

Machine Learning with Python



Hello!

I am Eslam Ahmed

I am a software engineer.

You can find me at jeksogsa@gmail.com



Hello!

I am Eman Ehab

I am a ML research engineer.

You can find me at
emanehab.ieee@gmail.com



Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

Reinforcement Learning

Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

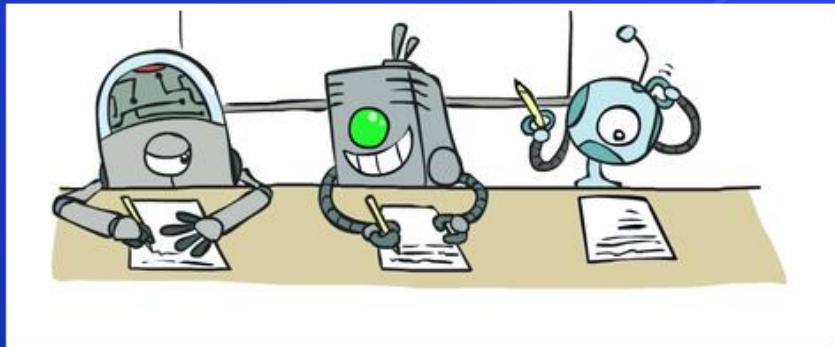
Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

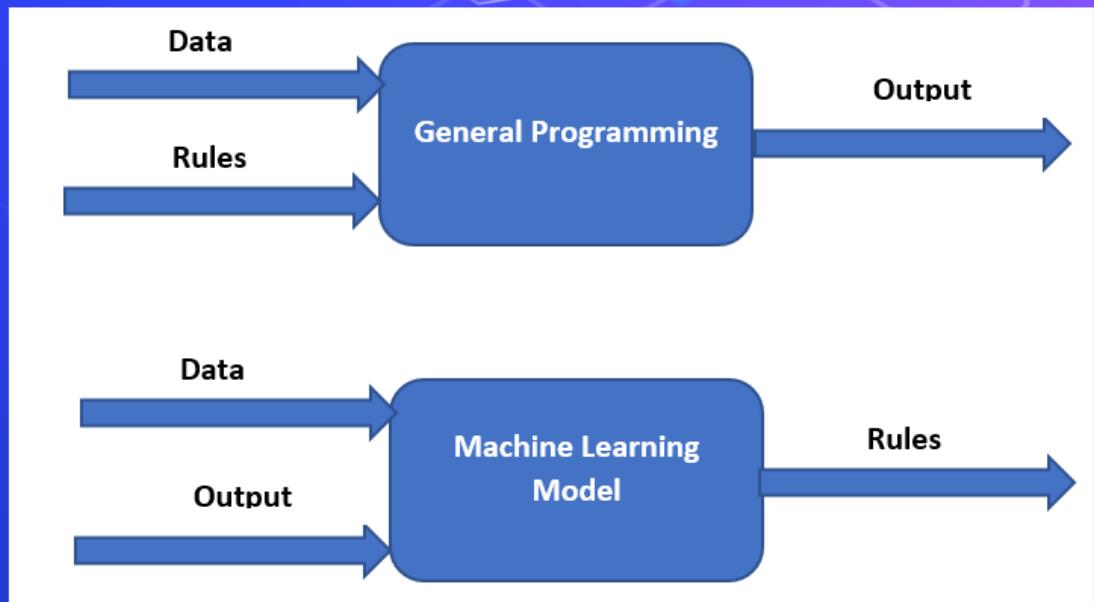
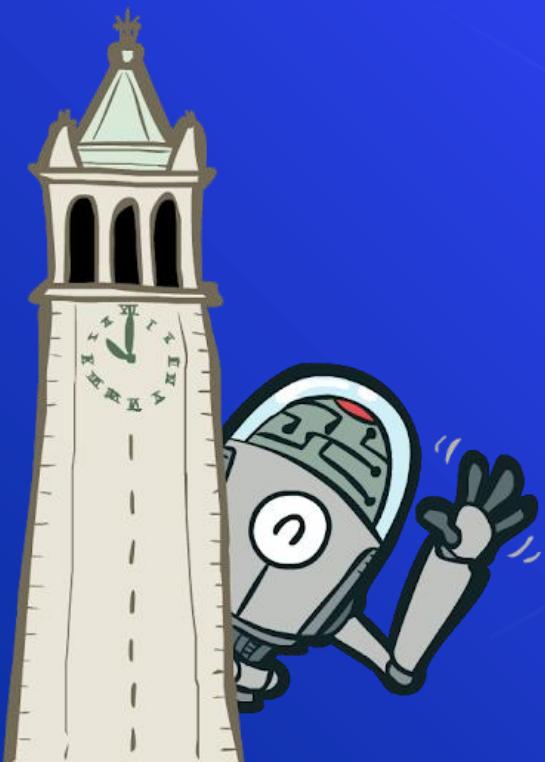
Reinforcement Learning

Machine Learning

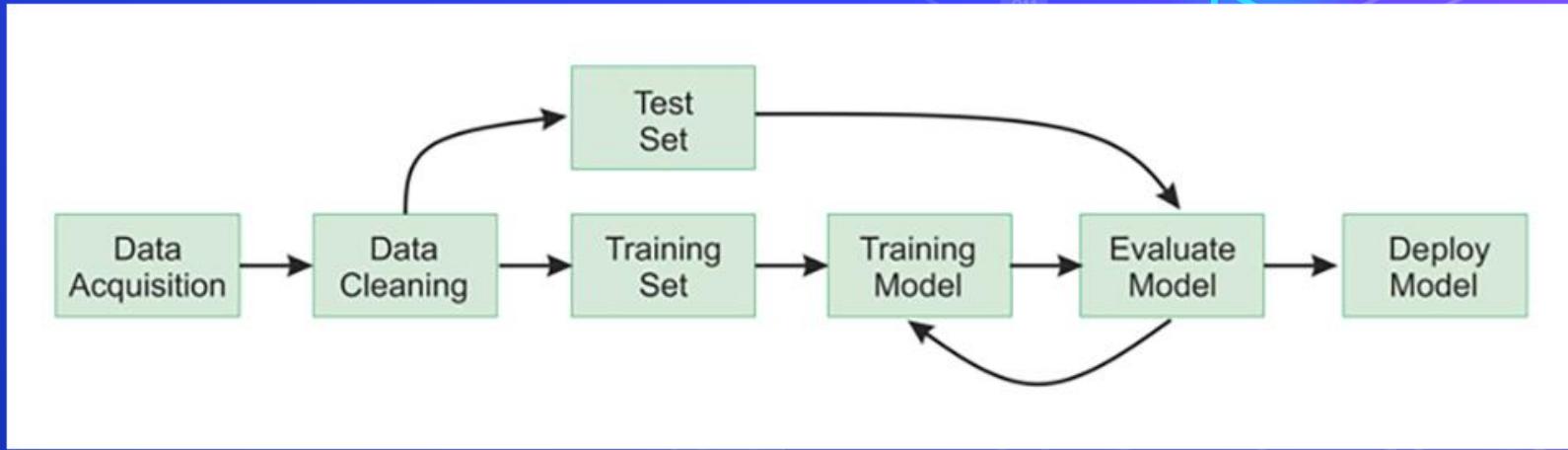
Machine learning involves computers discovering how they can perform tasks without being explicitly programmed to do so. It involves computers learning from data provided so that they carry out certain tasks.



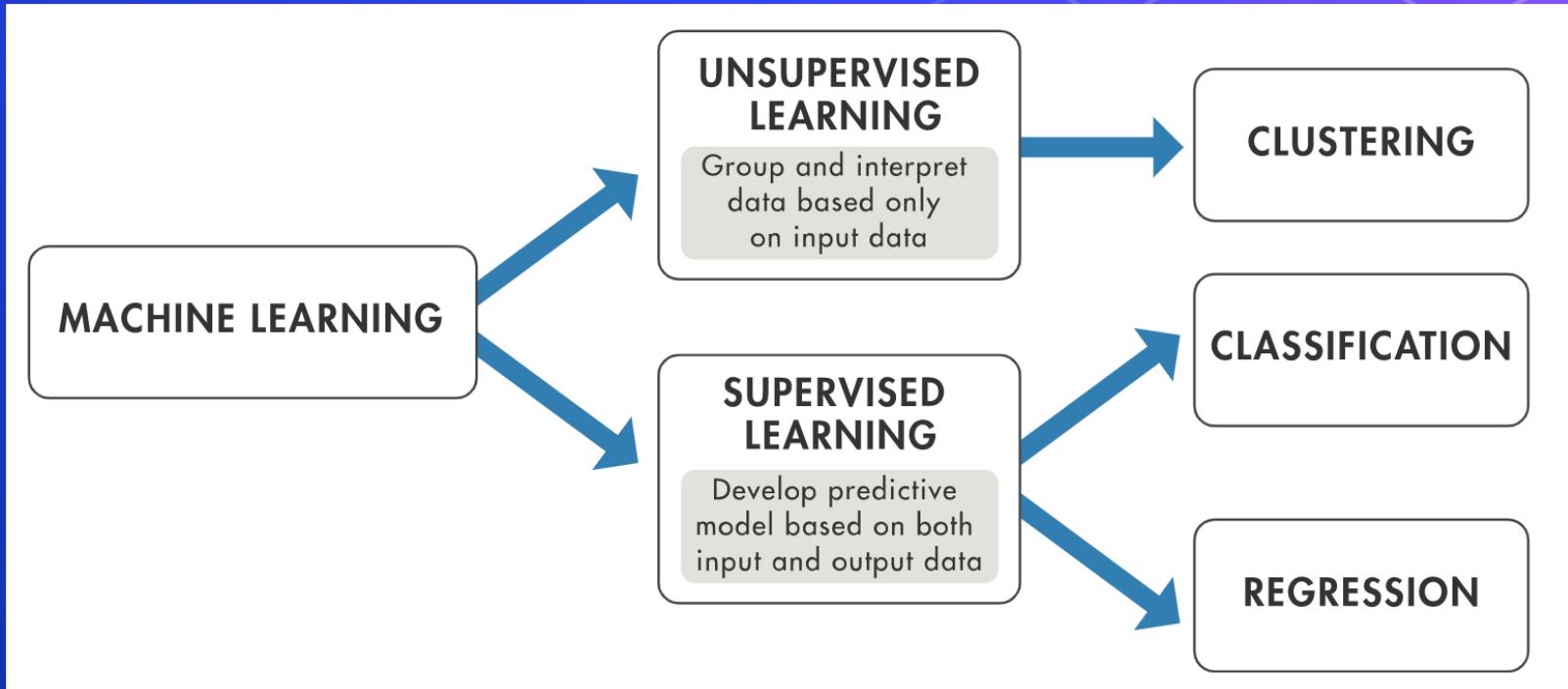
Machine Learning



Machine Learning



Machine Learning



Machine Learning

Classical Machine Learning

Task Driven

Supervised Learning

(Pre Categorized Data)



Classification

(Divide the socks by Color)

Eg. Identity Fraud Detection



Regression

(Divide the Ties by Length)

Eg. Market Forecasting

Data Driven

Unsupervised Learning

(Unlabelled Data)



Clustering

(Divide by Similarity)

Eg. Targeted Marketing



Association

(Identify Sequences)

Eg. Customer Recommendation



Dimensionality Reduction

(Wider Dependencies)

Eg. Big Data Visualization

Obj: Predictions & Predictive Models

Pattern/ Structure Recognition



Machine Learning

- Supervised learning : Task Driven (Classification, Regression)
- Unsupervised learning : Data Driven (Clustering)
- Reinforcement learning :
 - Close to human learning.
 - Algorithm learns a policy of how to act in a given environment.
 - Every action has some impact in the environment, and the environment provides rewards that guides the learning algorithm.

Machine Learning

Supervised Learning

Regression

- Simple Linear Regression
- Multiple Linear Regression
- Polynomial Regression
- Evaluating Model Performance

Classification

- Logistic Regression
- K-Nearest Neighbors (KNN)
- Naive Bayes
- SVM
- Decision Trees
- Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
- Evaluating Model Performance

Unsupervised Learning

- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

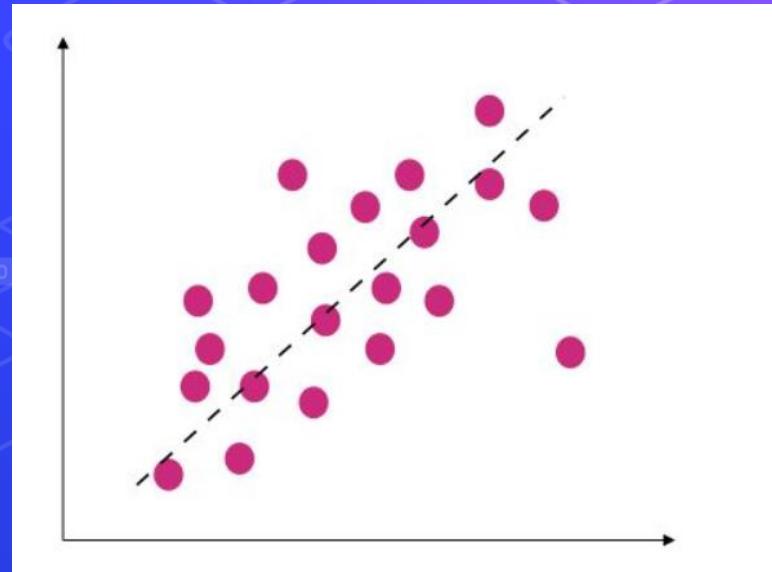
Reinforcement Learning

Regression

Regression quantifies the relationship between one or more predictor variable(s) and one outcome variable.

For example,
it can be used to quantify the relative impacts of age, gender,
and diet (the predictors) on weight (the output or dependent).

- House prices based on size, locations,...
- Salary prediction based on experience
- Stock Market prediction
- Weather prediction
- ...



Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

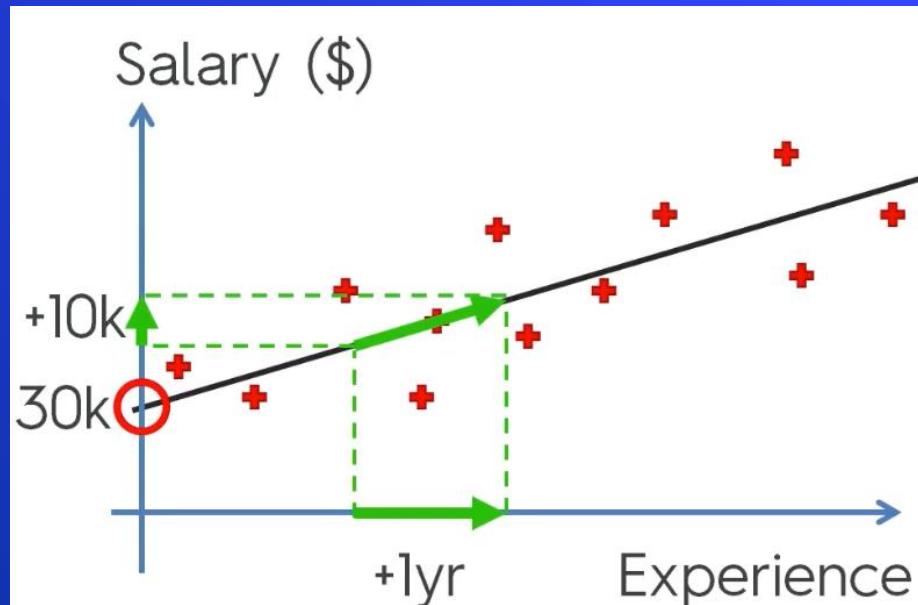
- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

Reinforcement Learning

Simple Linear Regression (Equation of a straight line)



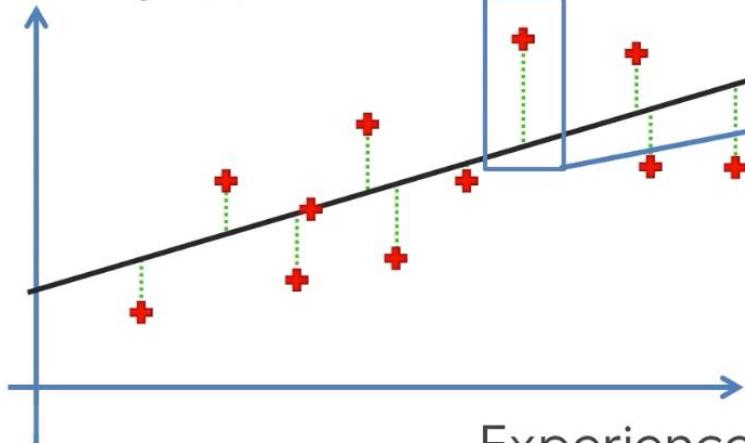
$$y = b_0 + b_1 * x$$

$$\text{Salary} = b_0 + b_1 * \text{Experience}$$

Simple Linear Regression (Calculate Cost Function)

Simple Linear Regression:

Salary (\$)



y_i

\hat{y}_i

Hypothesis:
$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

Parameters: θ_0, θ_1

Cost Function:
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Goal:
$$\underset{\theta_0, \theta_1}{\text{minimize}} J(\theta_0, \theta_1)$$

Simple Linear Regression (Example)

Theta0 = 5 , theta 1 = 2

Equation $h(x) = 5 + 2x$

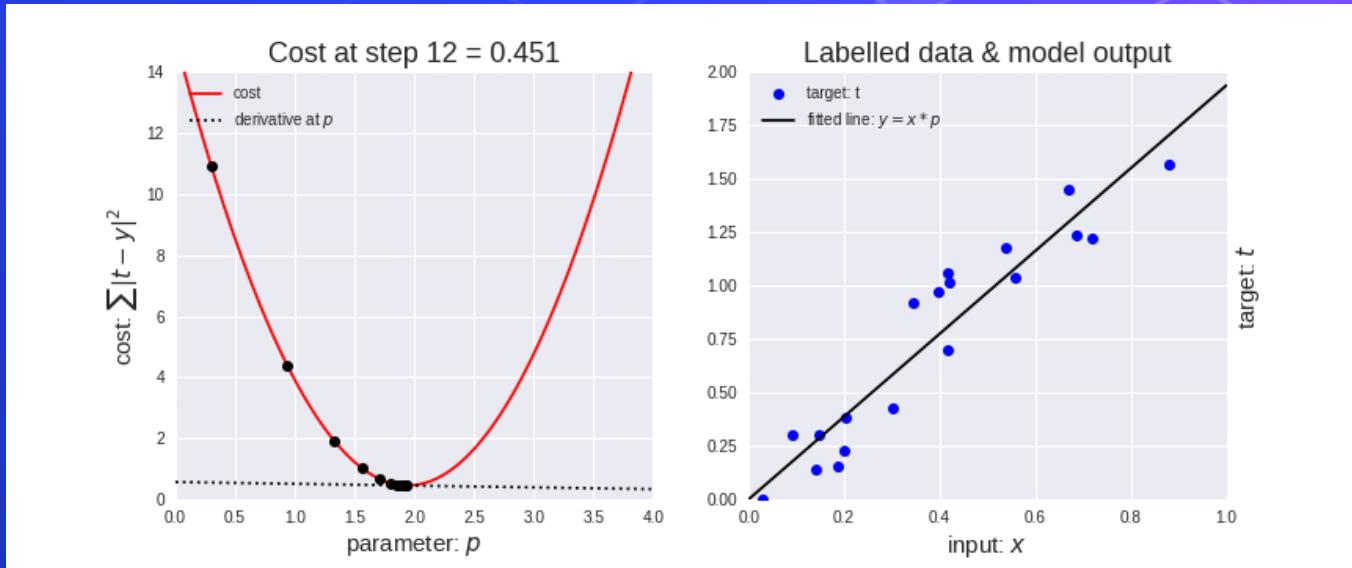
X	Y	$h(x)$	$h(x) - y$	$(h(x) - y)^2$
1	7	7	0	0
2	8	9	1	1
2	7	9	2	4
3	9	11	2	4
4	11	13	2	4
5	10	15	5	25
5	12	15	3	9

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J = 1 / 14 (0+1+4+4+4+25+9)$$

$$J = 47/14 = 3.3$$

Simple Linear Regression (Minimize cost using Gradient Descent)



repeat until convergence:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x_i) - y_i)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m ((h_\theta(x_i) - y_i)x_i)$$

Simple Linear Regression (Minimize cost using Gradient Descent)

Cost Function

$$J(\Theta_0, \Theta_1) = \frac{1}{2m} \sum_{i=1}^m [h_\Theta(x_i) - y_i]^2$$

↑
Predicted Value ↑
True Value

Gradient Descent

$$\Theta_j = \Theta_j - \alpha \frac{\partial}{\partial \Theta_j} J(\Theta_0, \Theta_1)$$

↑
Learning Rate

NOW,

$$\begin{aligned}\frac{\partial}{\partial \Theta} J_\Theta &= \frac{\partial}{\partial \Theta} \frac{1}{2m} \sum_{i=1}^m [h_\Theta(x_i) - y_i]^2 \\&= \frac{1}{m} \sum_{i=1}^m (h_\Theta(x_i) - y) \frac{\partial}{\partial \Theta_j} (\Theta x_i - y) \\&= \frac{1}{m} (h_\Theta(x_i) - y) x_i\end{aligned}$$

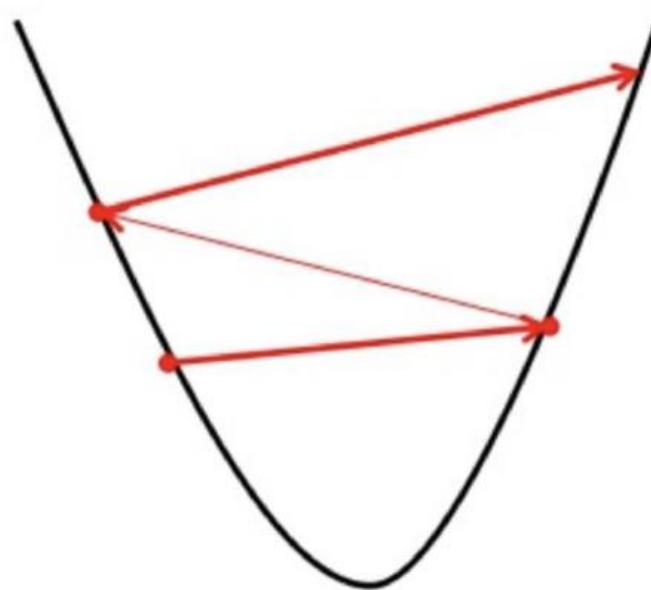
Therefore,

$$\Theta_j := \Theta_j - \frac{\alpha}{m} \sum_{i=1}^m [(h_\Theta(x_i) - y) x_i]$$

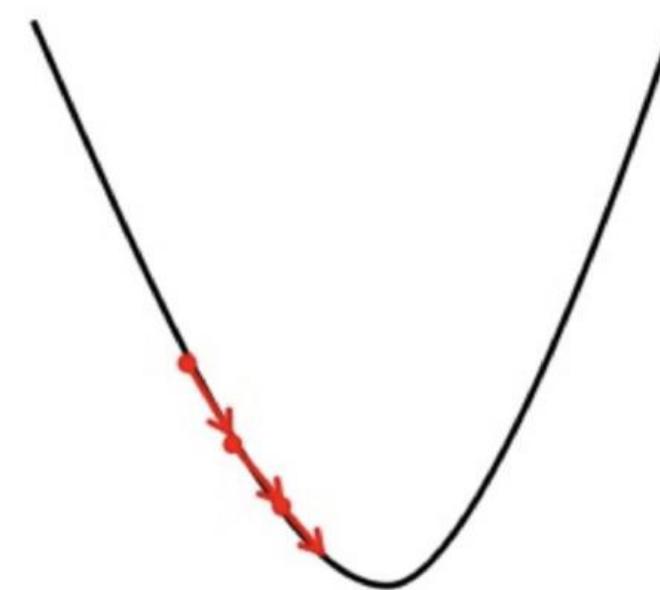


Simple Linear Regression (Alpha constant)

Big learning rate



Small learning rate



Simple Linear Regression (Code)

```
 1 from sklearn.linear_model import LinearRegression  
 2  
 3 # make model object  
 4 model = LinearRegression()  
 5  
 6 # train  
 7 model.fit(x_train, y_train)  
 8  
 9 # test  
10 model.predict(x_test)  
11  
12 # calculate R2 score on training data  
13 model.score(x_train, y_train)  
14  
15 # calculate R2 score on testing data  
16 model.score(x_test, y_test)  
17  
18 # get model parameters  
19 model.coef_  
20 model.intercept_
```



Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

Reinforcement Learning

Multiple Linear Regression (Equation)

Simple
Linear
Regression

$$y = b_0 + b_1 * x_1$$

Multiple
Linear
Regression

Dependent variable (DV) Independent variables (IVs)

$$y = b_0 + b_1 * x_1 + b_2 * x_2 + \dots + b_n * x_n$$

Constant Coefficients

```
graph TD; DV[Dependent variable DV] --> y; IVs[Independent variables IVs] --> x1; IVs --> x2; IVs --> xn; Constant[Constant] --> b0; Coefficients[Coefficients] --> b1x1; Coefficients --> b2x2; Coefficients --> bnxn;
```

Multiple Linear Regression (Update theta values)

repeat until convergence: {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)}$$

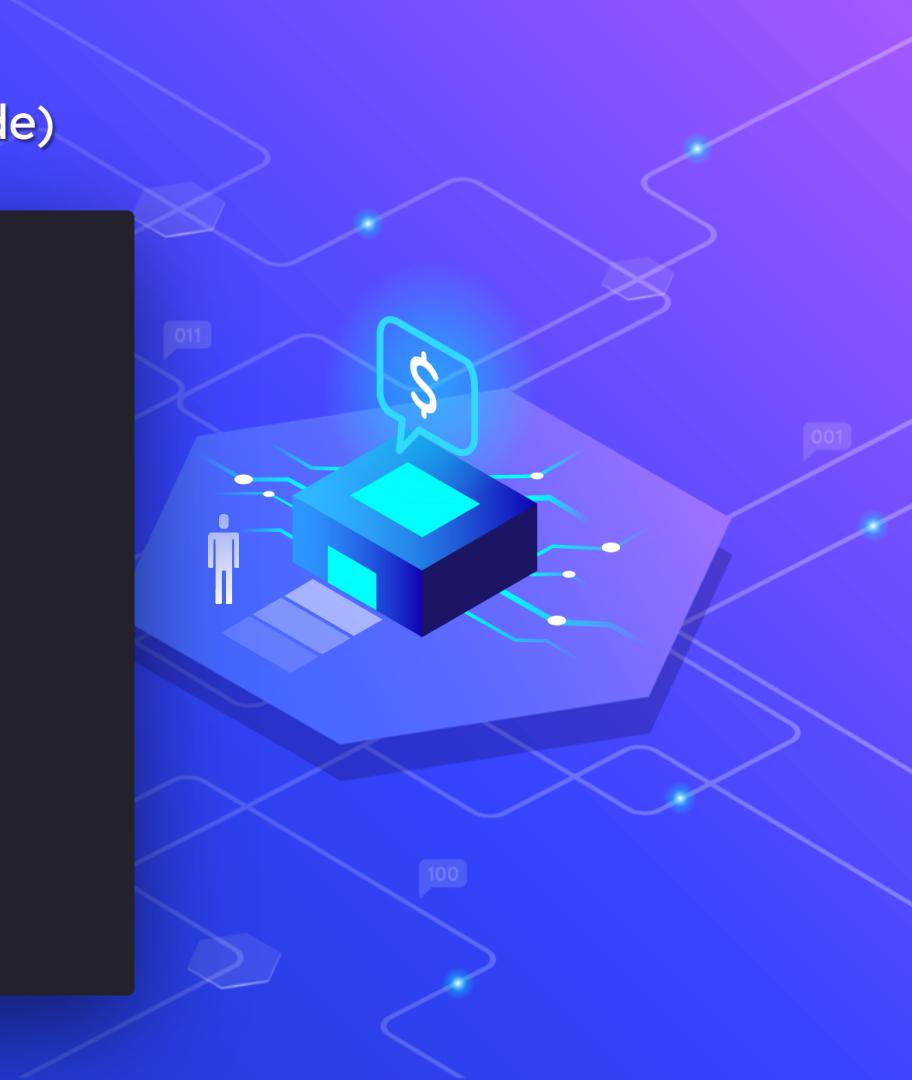
$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_2^{(i)}$$

...

