then model-free approaches are often superior.

Model-based agents that learn their own models for planning have a problem that inaccuracy in these models can cause instability (the inaccuracies multiply the further into the future the agent looks).

If you have a real-world problem in an environment without an explicit known model at the start, then the safest bet is to use a model-free approach such as DQN or A3C.

The distinction between model-free and model-based reinforcement learning algorithms is analogous to habitual and goal-directed control of learned behavioral patterns. Habits are automatic. They are behavior patterns triggered by appropriate stimuli (think: reflexes). Whereas goal-directed behavior is controlled by knowledge of the value of goals and the relationship between actions and their consequences.

What's the difference between model-free and model-based reinforcement learning?

RL problem is formulated as Markov Decision Process.

Represents the dynamics of the environment, the way the environment will react to the possible actions the agent might take, at any given state.

Transition Function: Fn(state, action) P(all next possible states)

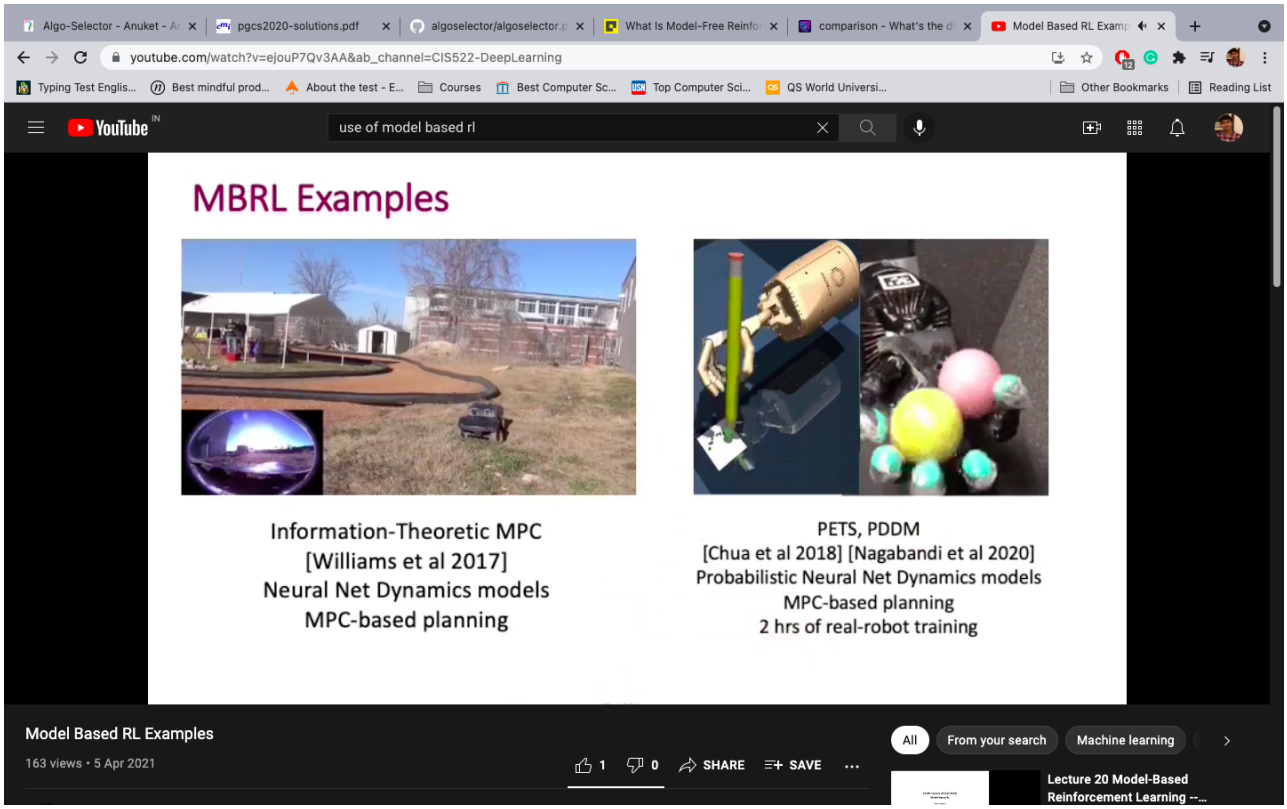
Reward Function: Fn(state) Reward

The Transition Function and Reward Function constitutes the "model" of the RL problem.