}

Multiple Linear Regression (Code)

```
 1 from sklearn.linear_model import LinearRegression  
 2  
 3 # make model object  
 4 model = LinearRegression()  
 5  
 6 # train  
 7 model.fit(x_train, y_train)  
 8  
 9 # test  
10 model.predict(x_test)  
11  
12 # calculate R2 score on training data  
13 model.score(x_train, y_train)  
14  
15 # calculate R2 score on testing data  
16 model.score(x_test, y_test)  
17  
18 # get model parameters  
19 model.coef_  
20 model.intercept_
```



Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - **Polynomial Regression**
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

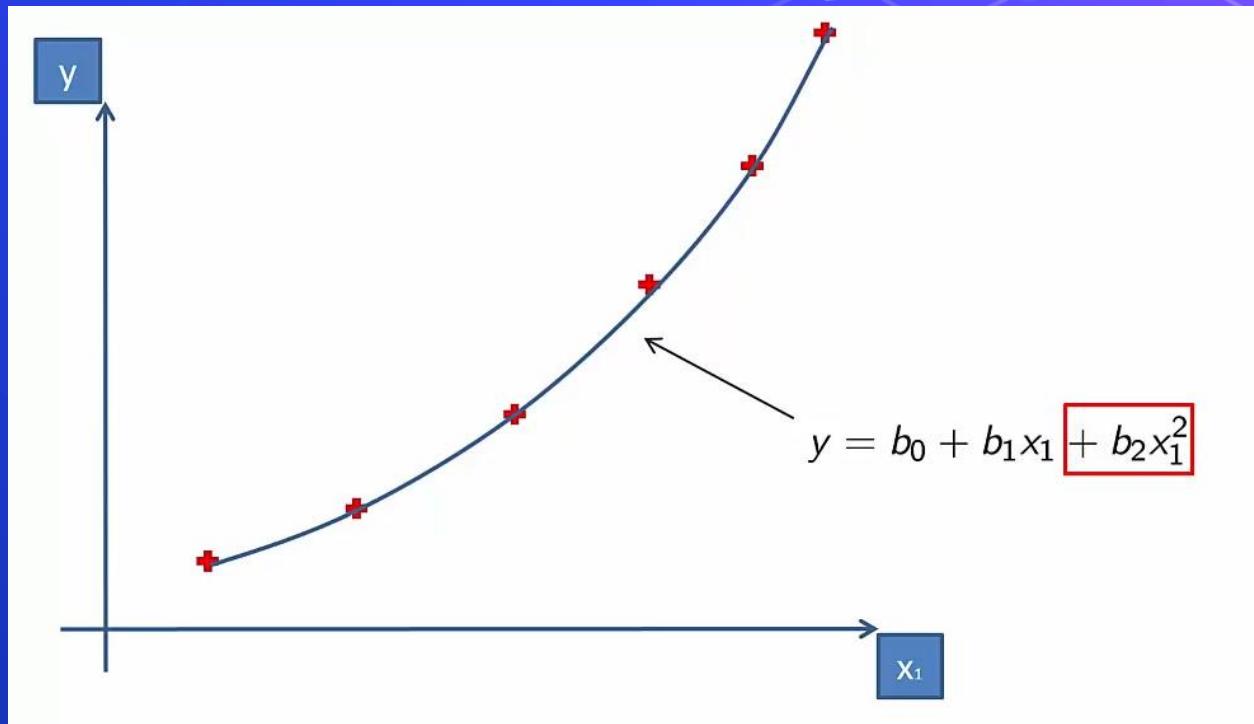
- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

Reinforcement Learning

Polynomial Regression (Poly equation)



Polynomial Regression (Poly equation)

Polynomial
Linear
Regression

$$y = b_0 + b_1x_1 + b_2x_1^2 + \dots + b_nx_1^n$$

Polynomial Regression (Code)

```
 1 from sklearn.linear_model import LinearRegression
 2 from sklearn.preprocessing import PolynomialFeatures
 3
 4 # create poly features for a feature
 5 poly = PolynomialFeatures(degree=2)
 6 x_train_poly = poly.fit_transform(x_train)
 7 x_test_poly = poly.fit_transform(x_test)
 8
 9 # make model object
10 model = LinearRegression()
11
12 # train
13 model.fit(x_train_poly, y_train)
14
15 # test
16 model.predict(x_test_poly)
17
18 # calculate R2 score on training data
19 model.score(x_train_poly, y_train)
20
21 # calculate R2 score on testing data
22 model.score(x_train_poly, y_train)
23
24 # get model parameters
25 model.coef_
26 model.intercept_
```



Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - **Evaluating Model Performance**
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

Reinforcement Learning

Evaluating Model Performance (R²)

Hrs Studied (X)	Marks (Y)
0	40
2	52
3	53
4	55
4	56
5	72
6	71
6	88
7	56
7	74
8	89
9	67
9	89
5.38	66.31
Mean	

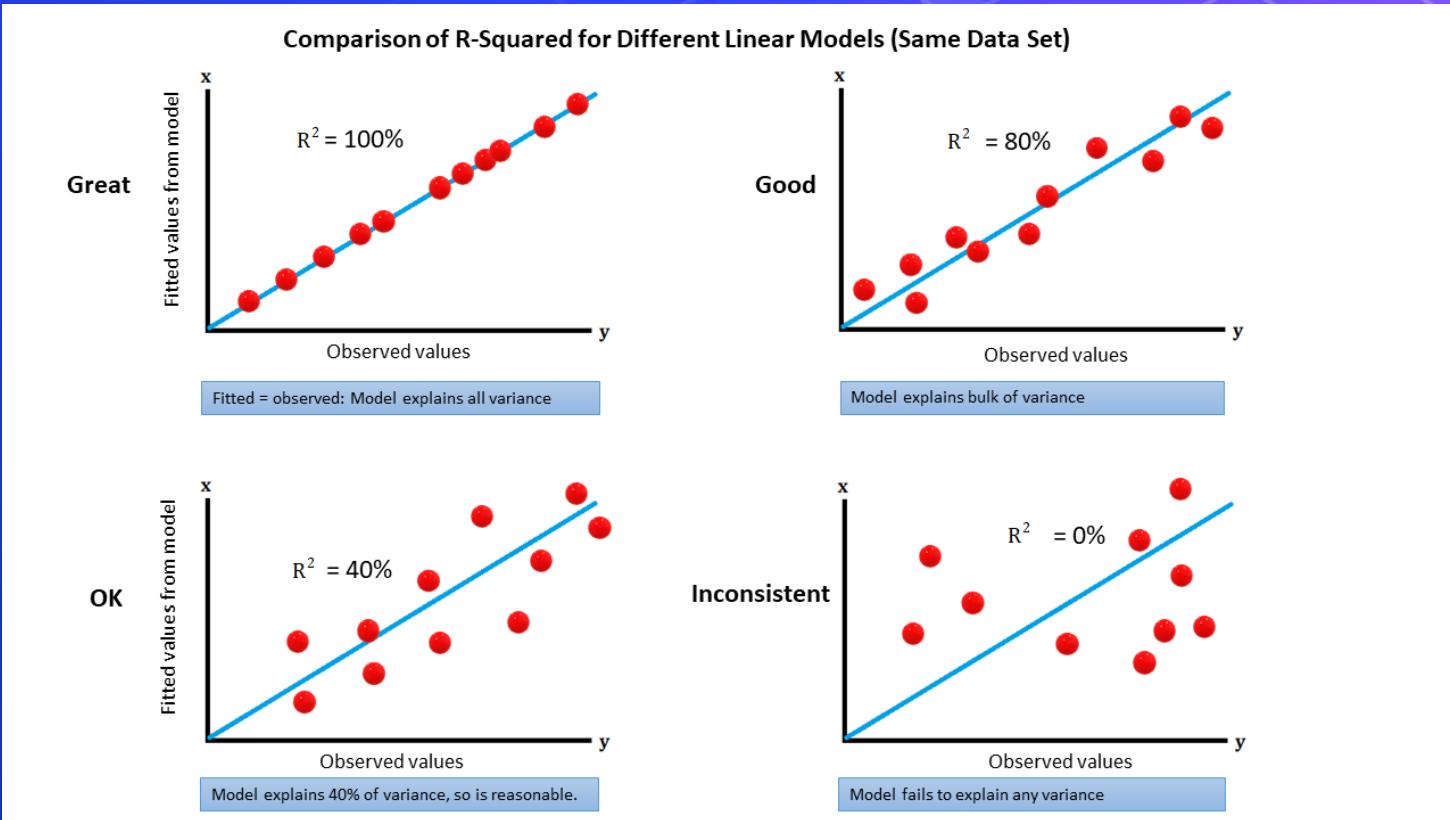
$$\begin{aligned} R^2 &= \text{SSR/SST} \\ &= 1844.12/3028.77 \\ &= 0.60886 \end{aligned}$$

Higher the value → Variation in Y is explained by variation in X.

$$\hat{y} \rightarrow y \quad \text{SSR} \rightarrow \text{SST} \quad R^2 \approx 1$$

$(Y - \bar{Y})^2$	$(\hat{Y} - \bar{Y})^2$
692.22	600.74
204.78	237.47
177.16	117.94
127.92	39.82
106.30	39.82
32.38	3.10
22.00	7.78
470.46	7.78
106.30	53.88
59.14	53.88
514.84	141.37
0.48	270.27
514.84	270.27
3028.77	1844.12
SST	SSR

Evaluating Model Performance (R^2)



Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

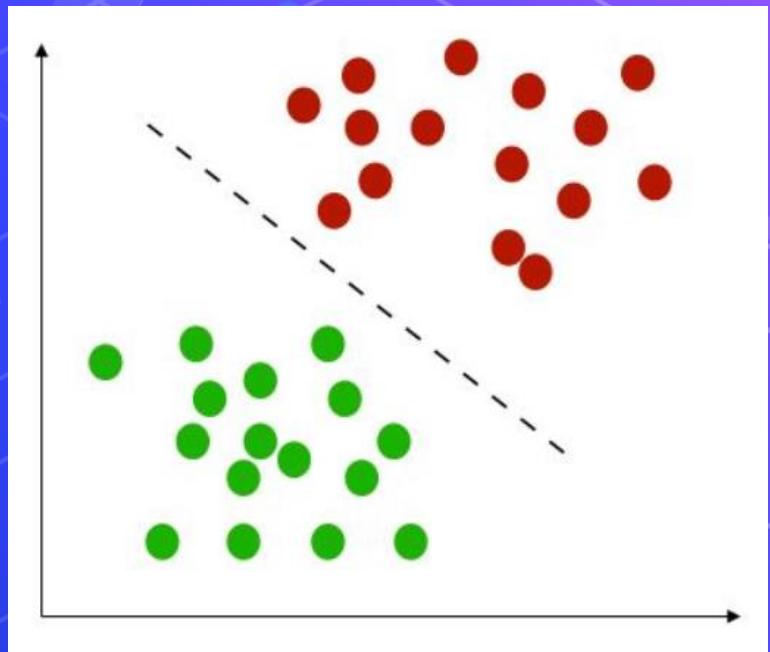
Reinforcement Learning

Classification

Classification specifies the class to which data elements belong to and is best used when the output has finite and discrete values. It predicts a class for an input variable as well.

For example,
it can be used to classify the animal in an image, it's either a cat or a dog.

- Email spam detection
- Face classification
- Patient has cancer or not
- Fraud detection
- ...



Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - **Logistic Regression**
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

Reinforcement Learning

Logistic Regression

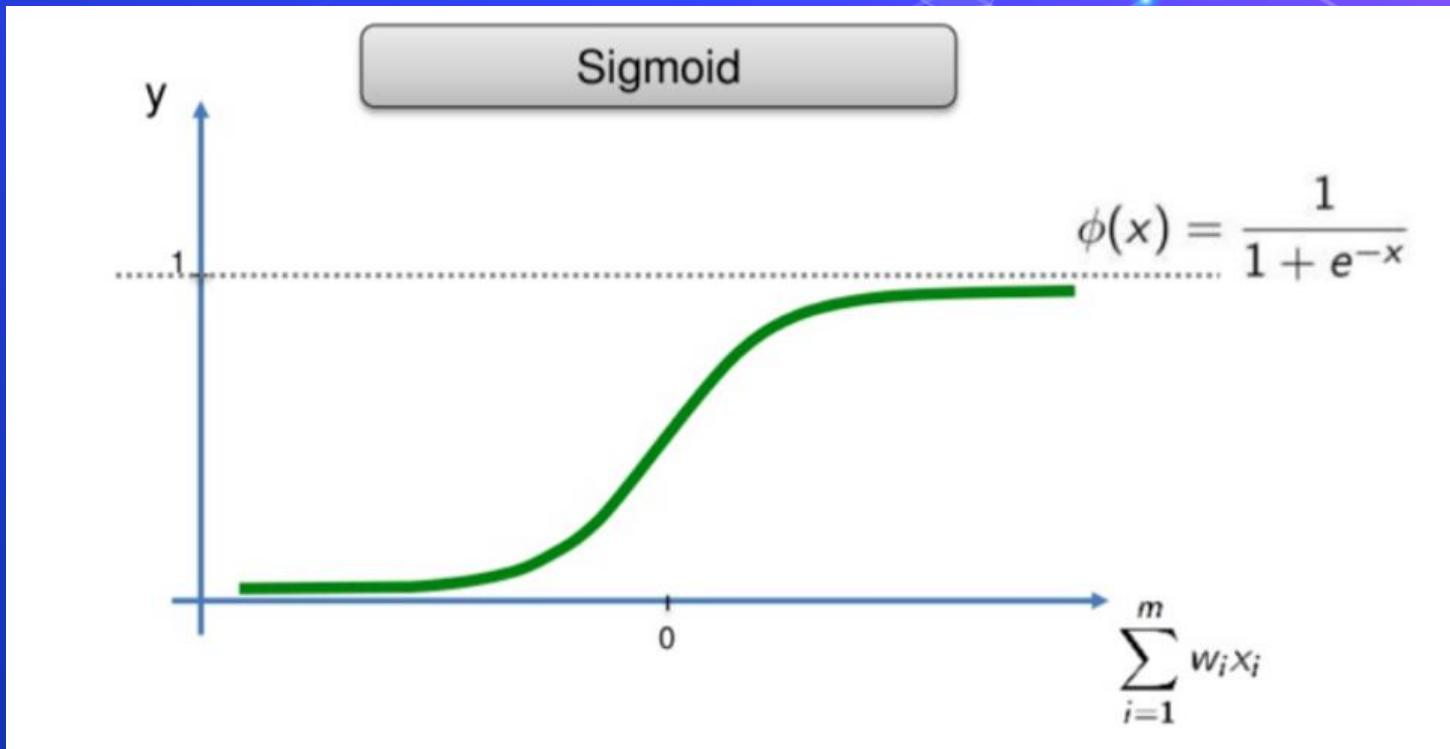
The Logistic Function

$$y = \frac{1}{1 + e^{-f(x_1, x_2, \dots, x_n)}} \in (0, 1)$$

where

$$f(x_1, x_2, \dots, x_n) = a_0 + a_1 x_1 + \dots + a_n x_n \in (-\infty, +\infty)$$

Logistic Regression



Logistic Regression

```
1 from sklearn.linear_model import LogisticRegression  
2 from sklearn.metrics import classification_report  
3 from sklearn.metrics import accuracy_score  
4 from sklearn.metrics import confusion_matrix  
5  
6  
7 model = LogisticRegression()  
8 model.fit(x_train, y_train)  
9 y_pred = model.predict(x_test)  
10  
11 print(confusion_matrix(y_test, y_pred))  
12 print(accuracy_score(y_test, y_pred))  
13 print(classification_report(y_test, y_pred))
```



Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - **K-Nearest Neighbors (KNN)**
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

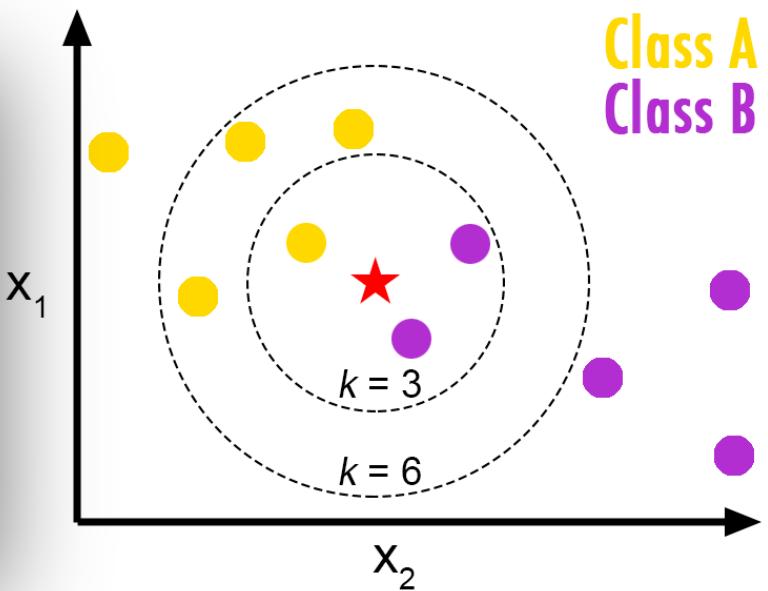
Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

Reinforcement Learning

K-Nearest Neighbors (KNN)

```
● ● ●  
1 from sklearn.neighbors import KNeighborsClassifier  
2 from sklearn.metrics import classification_report  
3 from sklearn.metrics import accuracy_score  
4 from sklearn.metrics import confusion_matrix  
5  
6  
7 model = KNeighborsClassifier()  
8 model.fit(x_train, y_train)  
9 y_pred = model.predict(x_test)  
10  
11 print(confusion_matrix(y_test, y_pred))  
12 print(accuracy_score(y_test, y_pred))  
13 print(classification_report(y_test, y_pred))
```



Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - **Naive Bayes**
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

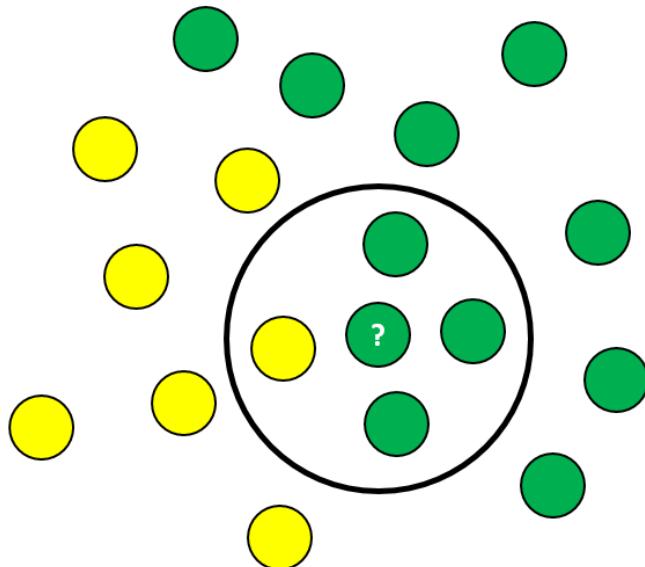
- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

Reinforcement Learning

Naive Bayes



$$P(\text{yellow}) = \frac{7}{17}$$

$$P(\text{green}) = \frac{10}{17}$$

$$P'(? | \text{green}) = \frac{3}{10}$$

$$P'(? | \text{yellow}) = \frac{1}{7}$$

prior probabilities

number of samples in a given class
divided by the total number of samples

we consider just the vicinity
of the new sample we want to classify

$$P''(\text{? is green}) = P(\text{green}) * P'(\text{?} | \text{green}) = \frac{10}{17} * \frac{3}{10} = \frac{30}{170}$$

$$P''(\text{? is yellow}) = P(\text{yellow}) * P'(\text{?} | \text{yellow}) = \frac{7}{17} * \frac{1}{7} = \frac{7}{119}$$

Naive Bayes



```
1 from sklearn.naive_bayes import MultinomialNB
2 from sklearn.metrics import classification_report
3 from sklearn.metrics import accuracy_score
4 from sklearn.metrics import confusion_matrix
5
6
7 model = MultinomialNB()
8 model.fit(x_train, y_train)
9 y_pred = model.predict(x_test)
10
11 print(confusion_matrix(y_test, y_pred))
12 print(accuracy_score(y_test, y_pred))
13 print(classification_report(y_test, y_pred))
```



Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - **SVM**
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