Therefore, MDP is the problem and Policy Function is its solution.

The model-based RL can be of two types, algorithm learn the model separately or model is provided.

Examples of Model: Rules of Games, Rules of Physics, etc.



In the Absence of MDP, Agent Interacts with the environment and observes the responses of the environment. They don't have any transition or reward function, the algorithms purely sample and learn from the experience. They rely on real samples from the environment and never use generated predictions of the next state and next reward to alter behavior.

A model-free algorithm either estimates a "value function" or the "policy" directly from experience. A value function can be thought of as a function that evaluates a state (or an action taken in a state), for all states. From this value function, a policy can then be derived.

Examples: Robot Hand Trying to Solve Rubix Cube, Robotic Model learn to walk, etc. These tasks can be learned in a "model-free" approach.

There is no direct distinction b/w model-free and model-based by applications. The use of the algorithm depends upon if "model" is provided or can be learned then use "model-based" else use "model-free".

What to Learn in Model-Free RL

1. Policy Optimization
2. Q-Learning

Policy Optimization	Q-Learning
optimize the parameters either directly by gradient ascent on the performance objective or indirectly, by maximizing local approximations	learn an approximator for the optimal action-value function
performed on-policy, each update only uses data collected while acting according to the most recent version of the policy	performed off-policy, each update can use data collected at any point during training
directly optimize for the thing you want	indirectly optimize for agent performance
More stable	tends to be less stable

advantage of being substantially more sample efficient when they do work, because they can reuse data more effectively	Less sample efficient and takes longer to learn as learning data is limited at every iteration.