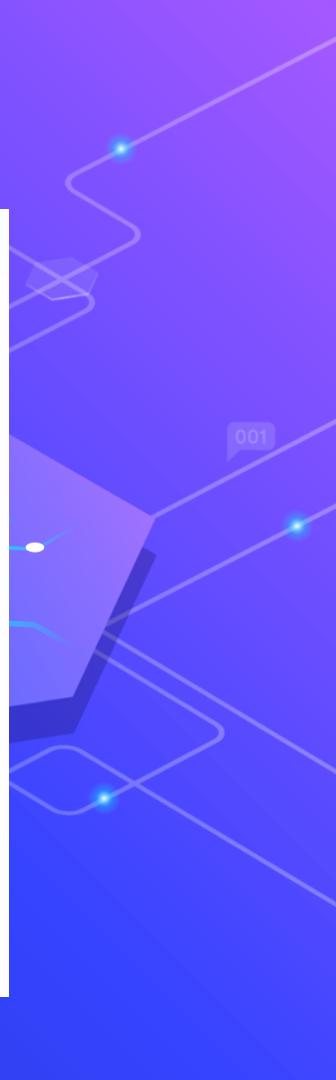
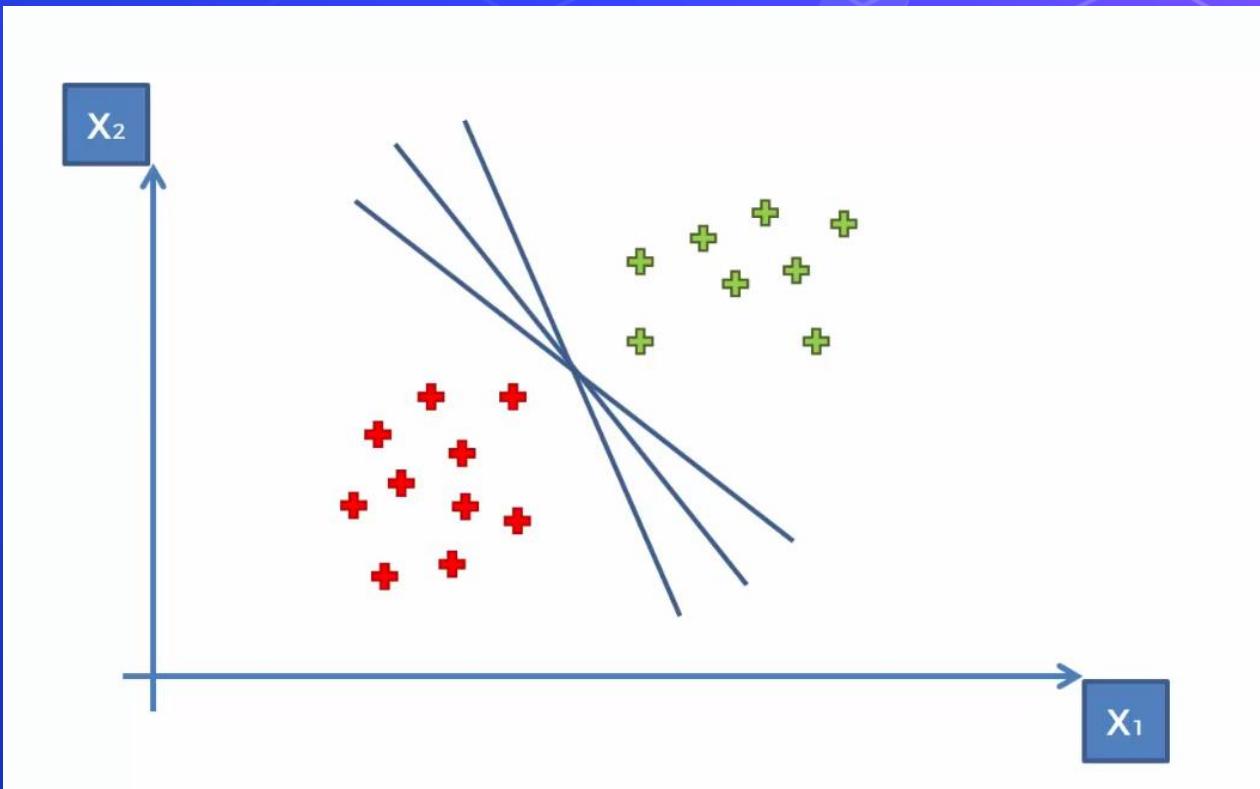
- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

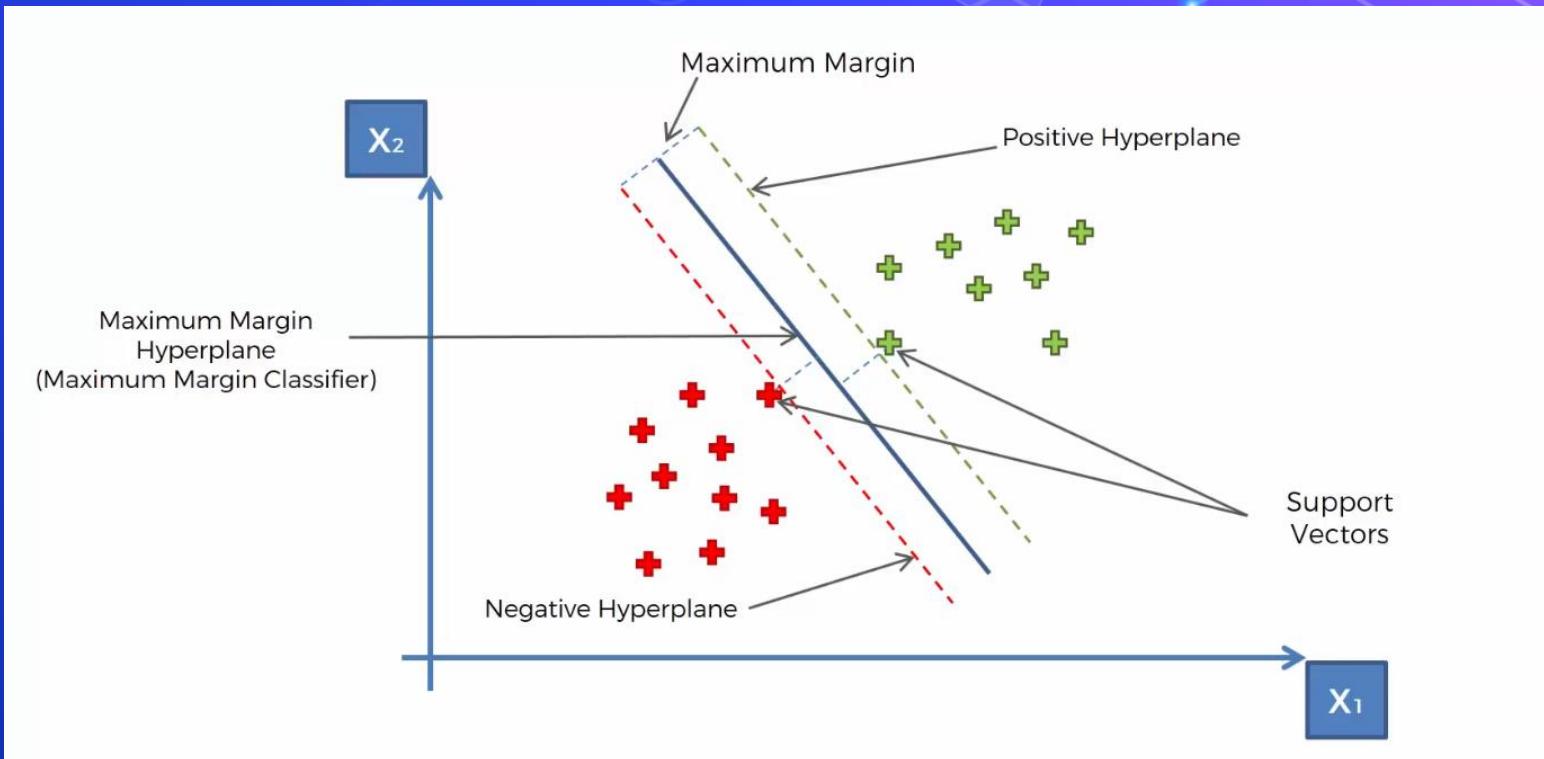
- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

Reinforcement Learning

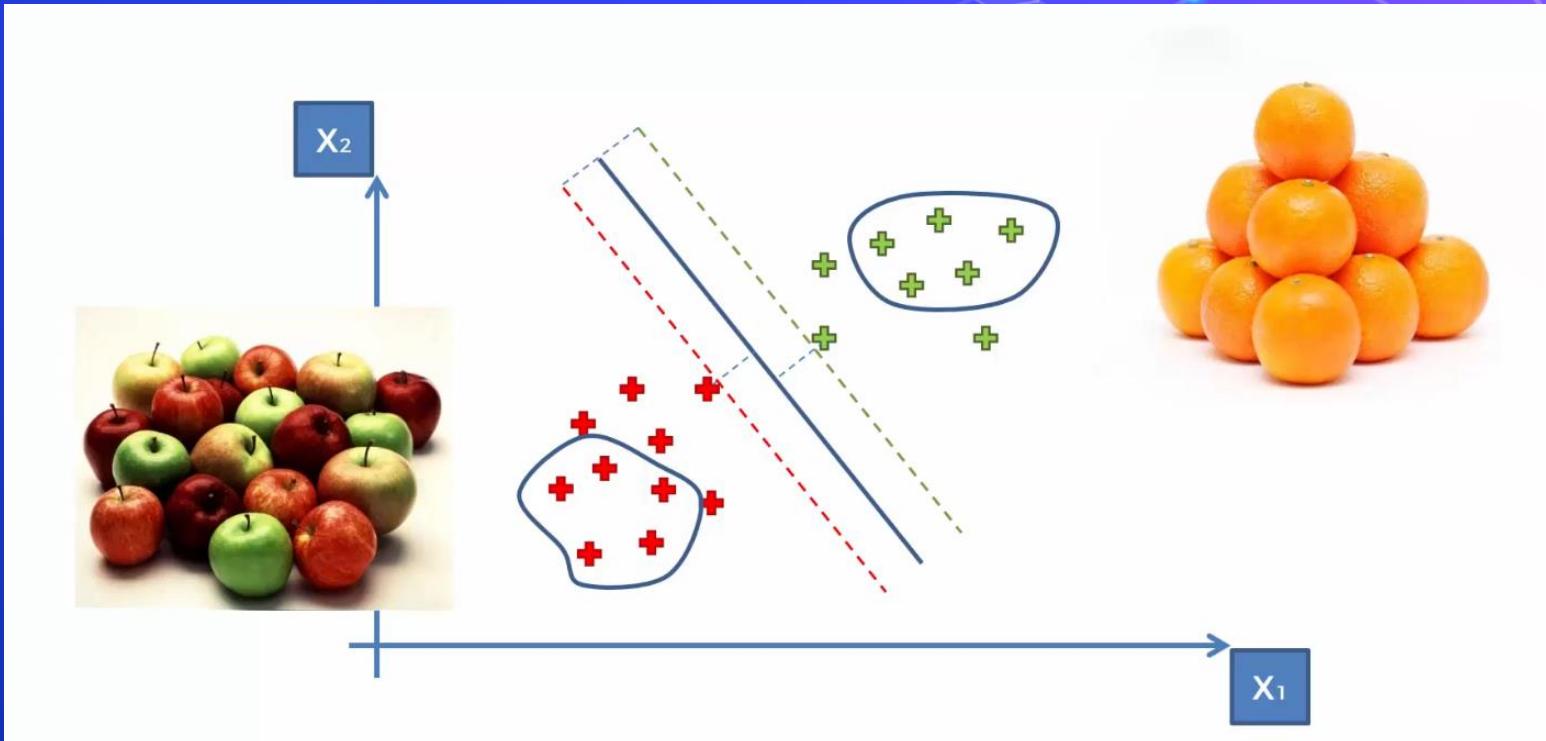
SVM (Support Vector Machine)



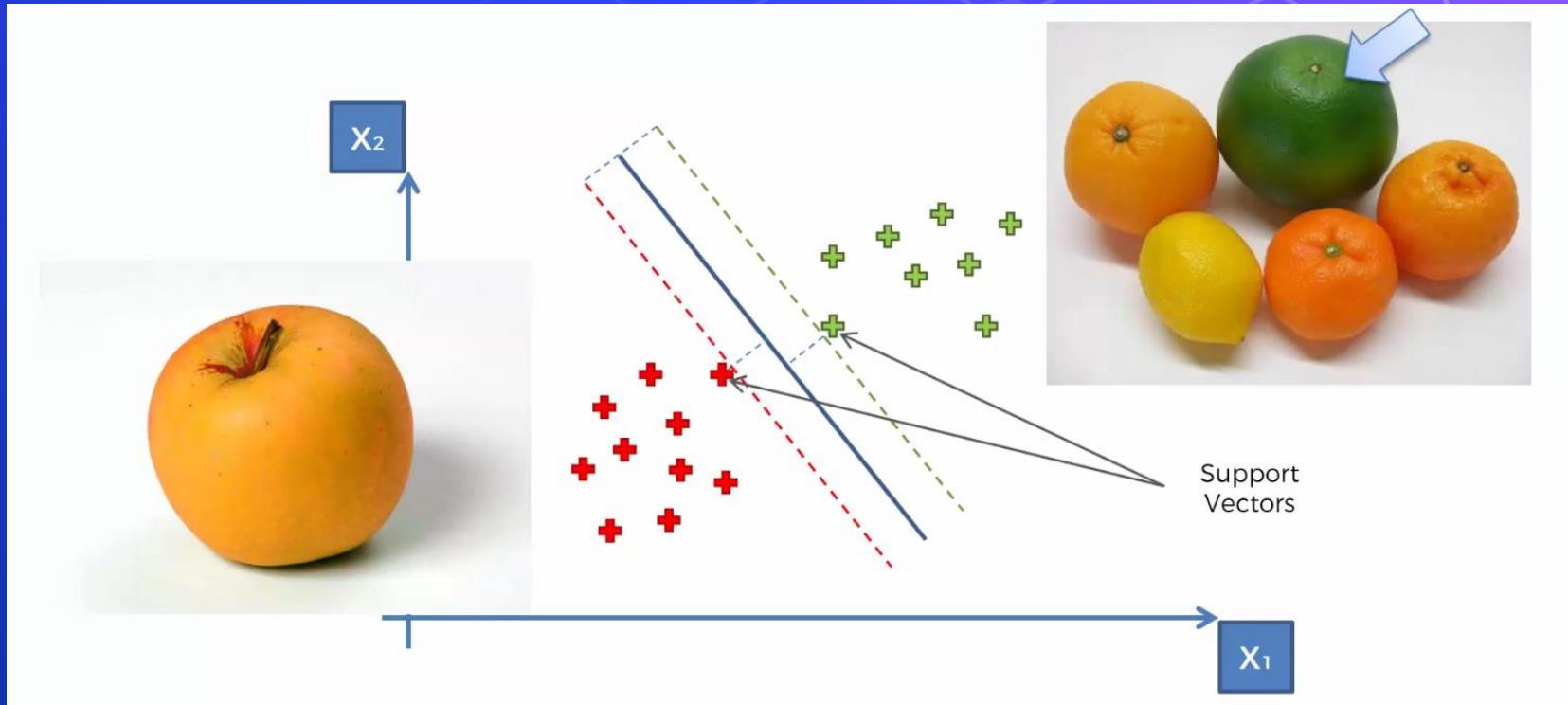
SVM (Support Vector Machine)



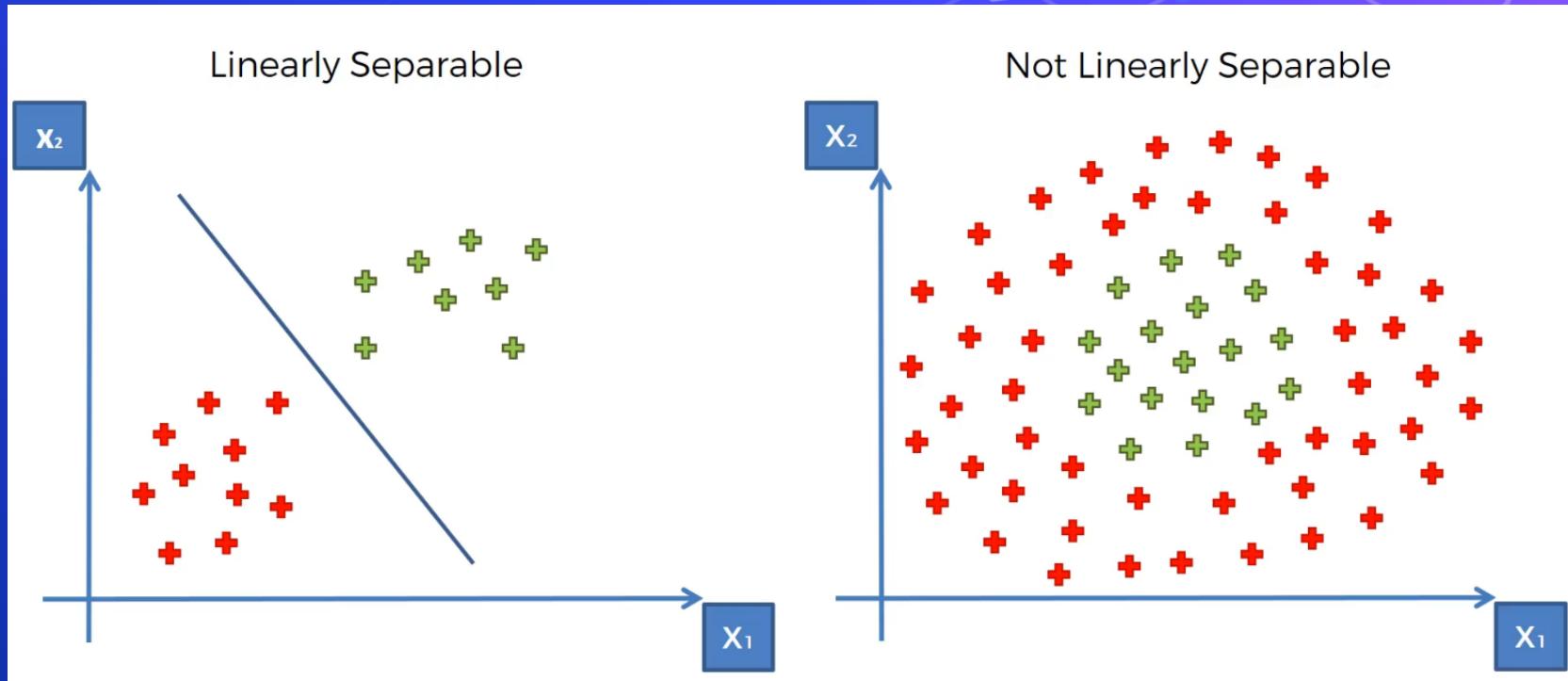
SVM (Support Vector Machine)



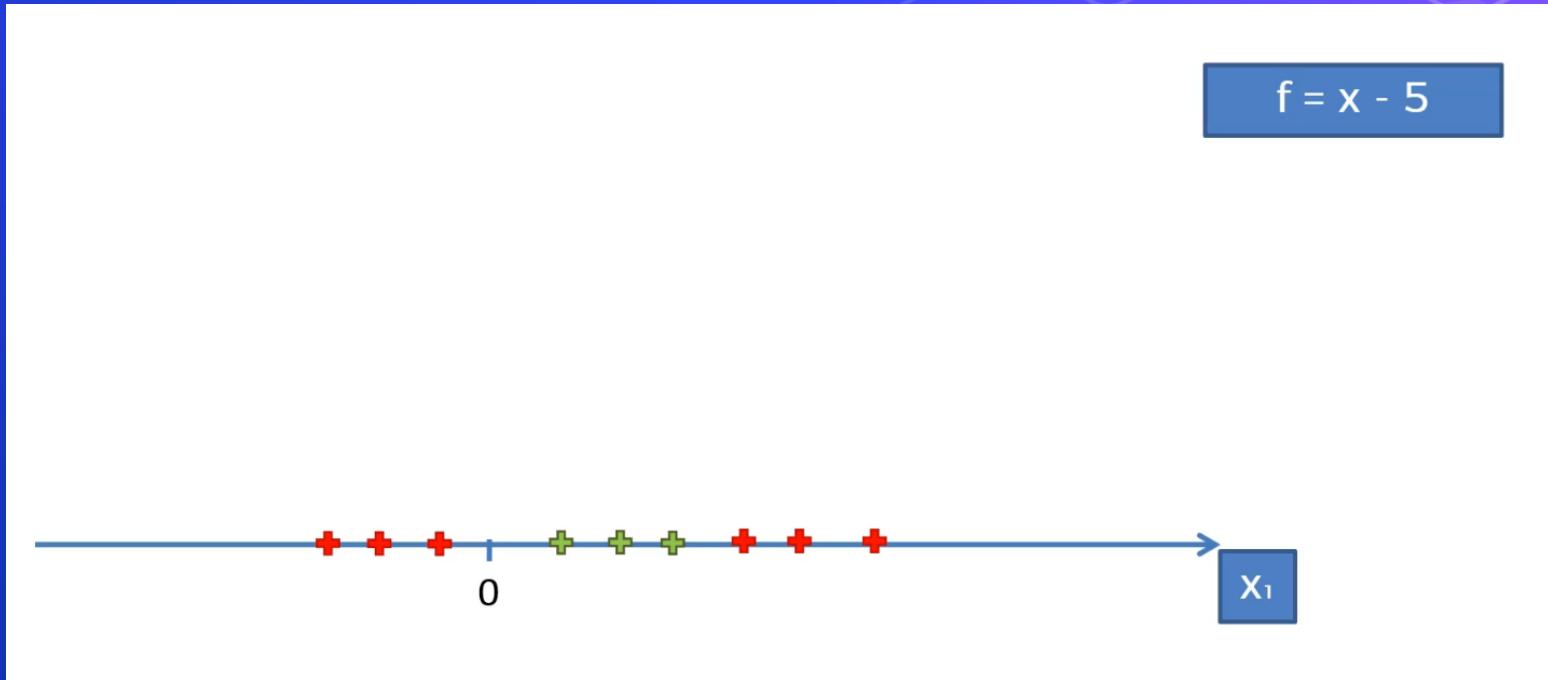
SVM (Support Vector Machine)



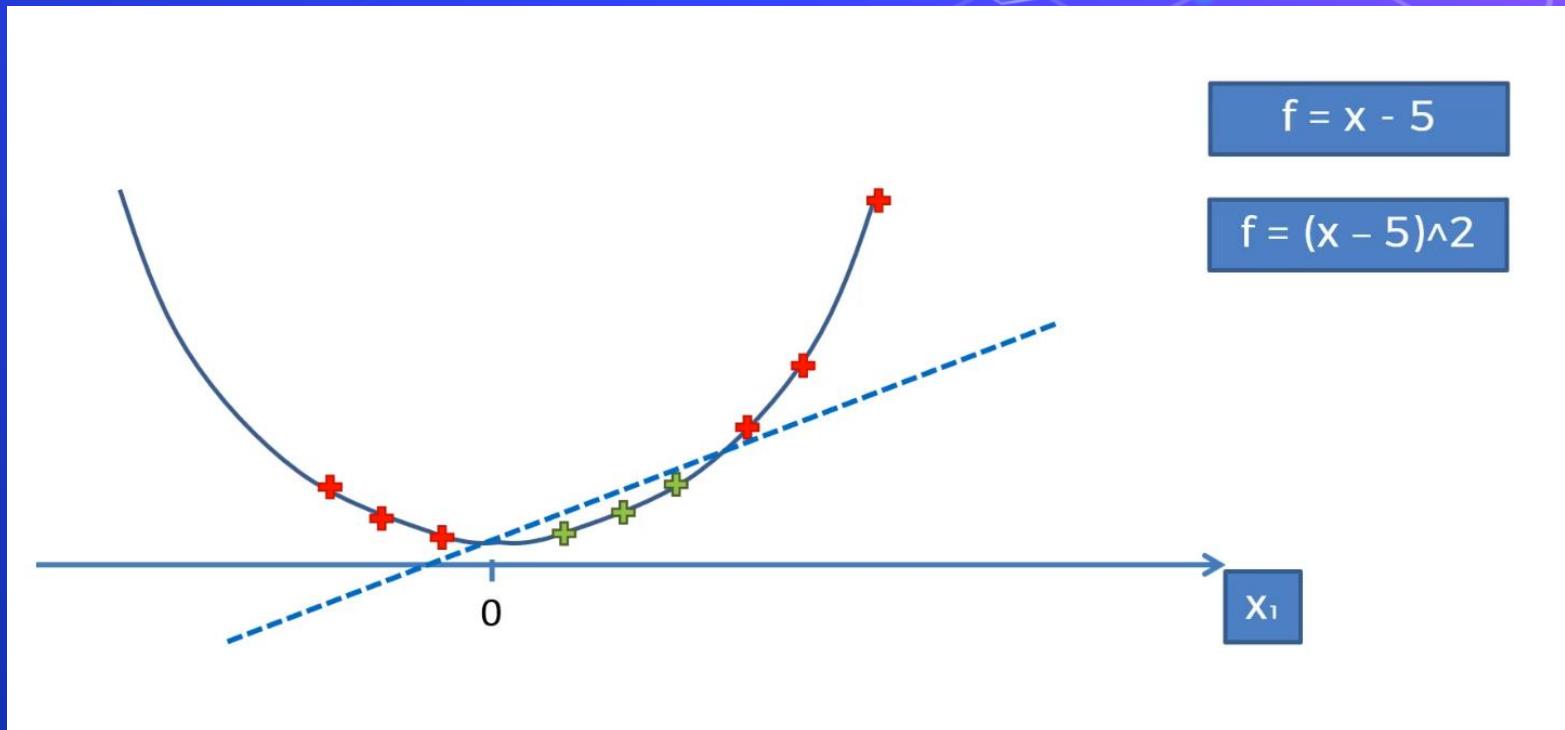
SVM (Support Vector Machine)



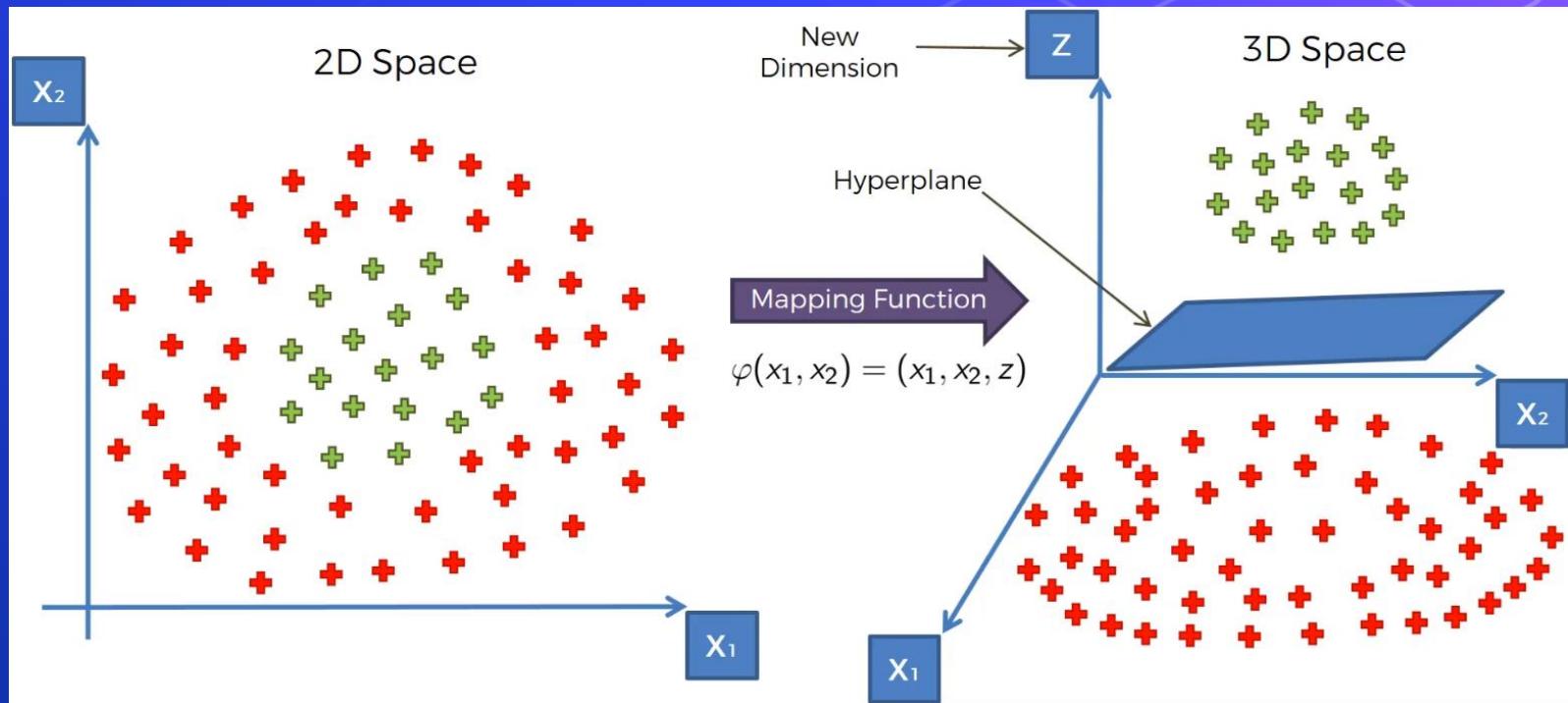
SVM (Support Vector Machine)



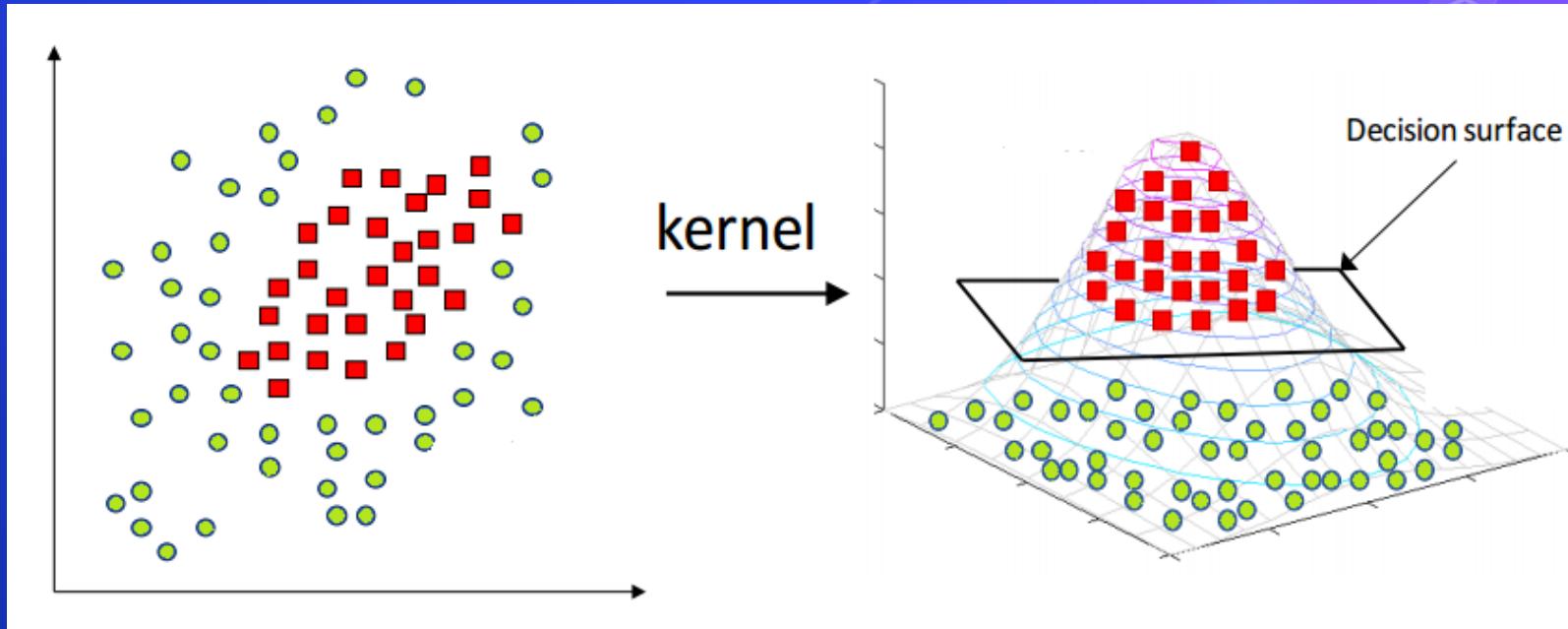
SVM (Support Vector Machine)



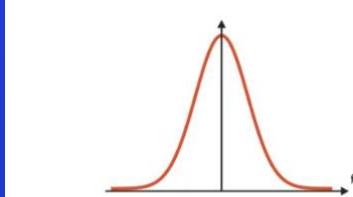
SVM (Support Vector Machine)



SVM (Support Vector Machine)

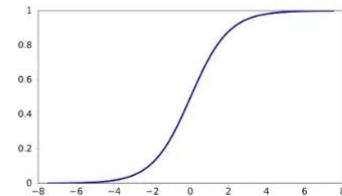


SVM (Support Vector Machine)



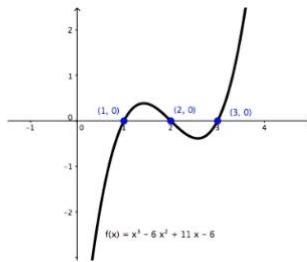
Gaussian RBF Kernel

$$K(\vec{x}, \vec{l}^i) = e^{-\frac{\|\vec{x}-\vec{l}^i\|^2}{2\sigma^2}}$$



Sigmoid Kernel

$$K(X, Y) = \tanh(\gamma \cdot X^T Y + r)$$



Polynomial Kernel

$$K(X, Y) = (\gamma \cdot X^T Y + r)^d, \gamma > 0$$

SVM (Support Vector Machine)

```
1 from sklearn.svm import SVC
2 from sklearn.metrics import classification_report
3 from sklearn.metrics import accuracy_score
4 from sklearn.metrics import confusion_matrix
5
6
7 model = SVC()
8 model.fit(x_train, y_train)
9 y_pred = model.predict(x_test)
10
11 print(confusion_matrix(y_test, y_pred))
12 print(accuracy_score(y_test, y_pred))
13 print(classification_report(y_test, y_pred))
```



Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - **Decision Trees**
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

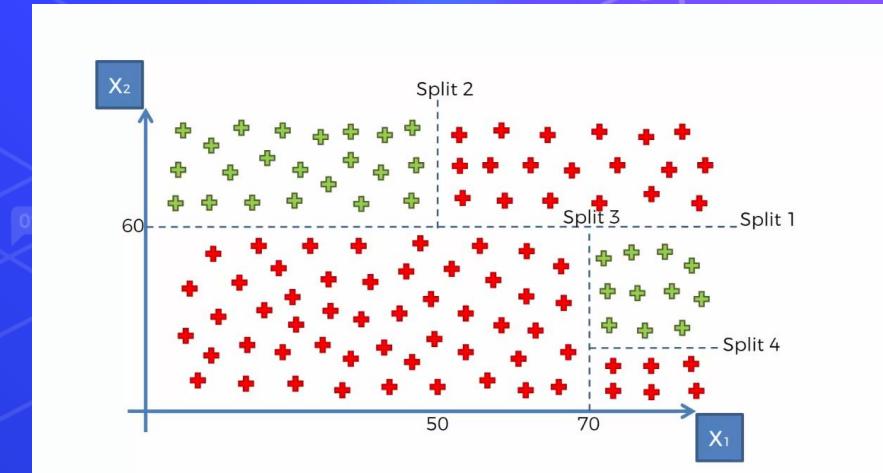
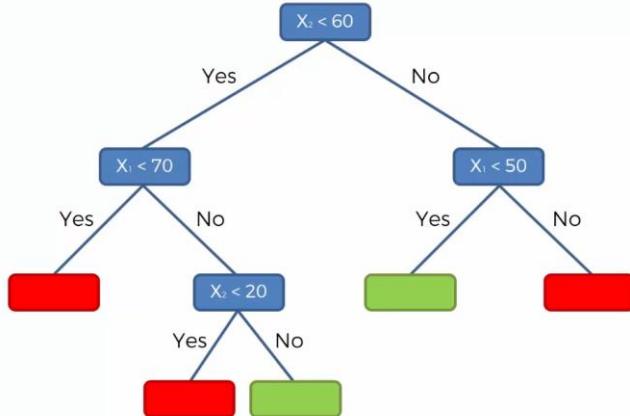
- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

Reinforcement Learning

Decision Trees



Decision Trees

```
1 from sklearn.tree import DecisionTreeClassifier  
2 from sklearn.metrics import classification_report  
3 from sklearn.metrics import accuracy_score  
4 from sklearn.metrics import confusion_matrix  
5  
6  
7 model = DecisionTreeClassifier()  
8 model.fit(x_train, y_train)  
9 y_pred = model.predict(x_test)  
10  
11 print(confusion_matrix(y_test, y_pred))  
12 print(accuracy_score(y_test, y_pred))  
13 print(classification_report(y_test, y_pred))
```



Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
- Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
- Evaluating Model Performance

Unsupervised Learning

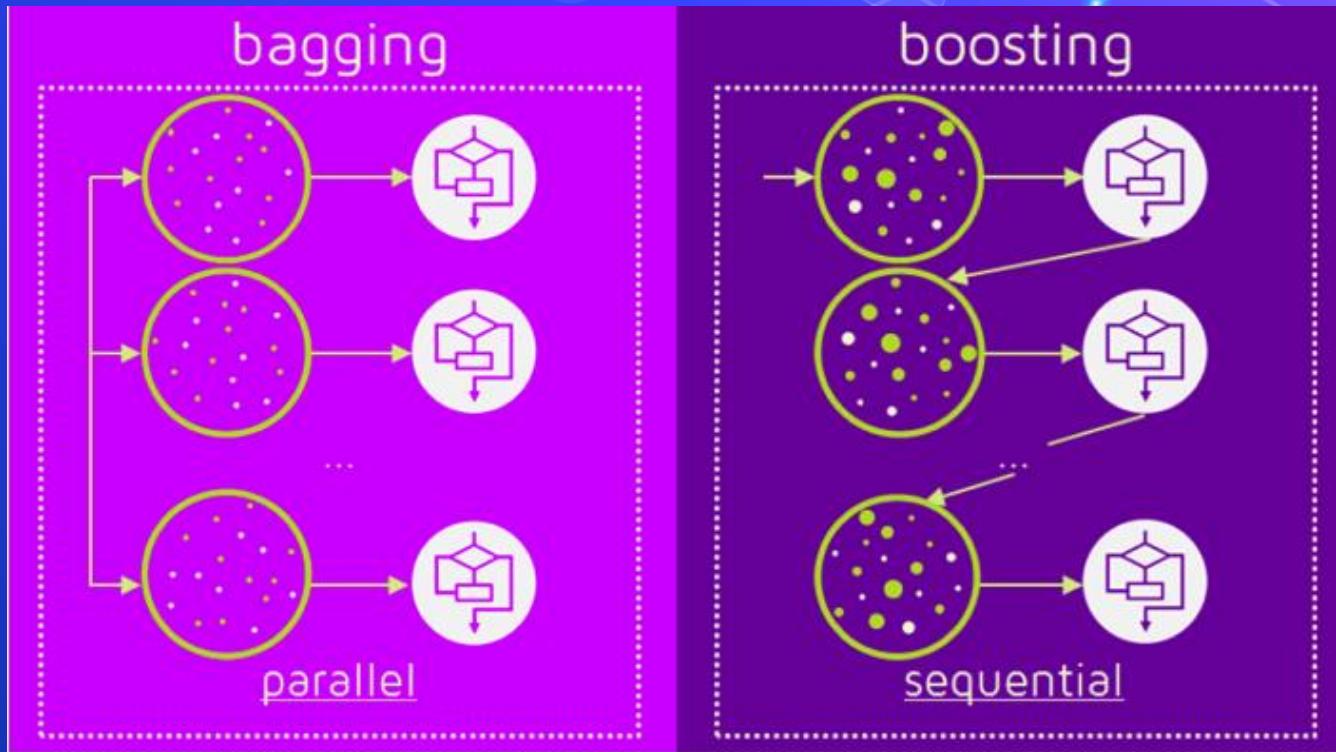
- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

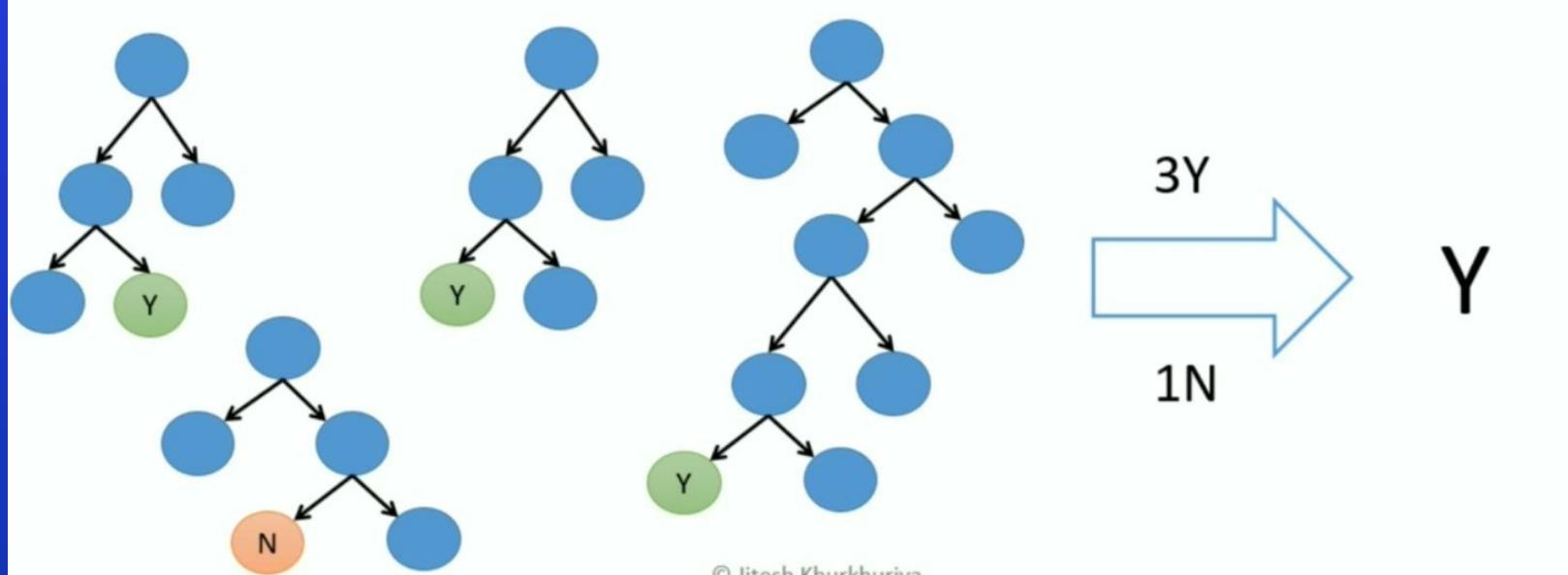
- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

Reinforcement Learning

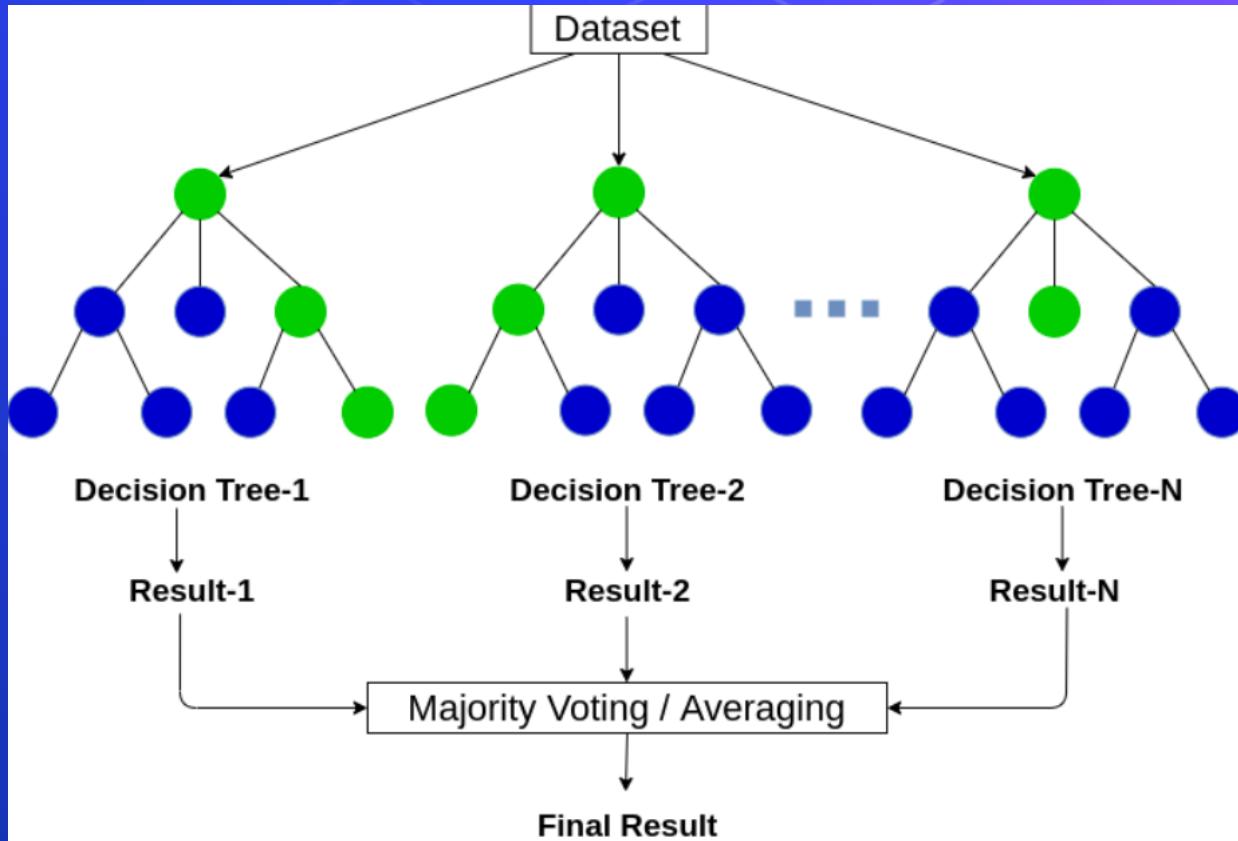
What is Bagging & Boosting



What is Bagging & Boosting



Random Forests

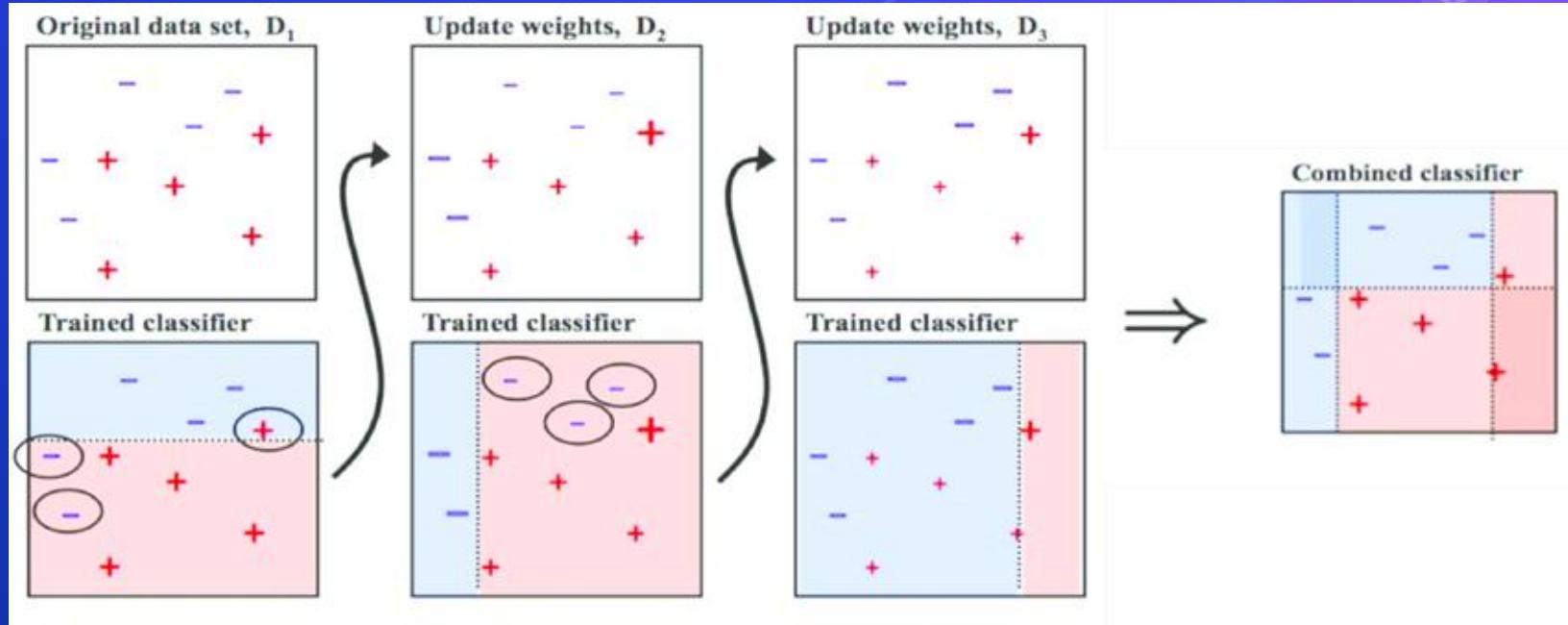


Random Forests

```
1 from sklearn.ensemble import RandomForestClassifier  
2 from sklearn.metrics import classification_report  
3 from sklearn.metrics import accuracy_score  
4 from sklearn.metrics import confusion_matrix  
5  
6  
7 model = RandomForestClassifier()  
8 model.fit(x_train, y_train)  
9 y_pred = model.predict(x_test)  
10  
11 print(confusion_matrix(y_test, y_pred))  
12 print(accuracy_score(y_test, y_pred))  
13 print(classification_report(y_test, y_pred))
```



What is Bagging & Boosting



XGBoost

```
1 import xgboost as xgb
2 from sklearn.metrics import classification_report
3 from sklearn.metrics import accuracy_score
4 from sklearn.metrics import confusion_matrix
5
6
7 model = xgb.XGBClassifier()
8 model.fit(x_train, y_train)
9 y_pred = model.predict(x_test)
10
11 print(confusion_matrix(y_test, y_pred))
12 print(accuracy_score(y_test, y_pred))
13 print(classification_report(y_test, y_pred))
```



Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - **Evaluating Model Performance**

Unsupervised Learning

- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

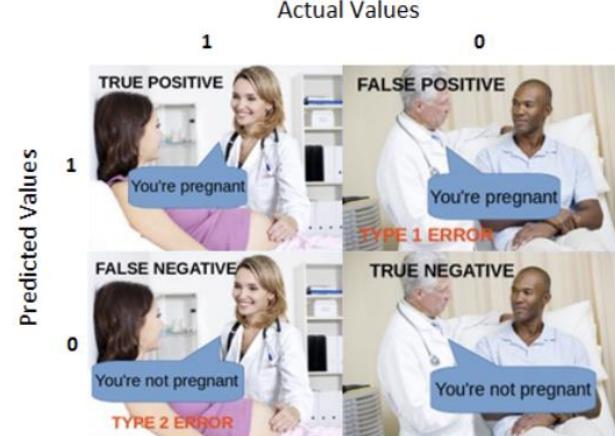
Reinforcement Learning

Evaluating Model Performance (Confusion Matrix)

		Actual Value	
		Positive	Negative
Predicted Value	Positive	TP (True Positive)	FP (False Positive)
	Negative	FN (False Negative)	TN (True Negative)

- True Positive (TP) : Observation is positive, and is predicted to be positive.
- False Negative (FN) : Observation is positive, but is predicted negative.
- True Negative (TN) : Observation is negative, and is predicted to be negative.
- False Positive (FP) : Observation is negative, but is predicted positive.