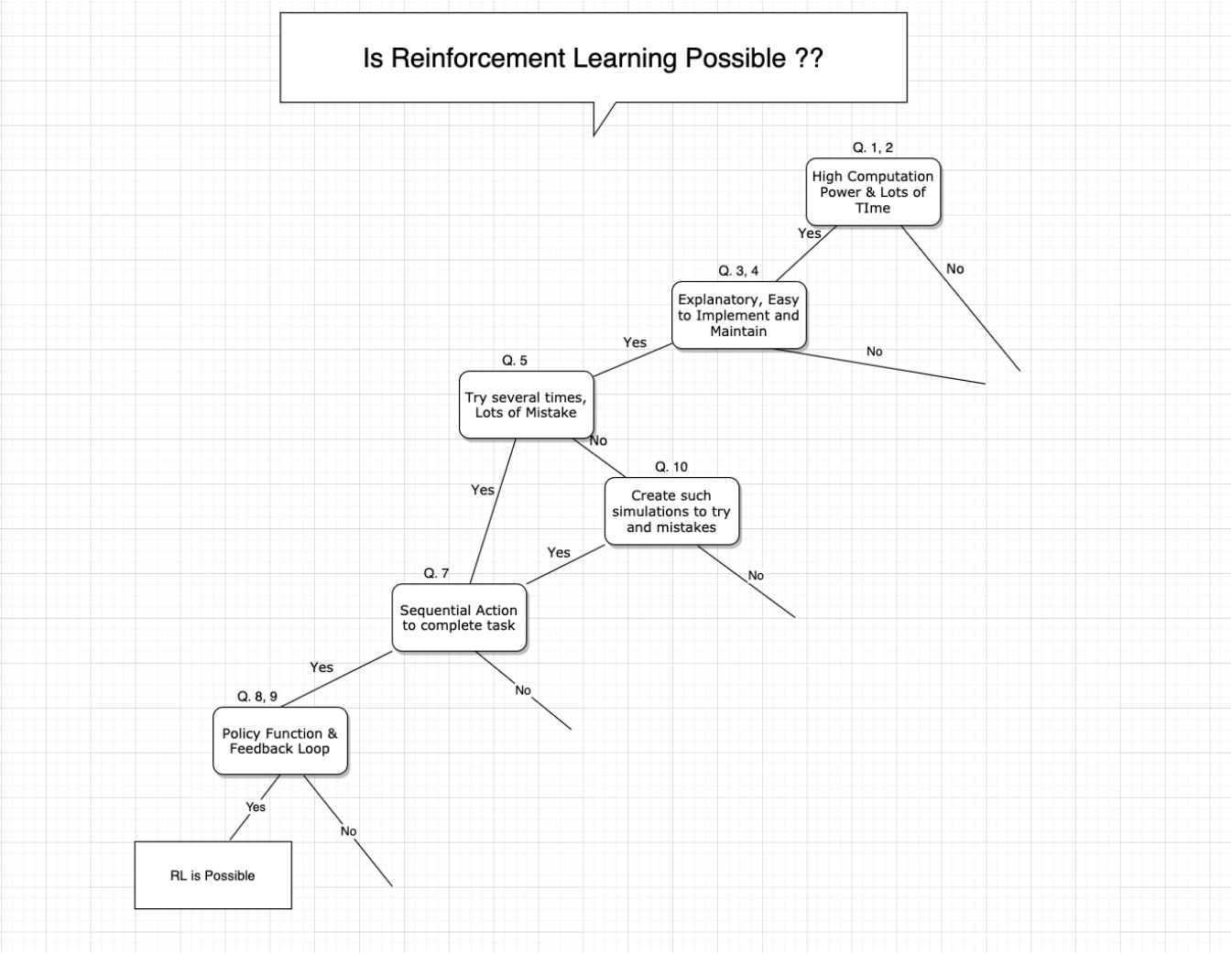
- **Value-based methods**
  - (Q-learning, Deep Q-learning): where we learn a value function **that will map each state action pair to a value.**
  - find the best action to take for each state — the action with the biggest value.
  - works well when you have a finite set of actions.
- **Policy-based methods**
  - REINFORCE with Policy Gradients
  - we directly optimize the policy without using a value function.
  - when the action space is continuous or stochastic.
  - use total rewards of the episode
  - problem is finding a good score function to compute how good a policy is
- **Hybrid Method**
  - **Actor-Critic Method**
    - Policy Learning + Value Learning
    - Policy Function Actor: Choses to make moves
    - Value Function Critic: Decides how the agent is performing
    - we make an update at each step (TD Learning)
    - Because we do an update at each time step, we can't use the total rewards R(t).
    - Both learn in parallel, like GANs
    - Not Stable but several variations which are stable

Algorithms

Name	Comments on Applicability	Reference
Q Learning		

Is RL Possible?

1. Do you have very high computation power?
2. Do you have lots of time to train an agent?
3. Do you need your model to be self-explanatory, humans can understand the reasoning behind the predictions and decisions made by the model?
4. Do you need your model to be easy to implement and maintain?
5. Is it possible to try the problem several times and afford to make many mistakes?
6. In your situation, do active and online learning of algorithms is possible i.e while learning by actions, explore new data space and then learn from such conditions and data?
7. In your situation, Can the algorithm take sequential action and complete the task?
8. Is it possible to define *policy function*, actions that the agent takes as a function of the agent's state and the environment.?
9. Is it possible to define a function to receive feedback from actions, such that feedback helps to learn and take new action?
10. Can you simulate an environment for the task so that algorithm can try lots of times and can make mistakes to learn?



Model-Based vs Model Free :

Do you have a probability function that helps you to select new actions based on current action?

If a model is not available, Is it possible to train such a model?