Evaluating Model Performance (Confusion Matrix)



True positives (TP): actuals are positives and are predicted as positives.

You predicted that a woman is pregnant and she actually is.

False positives (FP) - Type 1 Error: actuals are negatives and are predicted as positives.

You predicted that a man is pregnant but he actually is not.

False negatives (FN) - Type 2 Error: actuals are positives and are predicted as negatives.

You predicted that a woman is not pregnant but she actually is.

True negatives (TN): actuals are negatives and are predicted as positives.

You predicted that a man is not pregnant and he actually is not.

Evaluating Model Performance (Confusion Matrix)

```
1 from sklearn.metrics import confusion_matrix
2
3
4 confusion_matrix(y_pred, y_true)
5
6 """
7 array([[43,  2],
8        [11, 87]], dtype=int64)
9 """
```



Evaluating Model Performance (Accuracy)

Accuracy

- It's a measure of how good the model is.
- It's the ratio of the true predicted values over all the values.

Accuracy:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

Recall:

$$Recall = \frac{TP}{TP + FN}$$

Precision:

$$Precision = \frac{TP}{TP + FP}$$

F₁ score:

$$F_1 = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$

Evaluating Model Performance (Accuracy)

```
1 from sklearn.metrics import accuracy_score  
2  
3  
4 accuracy_score(y_pred, y_true)  
5  
6 """  
7 0.9090909090909091  
8 """
```



Evaluating Model Performance (Recall or Sensitivity)

- TP will be that the COVID 19 residents diagnosed with COVID-19.
- TN will be that healthy residents are with good health.
- FP will be that those actually healthy residents are predicted as COVID 19 residents.
- FN will be that those actual COVID 19 residents are predicted as the healthy residents.

which scenario do you think will have the highest cost?

Imagine that if we predict COVID-19 residents as healthy patients and they do not need to quarantine, there would be a massive number of COVID-19 infections. The cost of false negatives is much higher than the cost of false positives.

Case 1

COVID 19 = 1
Healthy = 0

Cost of FN > Cost of FP

Actual

Healthy predicted as sick

Sick predicted as healthy

Predict	COVID 19 (1)	Diagnosed COVID 19 (1)	Diagnosed Healthy (0)
Covid 19 (1)	TP	FP	
Healthy (0)	FN	TN	

Evaluating Model Performance (Recall or Sensitivity)

Recall (Sensitivity)

- It's a ratio of True Positives to all the positives in your data.
- Low recall: the more False Negatives the model predicts, the lower the recall.

$$\text{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

Accuracy:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

Recall:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Precision:

$$\text{Precision} = \frac{TP}{TP + FP}$$

F₁ score:

$$F_1 = \frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$$

Evaluating Model Performance (Recall or Sensitivity)

```
1 from sklearn.metrics import recall_score  
2  
3  
4 recall_score(y_pred, y_true)  
5  
6 """  
7 0.9090909090909091  
8 """
```



Evaluating Model Performance (Precision)

- TP will be that spam emails are placed in the spam folder.
- TN will be that important emails are received.
- FP will be that important emails are placed in the spam folder.
- FN will be that spam emails are received.

which scenario do you think will have the highest cost?

Well, since missing important emails will clearly be more of a problem than receiving spam, we can say that in this case, FP will have a higher cost than FN.

Case 2

Spam = 1
Not Spam = 0

Cost of FP > Cost of FN

Actual

Not spam predicted as spam

Predict

	Spam (1)	Not Spam (0)
Spam (1)	TP	FP
Not Spam (0)	FN	TN

Spam predicted as not spam

Evaluating Model Performance (Precision)

Precision

- It's a ratio of True Positives to all the positives predicted by the model.
- Low precision: the more False positives the model predicts, the lower the precision.

$$\text{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

Accuracy:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

Recall:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Precision:

$$\text{Precision} = \frac{TP}{TP + FP}$$

F₁ score:

$$F_1 = \frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$$

Evaluating Model Performance (Precision)

```
1 from sklearn.metrics import precision_score  
2  
3  
4 precision_score(y_pred, y_true)  
5  
6 """  
7 0.9090909090909091  
8 """
```



Evaluating Model Performance

Case 1



COVID 19/ Healthy

Case 2



Spam/Not Spam

Cost of **FN** > Cost of **FP**

Cost of **FP** > Cost of **FN**

Recall

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Precision

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

Evaluating Model Performance (F-1 Score)

F-1 Score

- F-Score called the Harmonic mean of precision and recall.
- F-Score provides a single score that balances both the concerns of precision and recall in one number.
- A good F1 score means that you have low false positives and low false negatives.
- An F1 score is considered perfect when it's 1, while the model is a total failure when it's 0.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Accuracy:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

Recall:

$$Recall = \frac{TP}{TP + FN}$$

Precision:

$$Precision = \frac{TP}{TP + FP}$$

F₁ score:

$$F_1 = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$

Evaluating Model Performance (F-1 Score)

```
1 from sklearn.metrics import f1_score  
2  
3  
4 f1_score(y_pred, y_true)  
5  
6 """  
7 0.9090909090909091  
8 """
```



Evaluating Model Performance (Classification Report)

```
1 from sklearn.metrics import classification_report  
2  
3  
4 classification_report(y_true, y_pred)  
5  
6 """  
7         precision    recall    f1-score    support  
8     class 0       0.50      1.00      0.67      1  
9     class 1       0.00      0.00      0.00      1  
10    class 2       1.00      0.67      0.80      3  
11 """
```

Evaluating Model Performance (F-beta Score)

F-beta Score

- Generalization of F-Score where beta is a parameter to weight the result of the precision or recall.
- Beta = 0.5 to focus more on precision, It's called F-0.5 Score.
- Beta = 2 to focus more on recall, It's called F-2 Score.
- Beta value should be used depending on your domain knowledge business case.

$$F_{\beta} = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}.$$



Evaluating Model Performance (F-beta Score)

```
1 from sklearn.metrics import fbeta_score
2
3
4 fbeta_score(y_true, y_pred, beta=0.5)
5
6 """
7 0.9090909090909091
8 """
```



Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

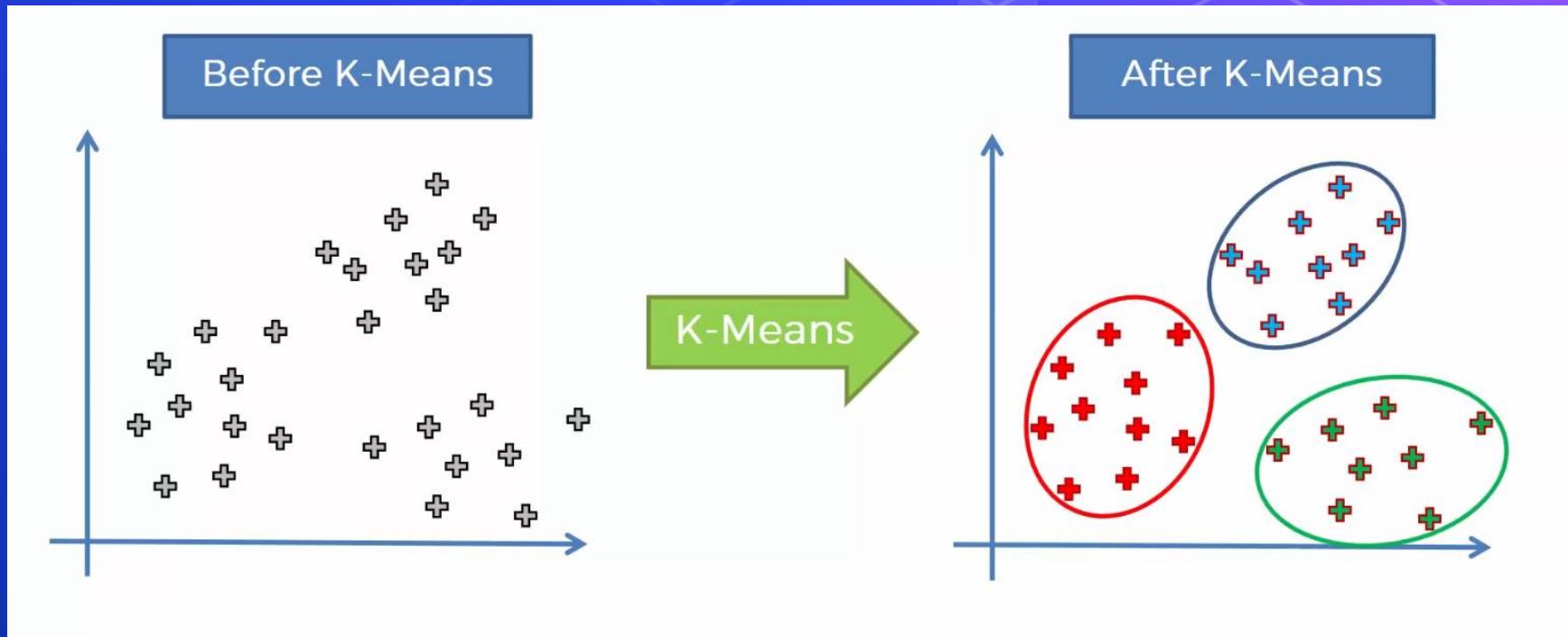
- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

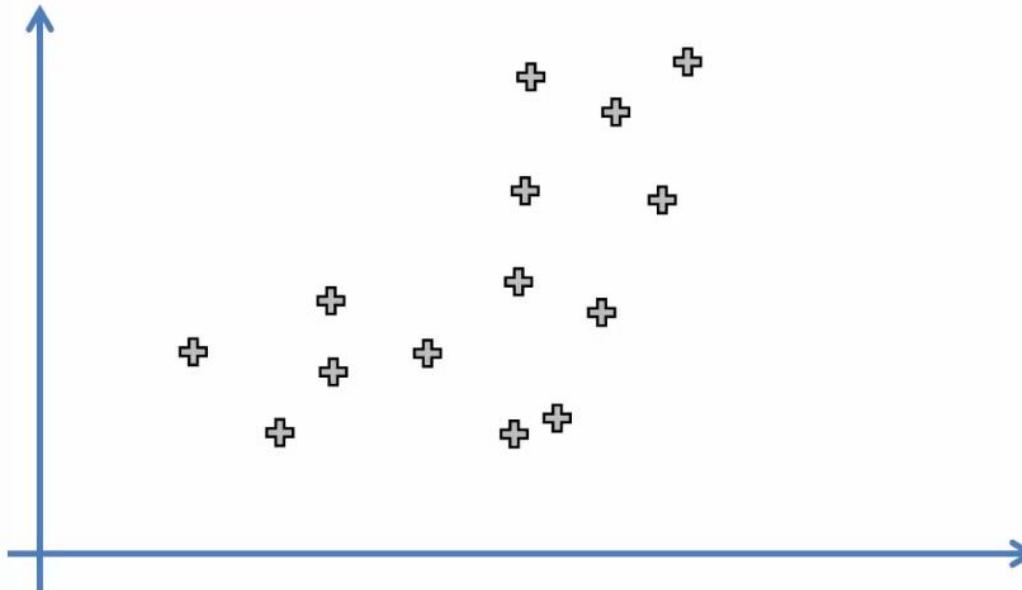
Reinforcement Learning

KMeans



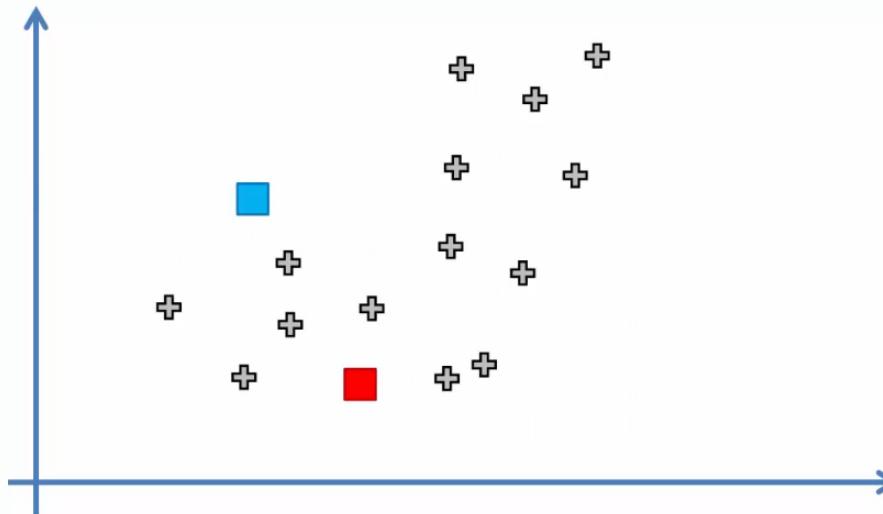
KMeans

STEP 1: Choose the number K of clusters: K = 2



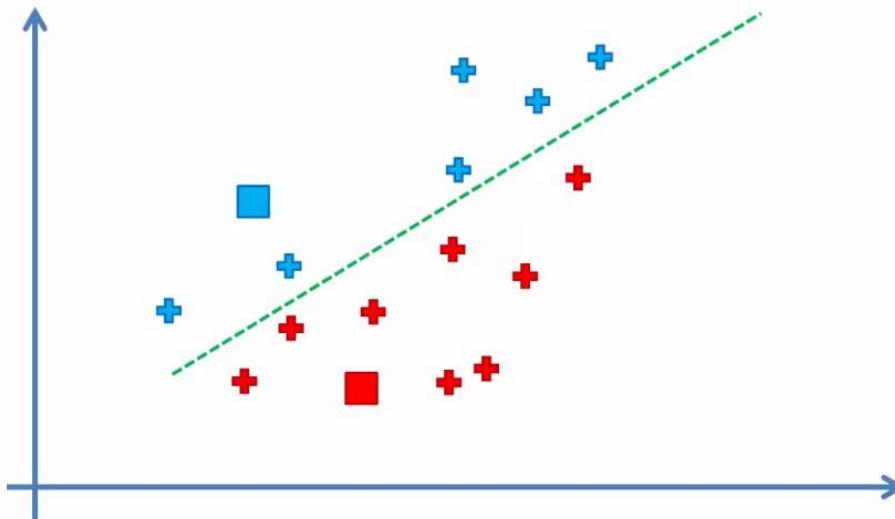
KMeans

STEP 2: Select at random K points, the centroids (not necessarily from your dataset)



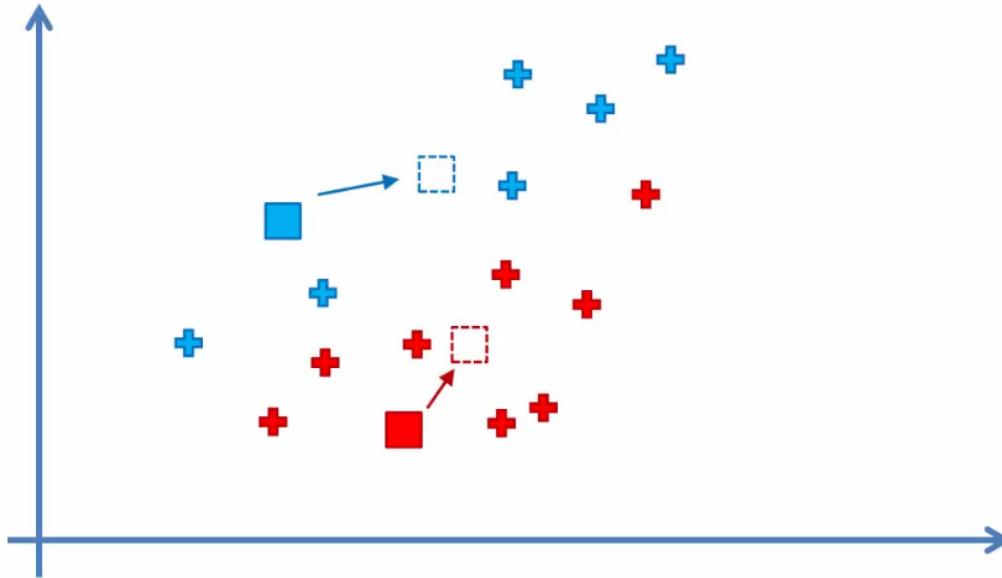
KMeans

STEP 3: Assign each data point to the closest centroid → That forms K clusters



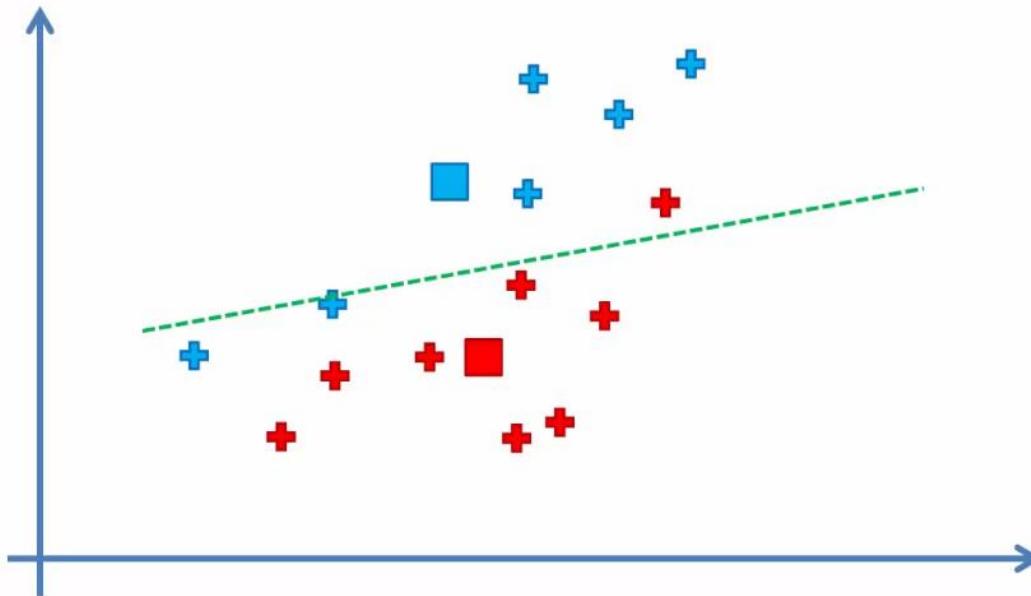
KMeans

STEP 4: Compute and place the new centroid of each cluster



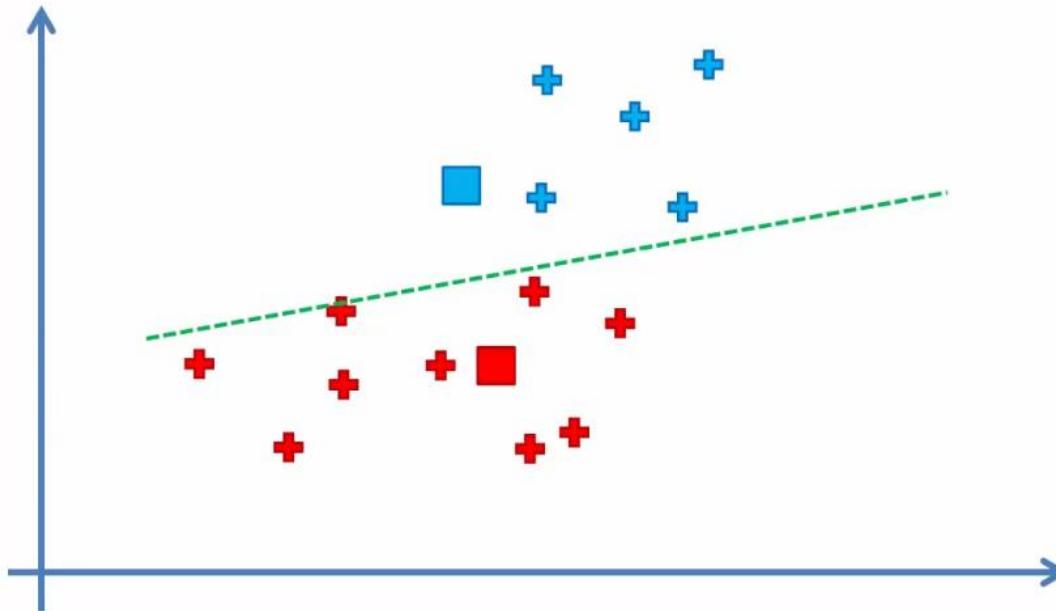
KMeans

STEP 5: Reassign each data point to the new closest centroid.
If any reassignment took place, go to STEP 4, otherwise go to FIN.



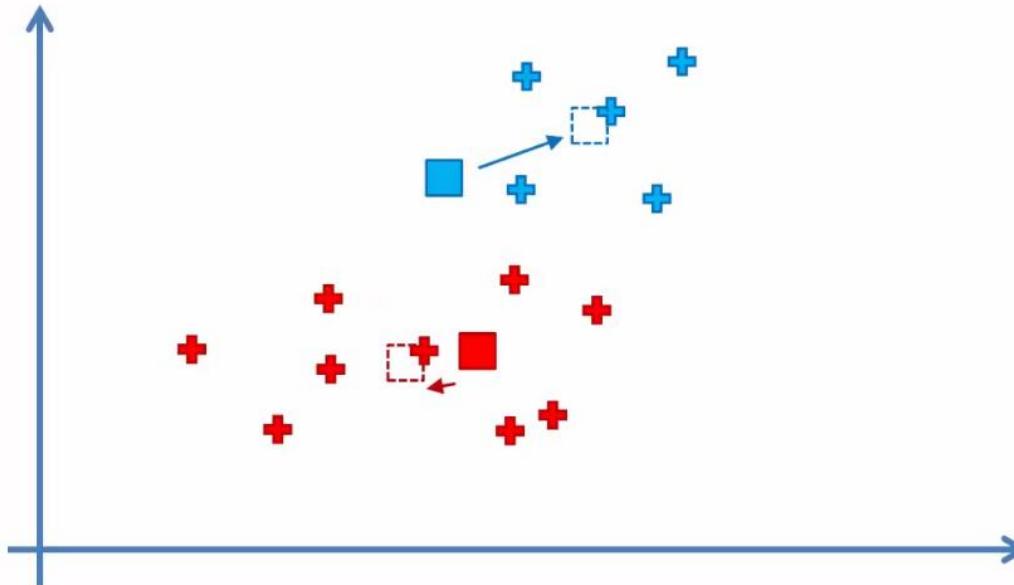
KMeans

STEP 5: Reassign each data point to the new closest centroid.
If any reassignment took place, go to STEP 4, otherwise go to FIN.



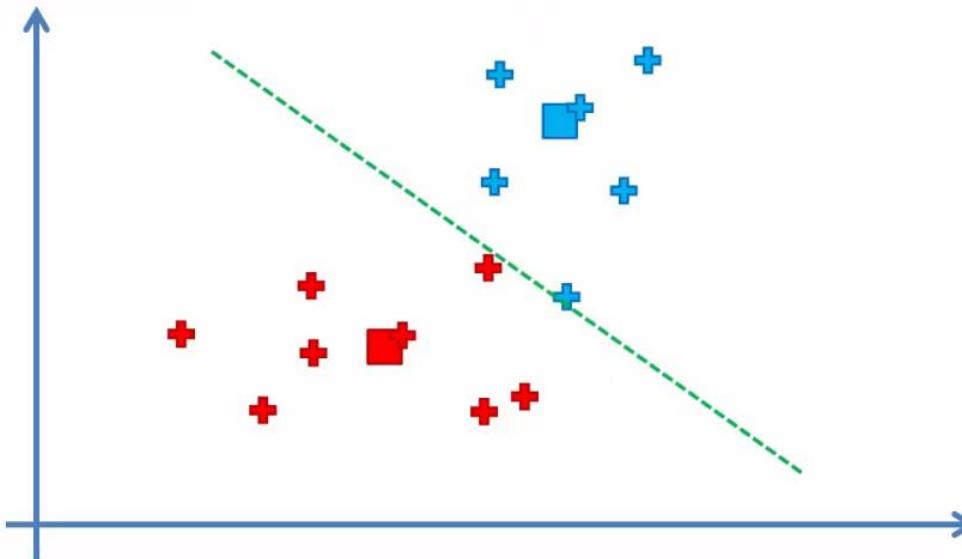
KMeans

STEP 4: Compute and place the new centroid of each cluster



KMeans

STEP 5: Reassign each data point to the new closest centroid.
If any reassignment took place, go to STEP 4, otherwise go to FIN.



KMeans

```
1 from sklearn.cluster import KMeans  
2  
3 kmeans = KMeans(n_clusters=3)  
4 kmeans.fit(X)  
5 kmeans.predict(X)
```

Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

▪ Clustering

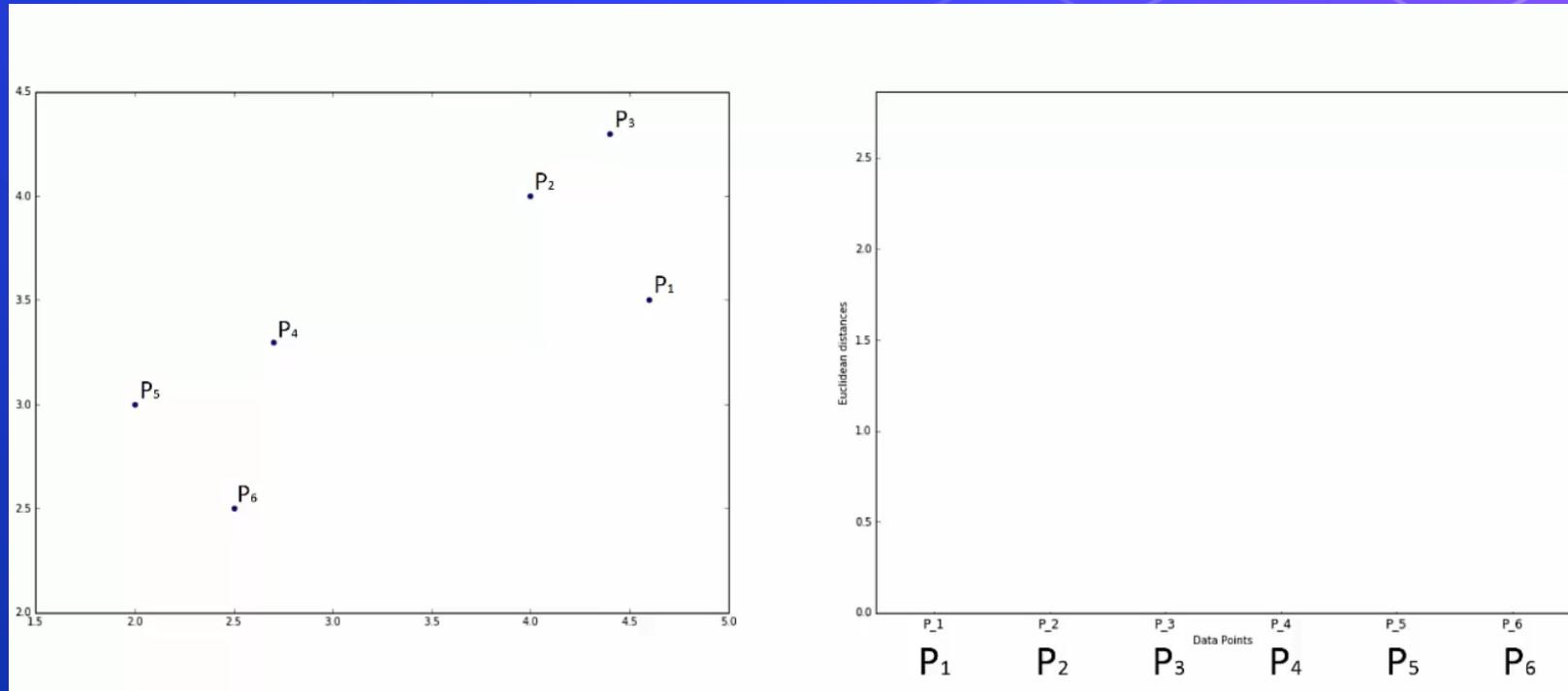
- KMeans
- Hierarchical Clustering
- Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

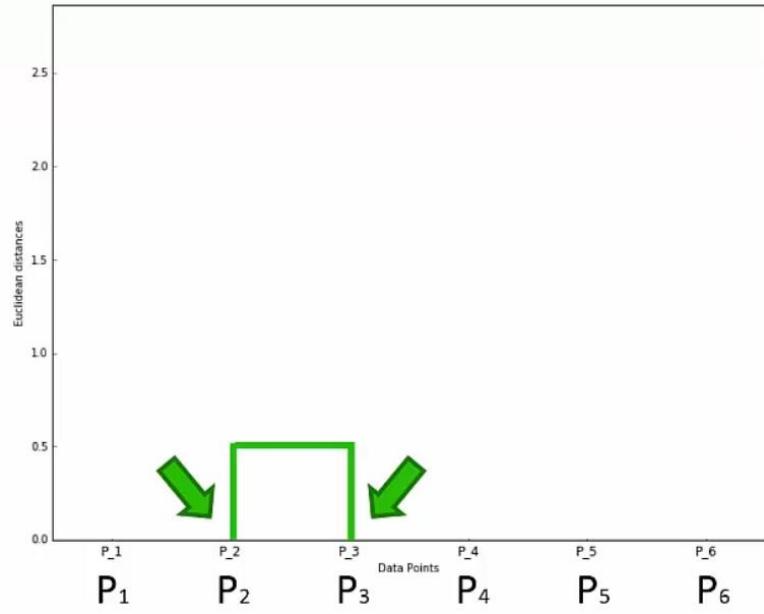
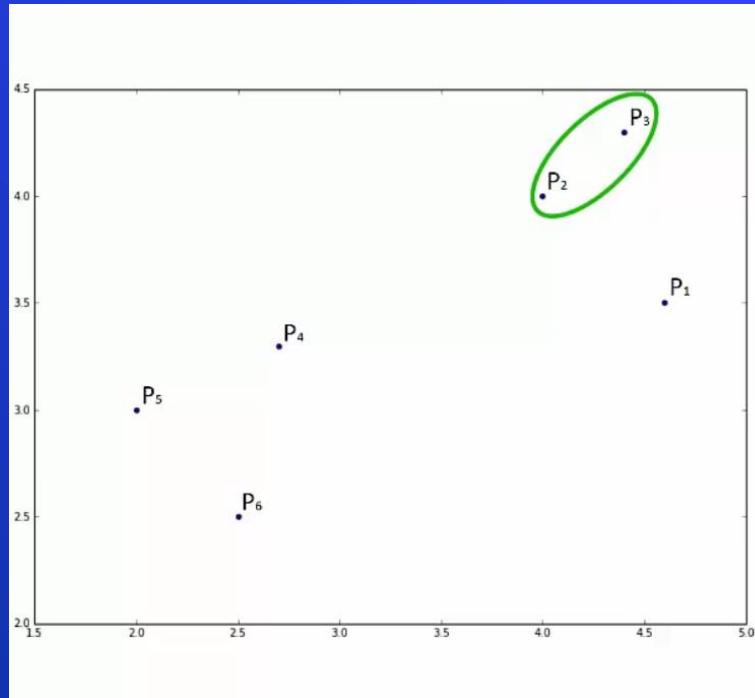
- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

Reinforcement Learning

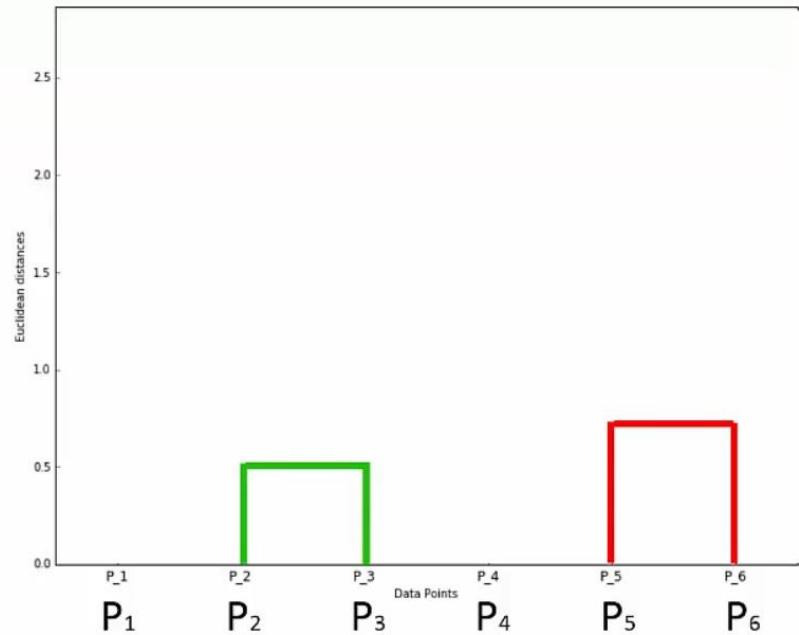
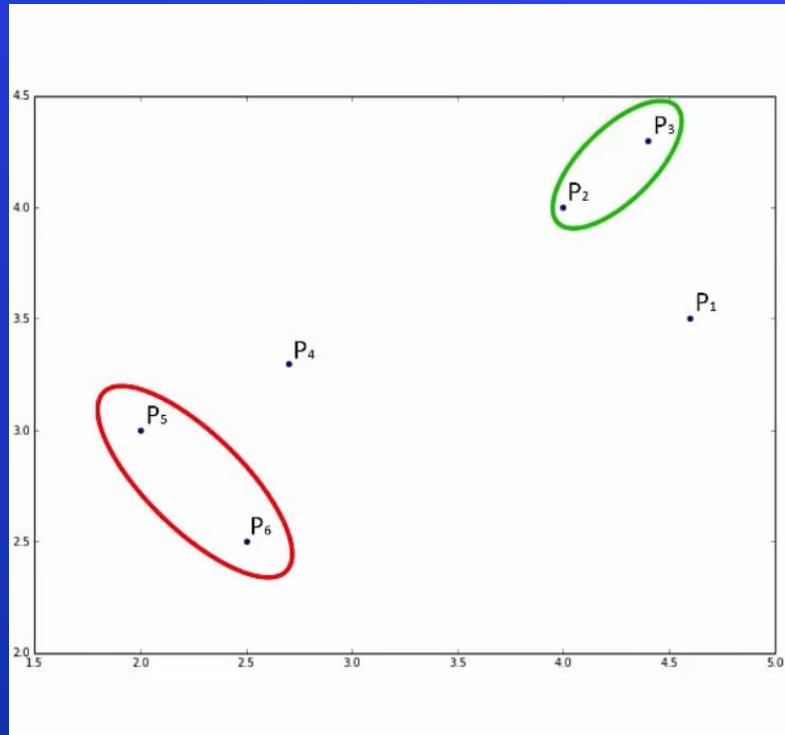
Hierarchical Clustering with dendrogram



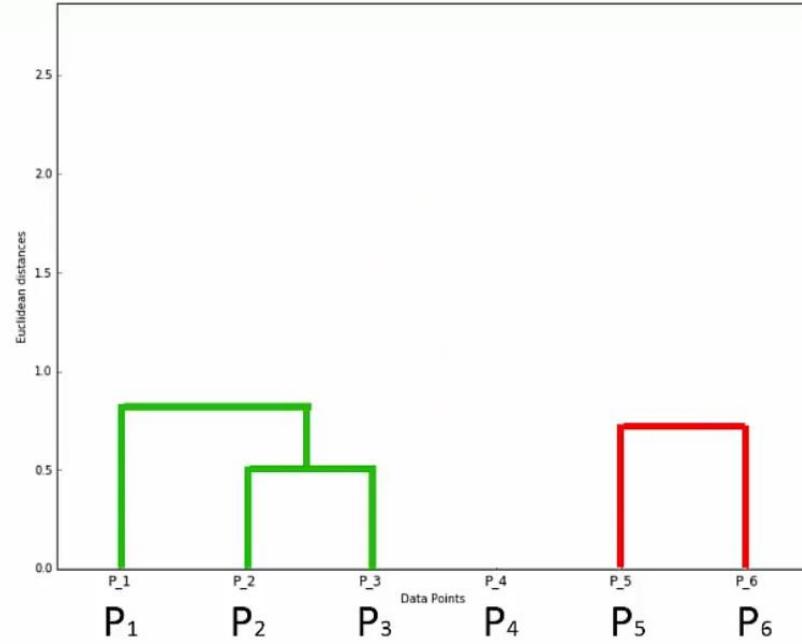
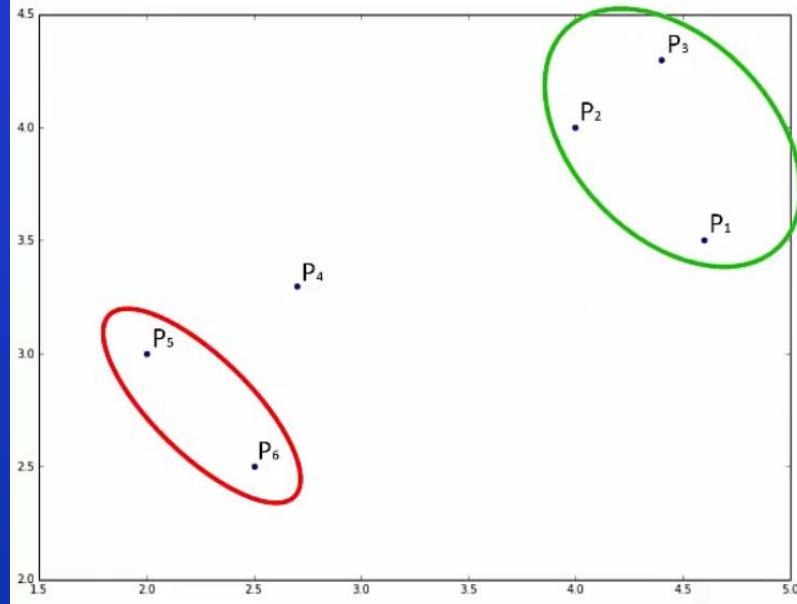
Hierarchical Clustering with dendrogram



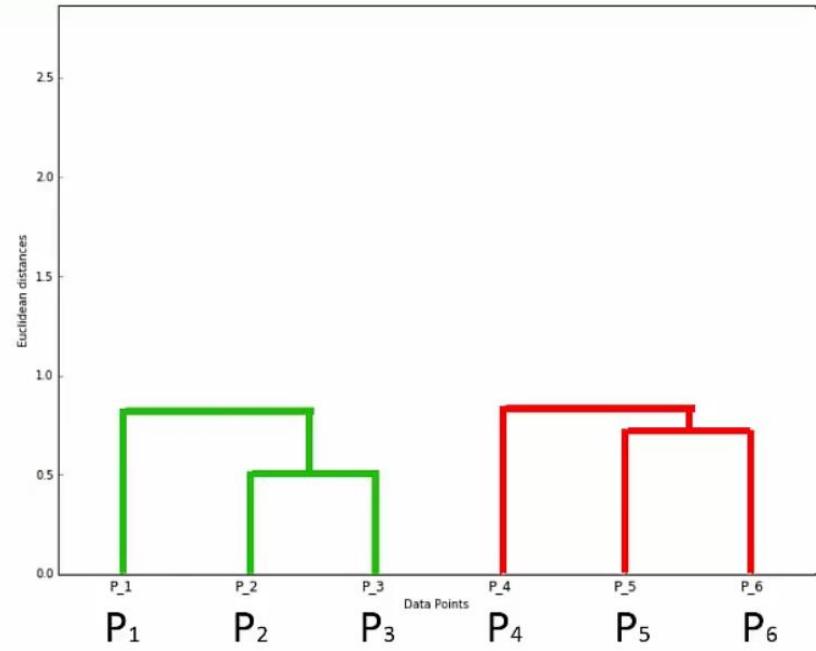
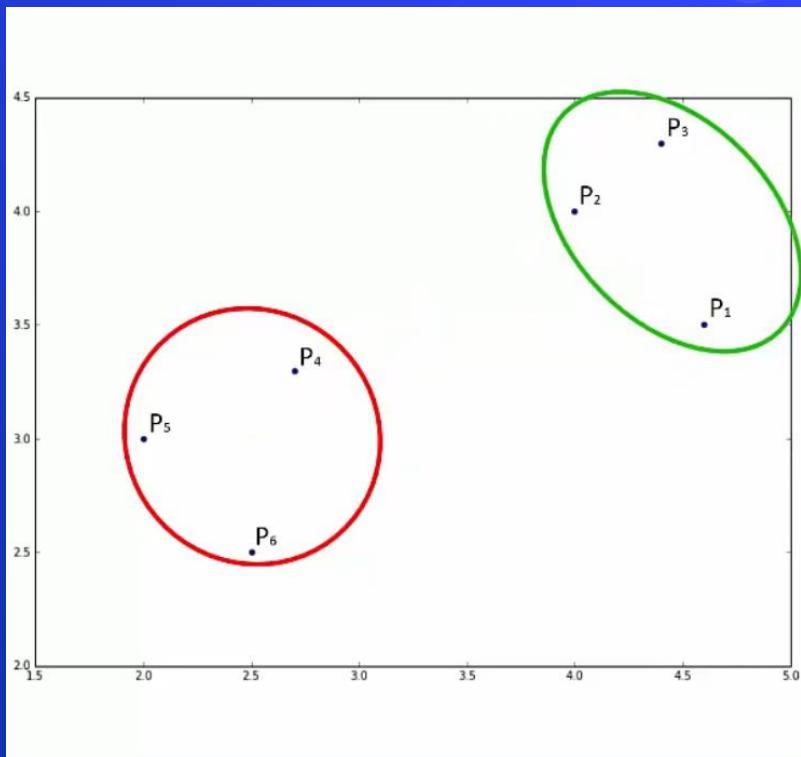
Hierarchical Clustering with dendrogram



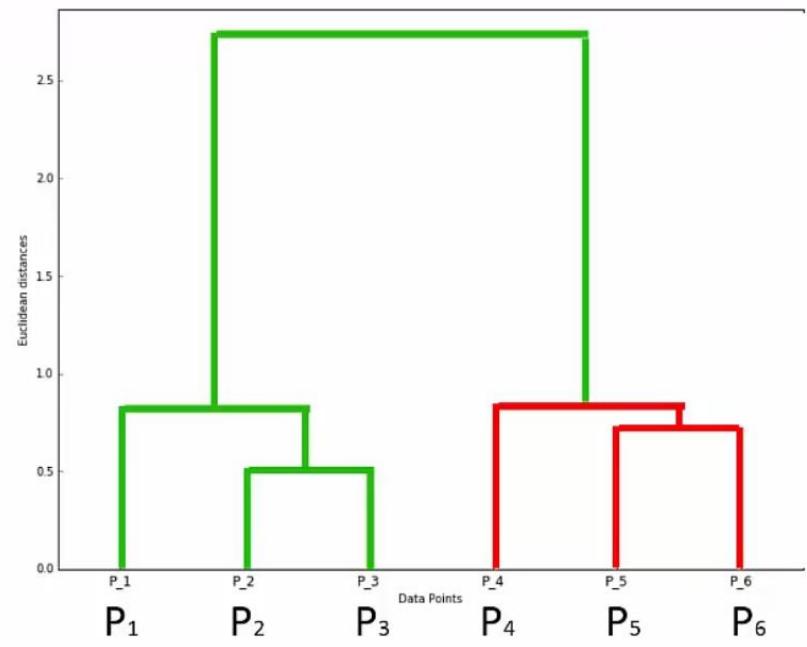
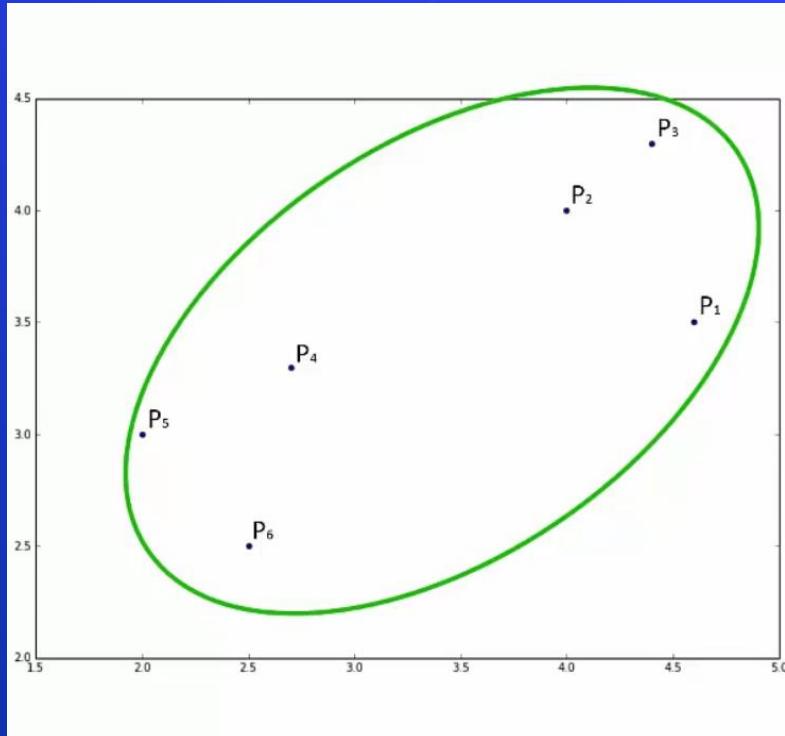
Hierarchical Clustering with dendrogram



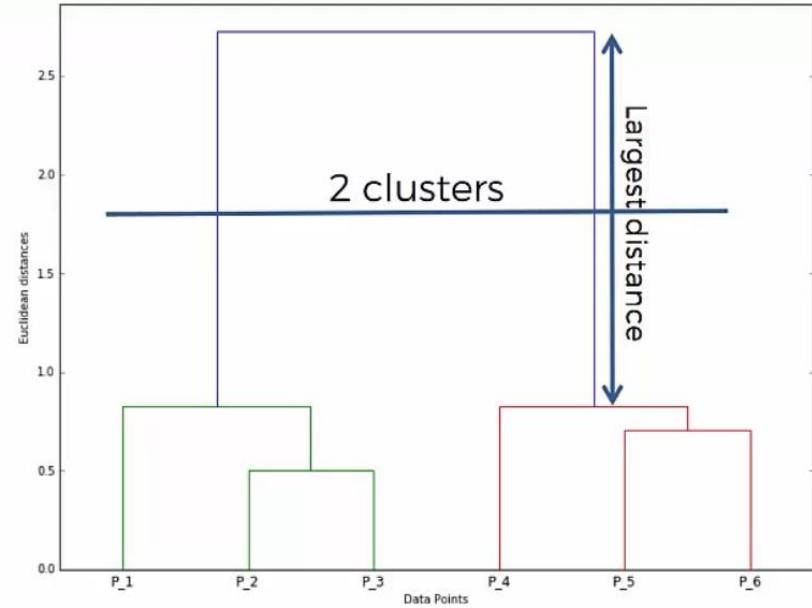
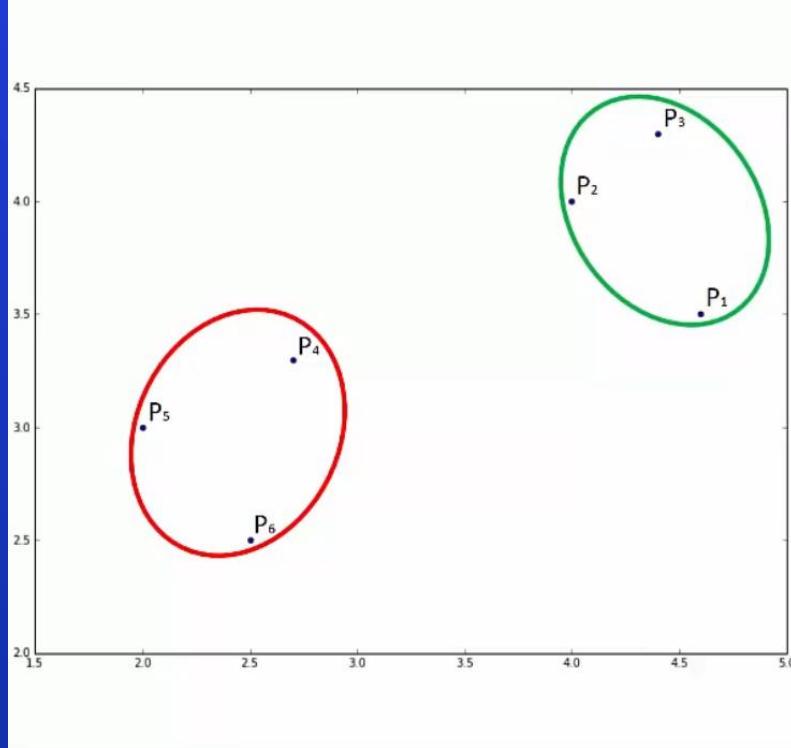
Hierarchical Clustering with dendrogram



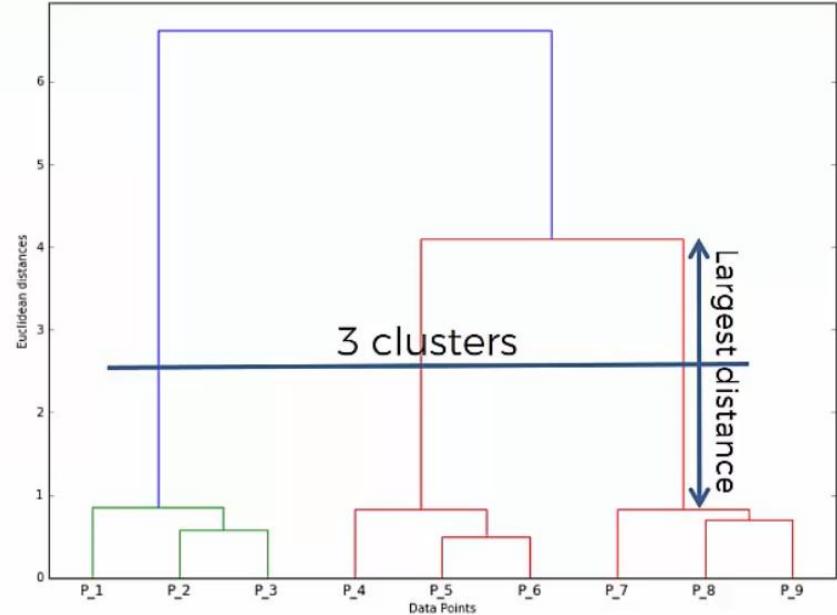
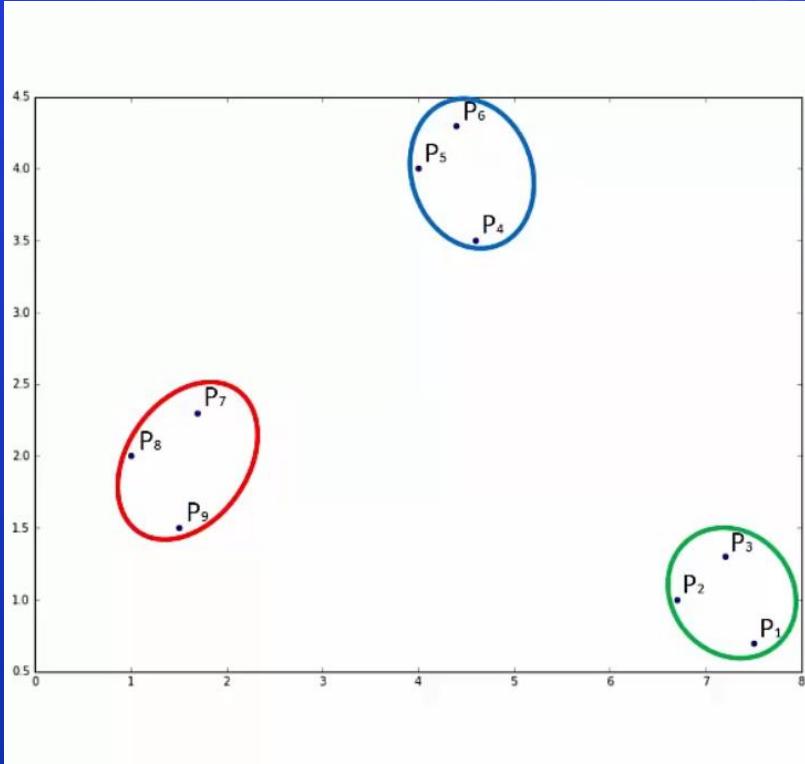
Hierarchical Clustering with dendrogram



Hierarchical Clustering with dendrogram



Hierarchical Clustering with dendrogram



Hierarchical Clustering with dendrogram

```
1 import scipy.cluster.hierarchy as sch
2 from sklearn.cluster import AgglomerativeClustering
3
4 # visualize dendrogram
5 dendrogram = sch.dendrogram(sch.linkage(x, method='ward'))
6
7 # train model
8 model = AgglomerativeClustering(n_clusters=3)
9 y_labels = model.fit_predict(x)
```

Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

▪ Clustering

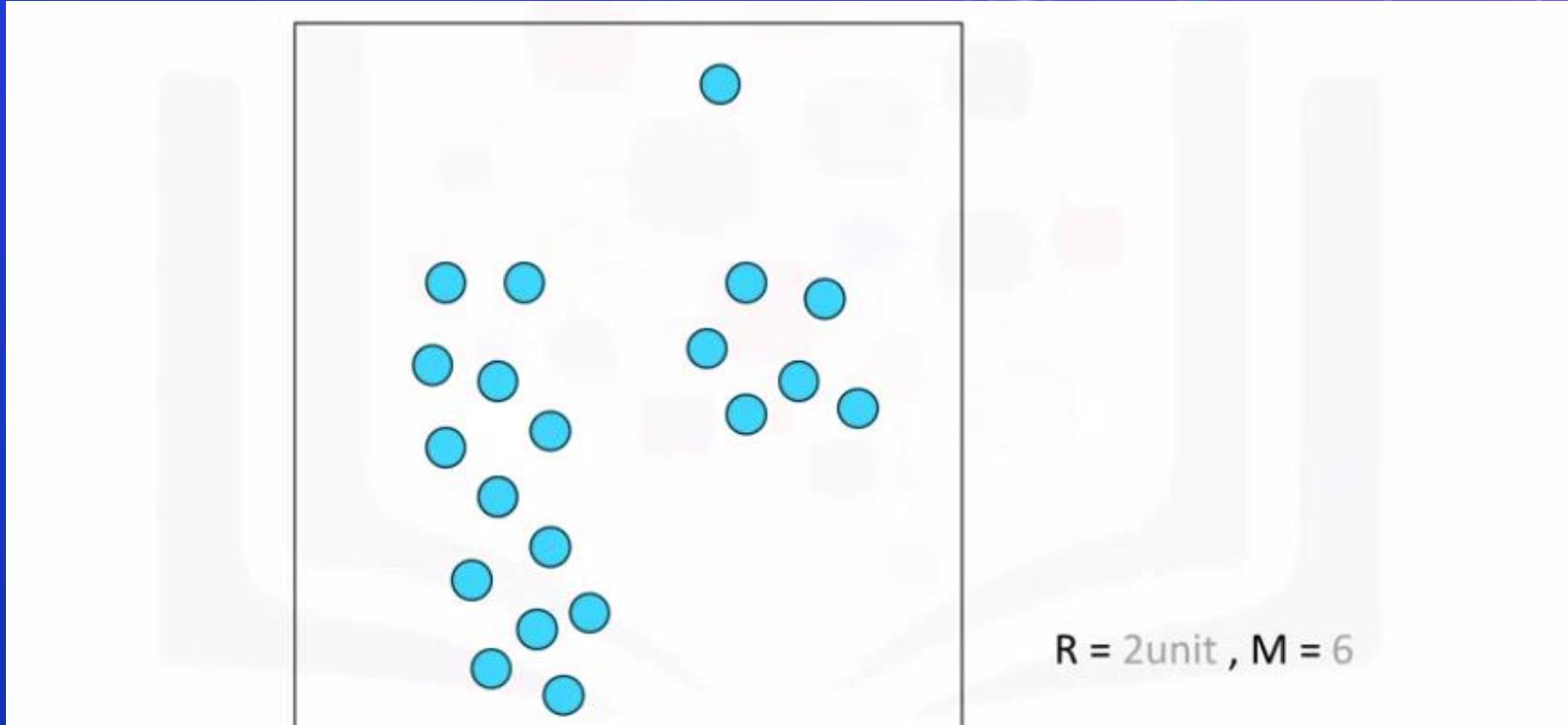
- KMeans
 - Hierarchical Clustering
 - **Density Based Clustering - DBSCAN**
- Association rule mining
 - Apriori
 - Dimension Reduction
 - PCA

Model Selection & Evaluation

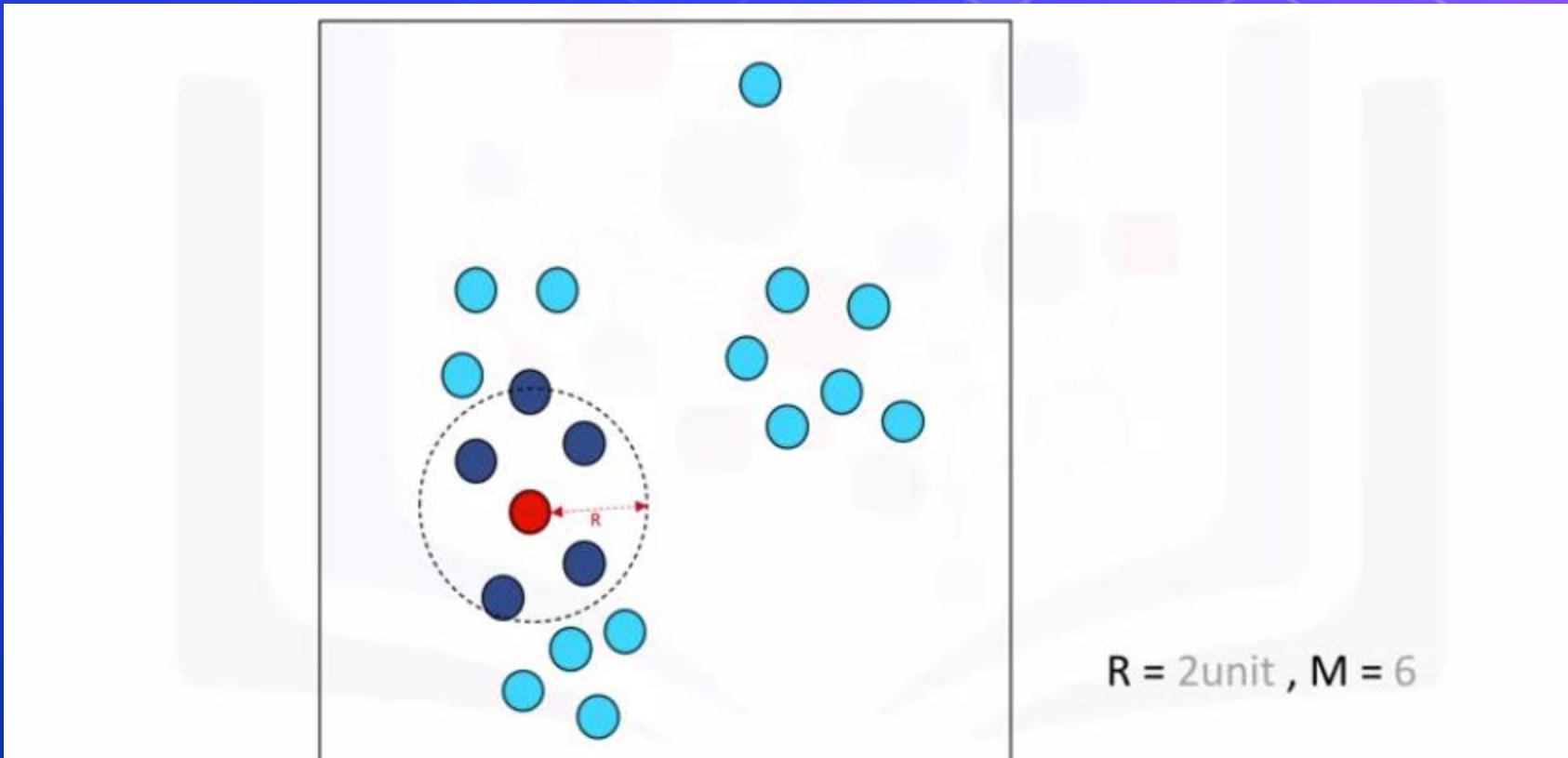
- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

Reinforcement Learning

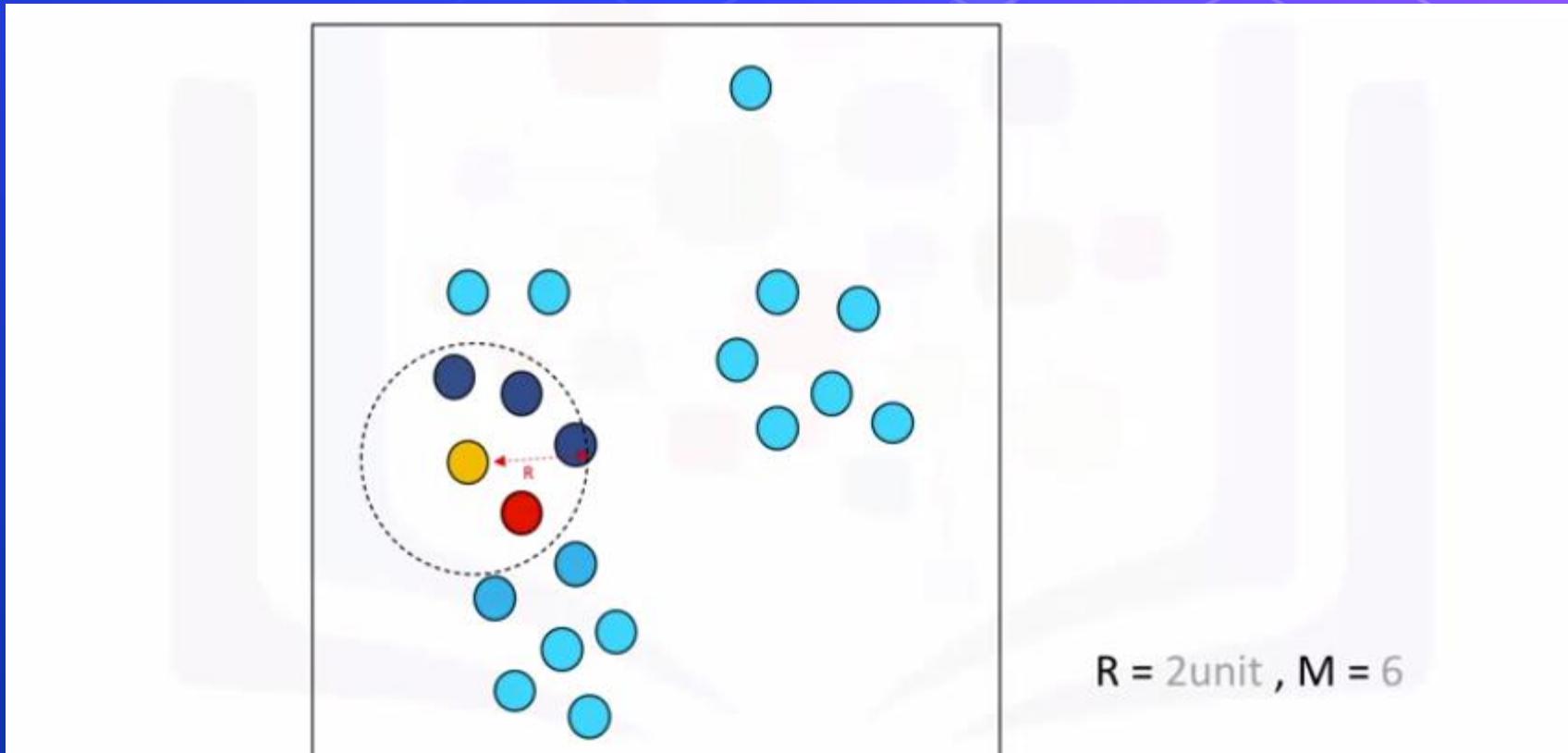
Density Based Clustering - DBSCAN



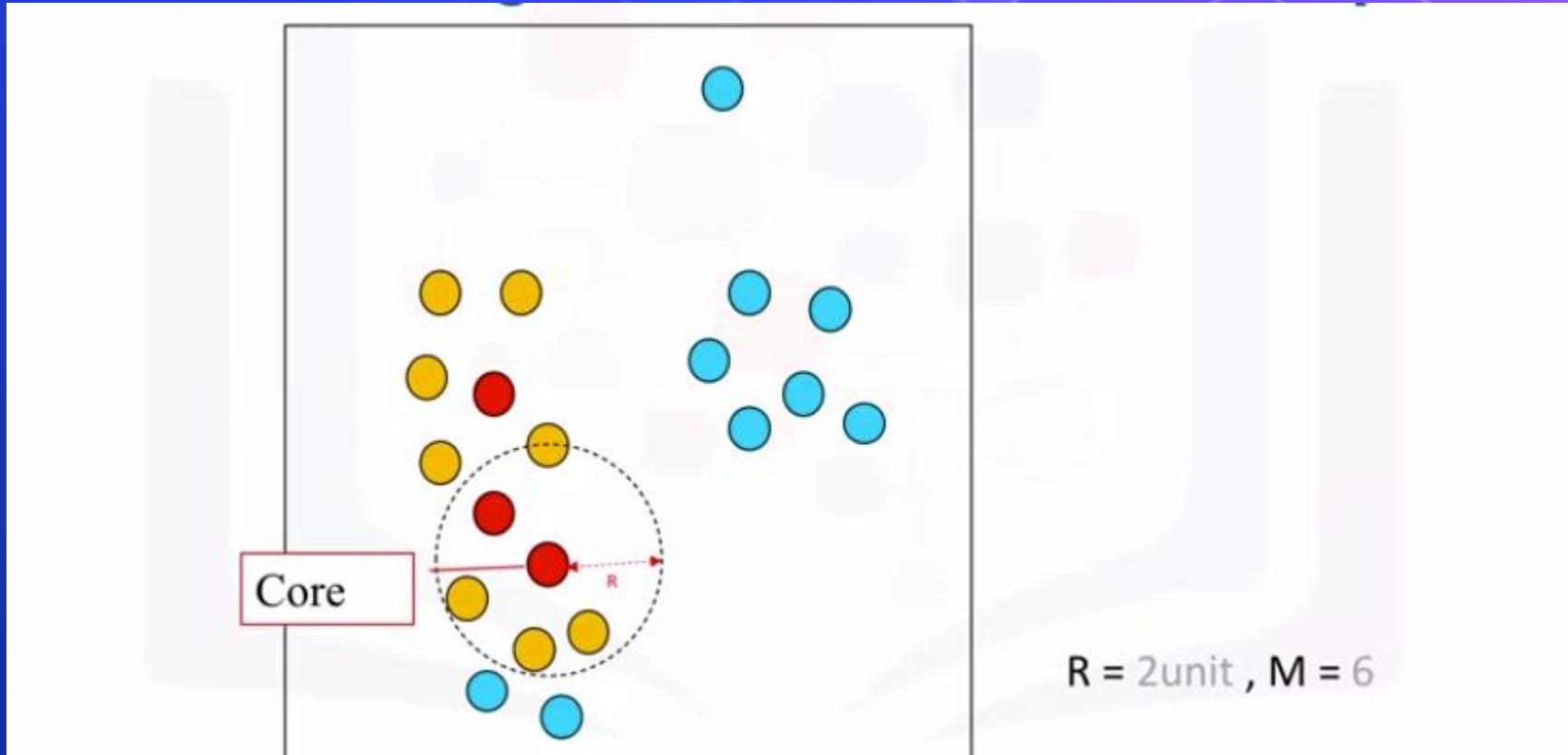
Density Based Clustering - DBSCAN (Core point)



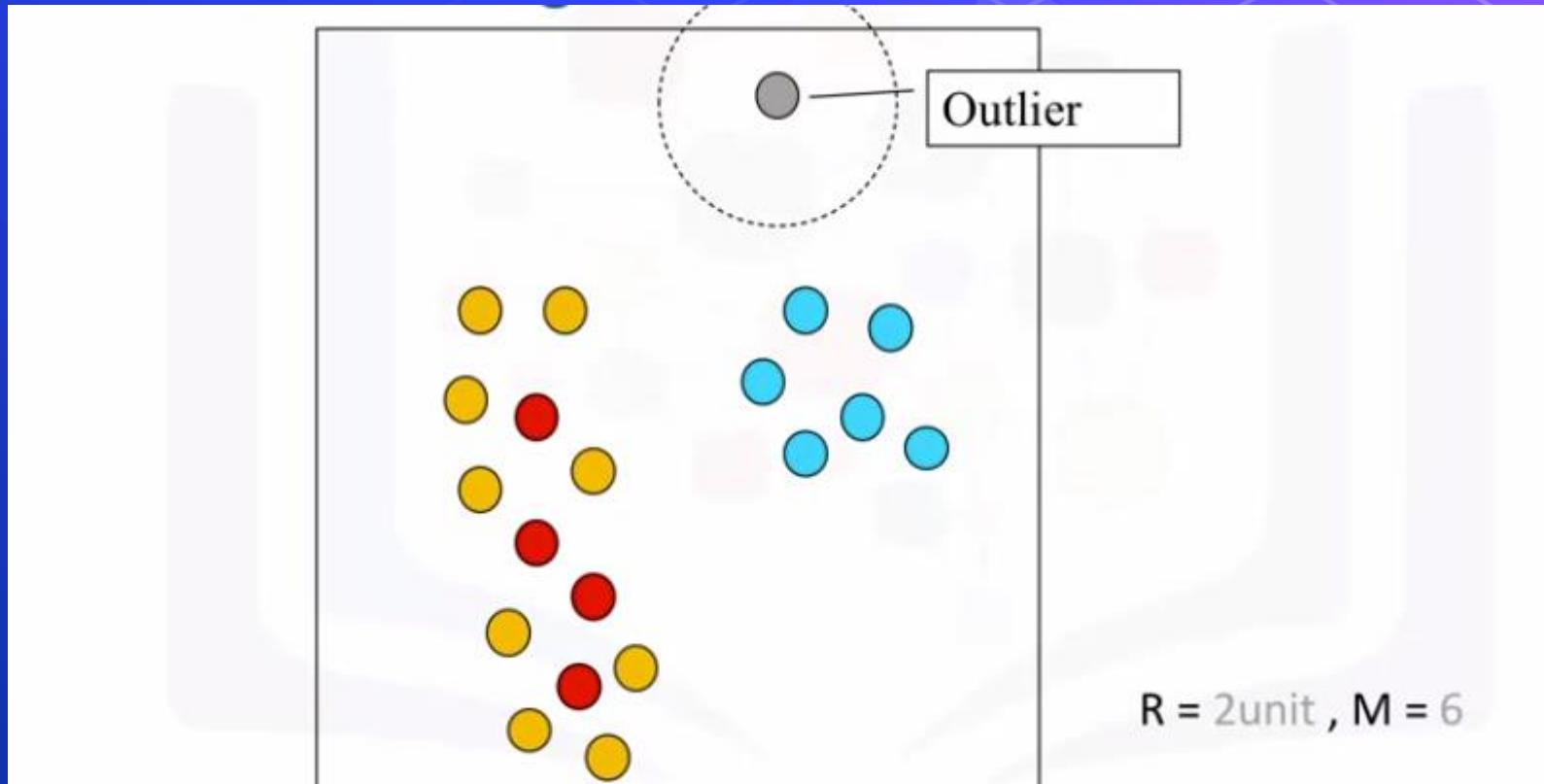
Density Based Clustering - DBSCAN (Border point)



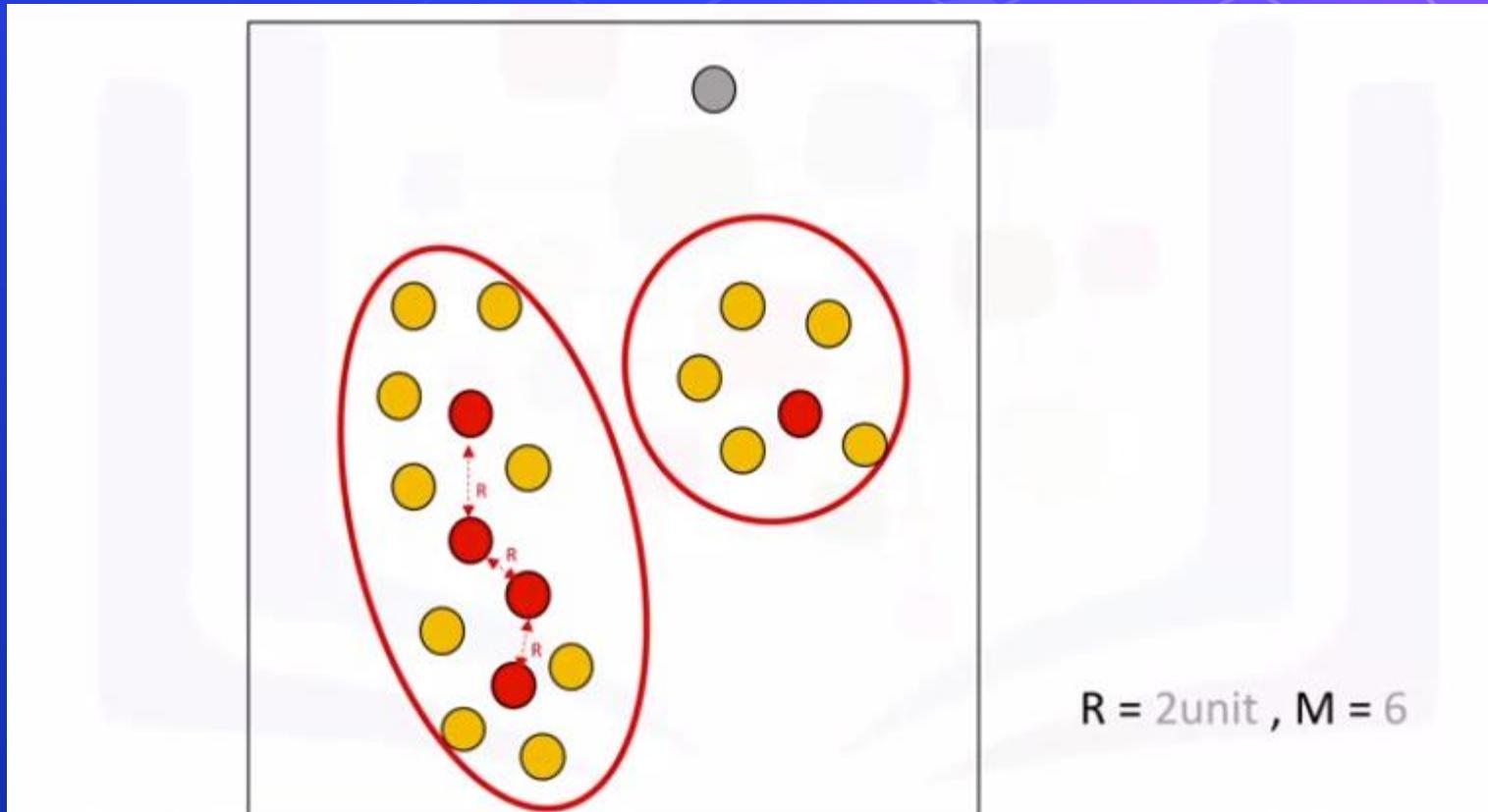
Density Based Clustering - DBSCAN



Density Based Clustering - DBSCAN



Density Based Clustering - DBSCAN



Density Based Clustering - DBSCAN

```
1 from sklearn.cluster import DBSCAN  
2  
3 model=DBSCAN(eps=0.3, min_samples=10)  
4 y_labels = model.fit_predict(x)
```

Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

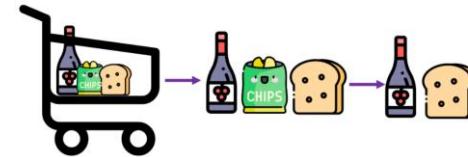
- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

Reinforcement Learning

Apriori



Transaction at a Local Market

T1	A	B	C
T2	A	C	D
T3	B	C	D
T4	A	D	E
T5	B	C	E

Association Rules Exercise

- Here are a dozen sales transactions.
- The objective is to use this transaction data to find affinities between products, that is, which products sell together often.
- The support level will be set at 33 percent; the confidence level will be set at 50 percent.

Apriori

Transactions List

1	Milk	Egg	Bread	Butter
2	Milk	Butter	Egg	Ketchup
3	Bread	Butter	Ketchup	
4	Milk	Bread	Butter	
5	Bread	Butter	Cookies	
6	Milk	Bread	Butter	Cookies
7	Milk	Cookies		
8	Milk	Bread	Butter	
9	Bread	Butter	Egg	Cookies
10	Milk	Butter	Bread	
11	Milk	Bread	Butter	
12	Milk	Bread	Cookies	Ketchup

1-item Sets	Frequency
Milk	9
Bread	10
Butter	10
Egg	3
Ketchup	3
Cookies	5

Frequent 1-item Sets	Frequency
Milk	9
Bread	10
Butter	10
Cookies	5

Apriori

Transactions List

1	Milk	Egg	Bread	Butter
2	Milk	Butter	Egg	Ketchup
3	Bread	Butter	Ketchup	
4	Milk	Bread	Butter	
5	Bread	Butter	Cookies	
6	Milk	Bread	Butter	Cookies
7	Milk	Cookies		
8	Milk	Bread	Butter	
9	Bread	Butter	Egg	Cookies
10	Milk	Butter	Bread	
11	Milk	Bread	Butter	
12	Milk	Bread	Cookies	Ketchup

2-item Sets	Frequency
Milk, Bread	7
Milk, Butter	7
Milk, Cookies	3
Bread, Butter	9
Butter, Cookies	3
Bread, Cookies	4

Frequent 2-item Sets	Frequency
Milk, Bread	7
Milk, Butter	7
Bread, Butter	9
Bread, Cookies	4

Apriori

Transactions List

1	Milk	Egg	Bread	Butter
2	Milk	Butter	Egg	Ketchup
3	Bread	Butter	Ketchup	
4	Milk	Bread	Butter	
5	Bread	Butter	Cookies	
6	Milk	Bread	Butter	Cookies
7	Milk	Cookies		
8	Milk	Bread	Butter	
9	Bread	Butter	Egg	Cookies
10	Milk	Butter	Bread	
11	Milk	Bread	Butter	
12	Milk	Bread	Cookies	Ketchup

Milk, Bread, Butter, Cookies

3-item Sets	Frequency
Milk, Bread, Butter	6
Milk, Bread, Cookies	1
Bread, Butter, Cookies	3
Milk, Butter, Cookies	2

Frequent 3-item Sets	Frequency
Milk, Bread, Butter	6

Association Rule Mining - Subset Creation

- Frequent 3-Item Set = $I \Rightarrow \{\text{Milk, Bread, Butter}\}$
- Non-Empty subset are
 - $\{\{\text{Milk}\}, \{\text{Bread}\}, \{\text{Butter}\}, \{\text{Milk, Bread}\}, \{\text{Milk, Butter}\}, \{\text{Bread, Butter}\}\}$
- How to form Association Rule...?
 - For every non-empty subset S of I , the association rule is,
 - $S \rightarrow (I-S)$
 - If $\text{support}(I) / \text{support}(S) \geq \text{min_confidence}$

Association Rule Mining - Subset Creation

- Non-Empty subset are
 - $\{\{\text{Milk}\}, \{\text{Bread}\}, \{\text{Butter}\}, \{\text{Milk, Bread}\}, \{\text{Milk, Butter}\}, \{\text{Bread, Butter}\}\}$
 - Min_Support = 30% and Min_Confidence = 60%
- Rule 1: $\{\text{Milk}\} \rightarrow \{\text{Bread, Butter}\}$ {S=50%, C=66.67%}
 - Support = $6/12 = 50\%$
 - Confidence = $\text{Support}(\text{Milk, Bread, Butter})/\text{Support}(\text{Milk}) = \frac{6/12}{9/12} = 6/9 = 66.67\% > 60\%$
 - Valid
- Rule 2: $\{\text{Bread}\} \rightarrow \{\text{Milk, Butter}\}$ {S=50%, C=60%}
 - Support = $6/12 = 50\%$
 - Confidence = $\text{Support}(\text{Milk, Bread, Butter})/\text{Support}(\text{Bread}) = 6/10 = 60\% \geq 60\%$
 - Valid

Association Rule Mining - Subset Creation

- Non-Empty subset are
 - $\{\{\text{Milk}\}, \{\text{Bread}\}, \{\text{Butter}\}, \{\text{Milk, Bread}\}, \{\text{Milk, Butter}\}, \{\text{Bread, Butter}\}\}$
 - Min_Support = 30% and Min_Confidence = 60%
- Rule 3: $\{\text{Butter}\} \rightarrow \{\text{Milk, Bread}\}$ {S=50%, C=60%}
 - Support = $6/12 = 50\%$
 - Confidence = $\text{Support}(\text{Milk, Bread, Butter})/\text{Support}(\text{Butter}) = 6/10 = 60\% >= 60$
 - Valid
- Rule 4: $\{\text{Milk, Bread}\} \rightarrow \{\text{Butter}\}$ {S=50%, C=85.7%}
 - Support = $6/12 = 50\%$
 - Confidence = $\text{Support}(\text{Milk, Bread, Butter})/\text{Support}(\text{Milk, Bread}) = 6/7 = 85.7\% > 60\%$
 - Valid

Association Rule Mining - Subset Creation

- Non-Empty subset are
 - $\{\{\text{Milk}\}, \{\text{Bread}\}, \{\text{Butter}\}, \{\text{Milk, Bread}\}, \{\text{Milk, Butter}\}, \{\text{Bread, Butter}\}\}$
 - Min_Support = 30% and Min_Confidence = 60%
- Rule 5: $\{\text{Milk, Butter}\} \rightarrow \{\text{Bread}\}$ {S=50%, C=85.7%}
 - Support = $6/12 = 50\%$
 - Confidence = $\text{Support}(\text{Milk, Bread, Butter})/\text{Support}(\text{Milk, Butter}) = 6/7 = 85.7\% \geq 60\%$
 - Valid
- Rule 6: $\{\text{Bread, Butter}\} \rightarrow \{\text{Milk}\}$ {S=50%, C=66.67%}
 - Support = $6/12 = 50\%$
 - Confidence = $\text{Support}(\text{Milk, Bread, Butter})/\text{Support}(\text{Bread, Butter}) = 6/9 = 66.67\% \geq 60\%$
 - Valid

Apriori

```
1 from apyori import apriori  
2  
3 records = []  
4 for i in range(0, rows):  
5     records.append([str(df.values[i,j]) for j in range(0, columns)])  
6  
7 association_rules = apriori(records)  
8 association_results = list(association_rules)
```

Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

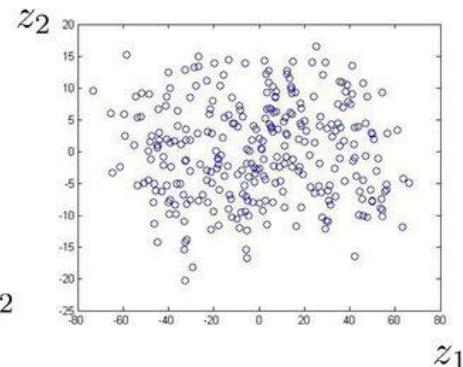
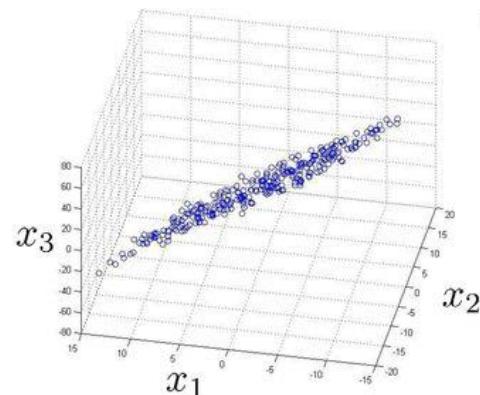
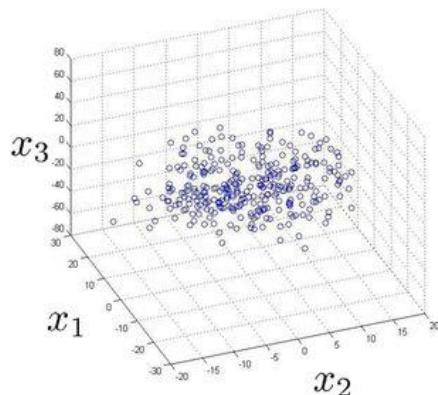
- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

Reinforcement Learning

PCA

Data Compression

Reduce data from 3D to 2D



Andrew Ng

Dimension Reduction with PCA

```
● ● ●  
1 from sklearn.decomposition import PCA  
2  
3 X = # feature vector  
4  
5 pca = PCA(0.9)  
6 X = pca.fit_transform(X)  
7 pca.explained_variance_ratio_
```

PRINCIPAL COMPONENT ANALYSIS

PCA projects the features onto the principal components. The motivation is to reduce the features dimensionality while only losing a small amount of information.

First principal component

Second principal component

https://en.wikipedia.org/wiki/Principal_component_analysis

Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

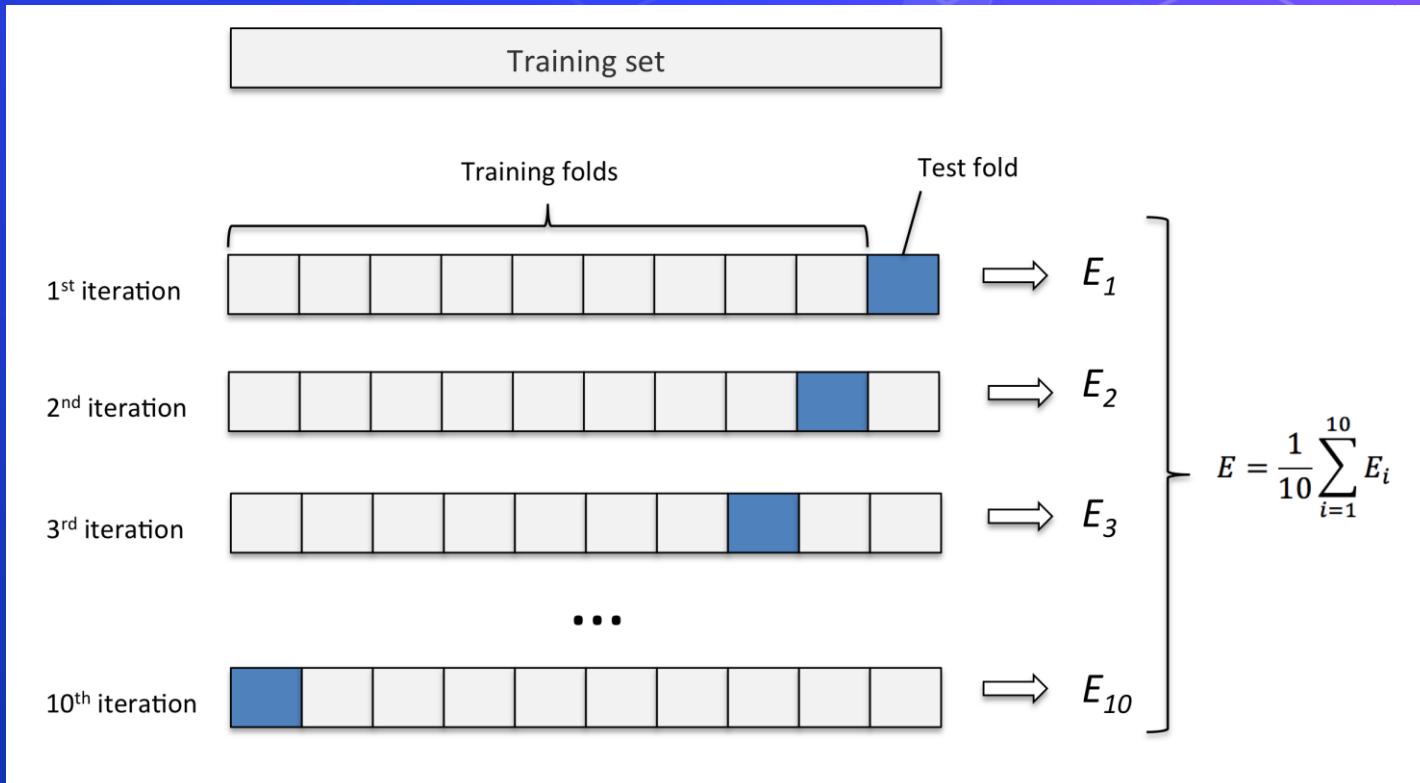
Reinforcement Learning

Cross Validation with K-Fold

Why we use Cross Validation

- Cross Validation uses **Stratified** technique to make sure the distribution of the categorical target is the same in every fold.
- More “efficient” use of data as every observation is used for both training and testing.
- Reduce overfitting and the model achieve the generalization.

Cross Validation with K-Fold



Cross Validation with K-Fold

```
1 from sklearn.svm import SVC
2 from sklearn.model_selection import cross_validate
3
4 svc = SVC()
5 cv_results_svc = cross_validate(svc, X, Y, cv=10, return_train_score=True)
6
7 cv_results_svc['test_score'].mean()
8 """
9 0.8036085674097743
10 """
```

Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

Reinforcement Learning

Grid Search

```
1 from sklearn.model_selection import GridSearchCV
2
3
4 params = [
5     {'C':[1, 10, 100], 'kernel':['linear', 'sigmoid', 'poly']},
6     {'C':[1, 10, 100], 'kernel':['rbf'], 'gamma':[0.5, 0.6, 0.7, 0.1, 0.01, 0.001]}
7 ]
8
9 grid_search = GridSearchCV(estimator=model,
10                         param_grid=params,
11                         scoring='accuracy',
12                         cv=10)
13 grid_search.fit(x_train, y_train)
14
15
16 print(grid_search.best_params_)
17 print(grid_search.best_params_)
```

Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

- Cross Validation
- **Hyperparameter Tuning**
 - Grid Search
 - Randomized Search

Reinforcement Learning

Randomized Search

```
1 from sklearn.model_selection import RandomizedSearchCV
2
3
4 params = [
5     {'C':[1, 10, 100], 'kernel':['linear', 'sigmoid', 'poly']},
6     {'C':[1, 10, 100], 'kernel':['rbf'], 'gamma':[0.5, 0.6, 0.7, 0.1, 0.01, 0.001]}
7     ]
8
9 randomized_search = RandomizedSearchCV(estimator=model,
10                                         param_grid=params,
11                                         scoring='accuracy',
12                                         cv=10)
13 randomized_search.fit(x_train, y_train)
14
15
16 print(randomized_search.best_params_)
17 print(randomized_search.best_params_)
```

Machine Learning

Supervised Learning

- Regression
 - Simple Linear Regression
 - Multiple Linear Regression
 - Polynomial Regression
 - Evaluating Model Performance
- Classification
 - Logistic Regression
 - K-Nearest Neighbors (KNN)
 - Naive Bayes
 - SVM
 - Decision Trees
 - Ensemble Methods
 - What is Bagging & Boosting
 - Random Forests
 - XGBoost
 - Evaluating Model Performance

Unsupervised Learning

- Clustering
 - KMeans
 - Hierarchical Clustering
 - Density Based Clustering - DBSCAN
- Association rule mining
 - Apriori
- Dimension Reduction
 - PCA

Model Selection & Evaluation

- Cross Validation
- Hyperparameter Tuning
 - Grid Search
 - Randomized Search

Reinforcement Learning

Reinforcement Learning

- What is Reinforcement Learning.
- Applications of Reinforcement Learning.
- Why RL is Freaky.
- Markov Decision Process (MDP).
- MDP Examples.
- MDP Solution.
- Q-Learning Algorithm.
- Open AI Gym environment.
- Coding time >_

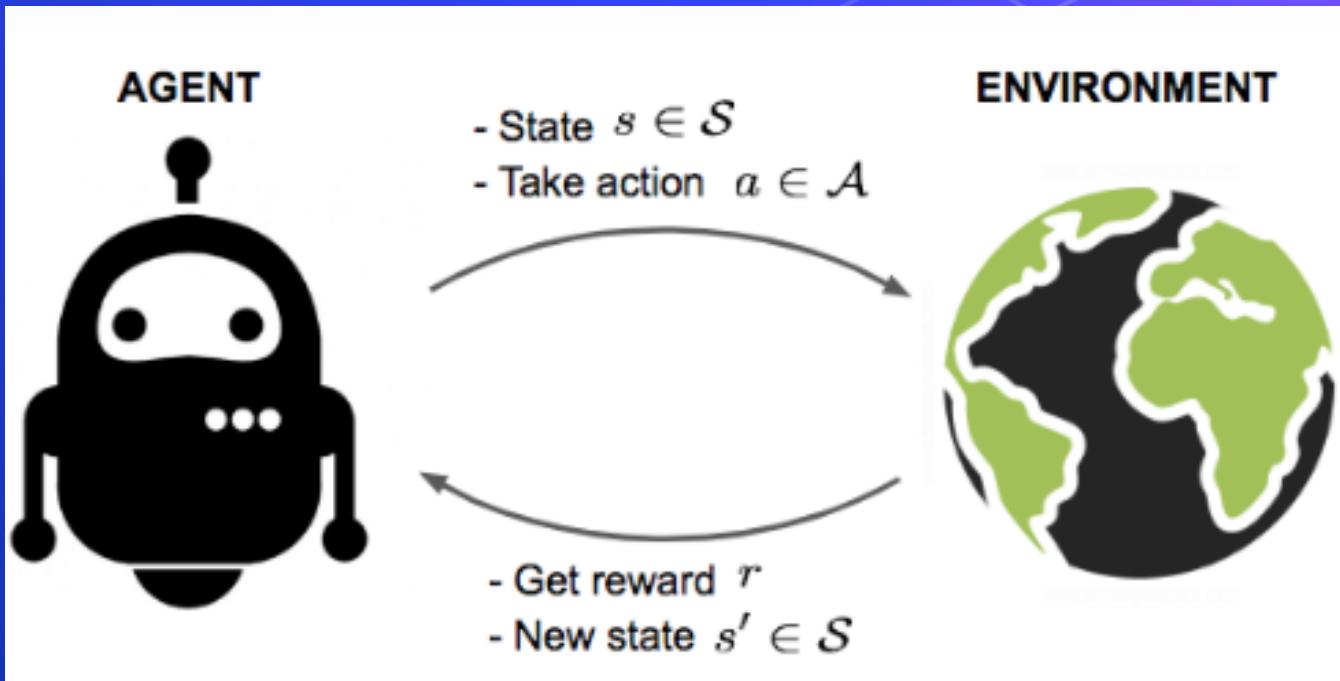


Reinforcement Learning

- What is Reinforcement Learning.
- Applications of Reinforcement Learning.
- Why RL is Freaky.
- Markov Decision Process (MDP).
- MDP Examples.
- MDP Solution.
- Q-Learning Algorithm.
- Open AI Gym environment.
- Coding time >_



What is RL ?



What is RL ?

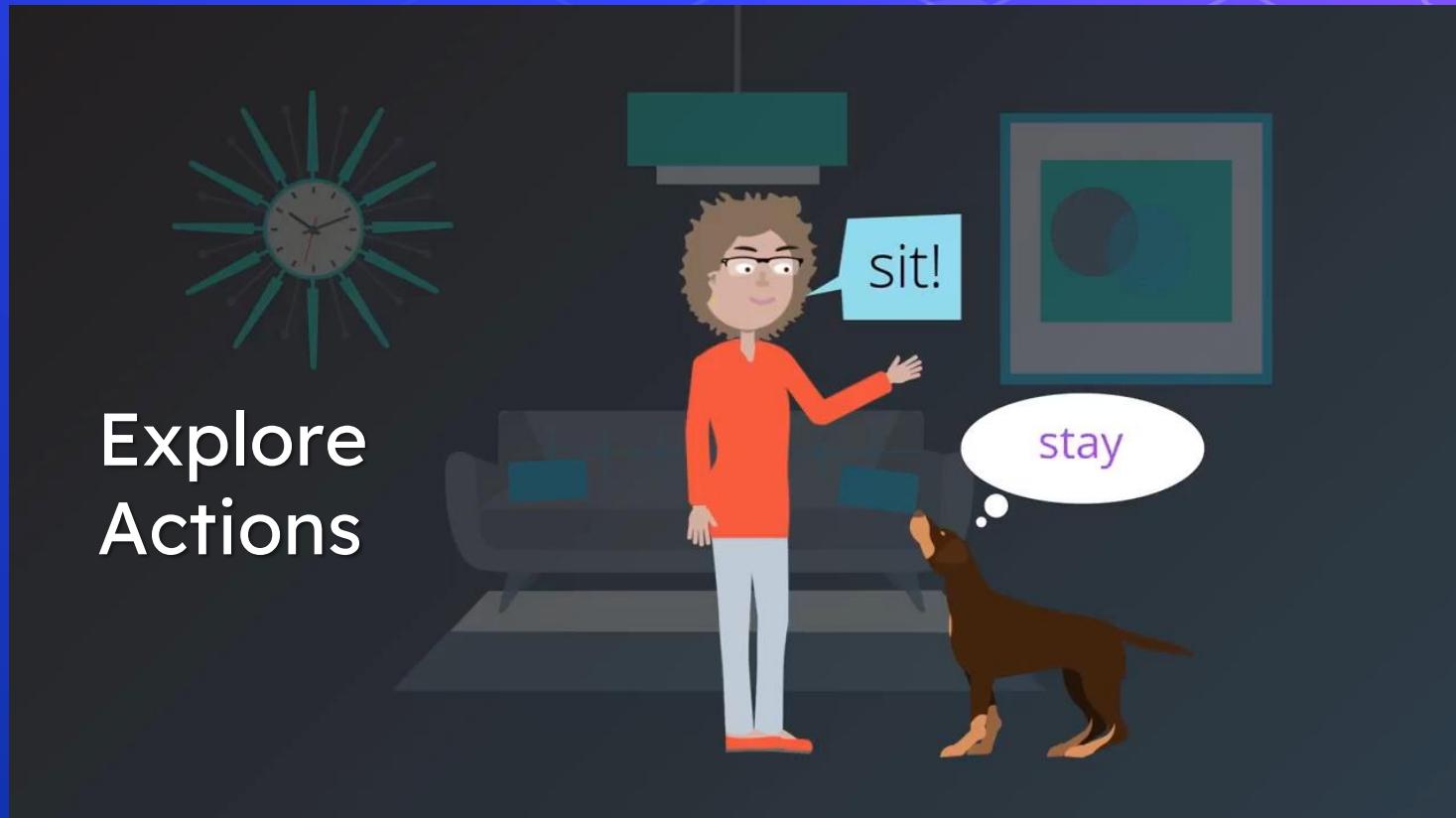


3azooz (Agent)

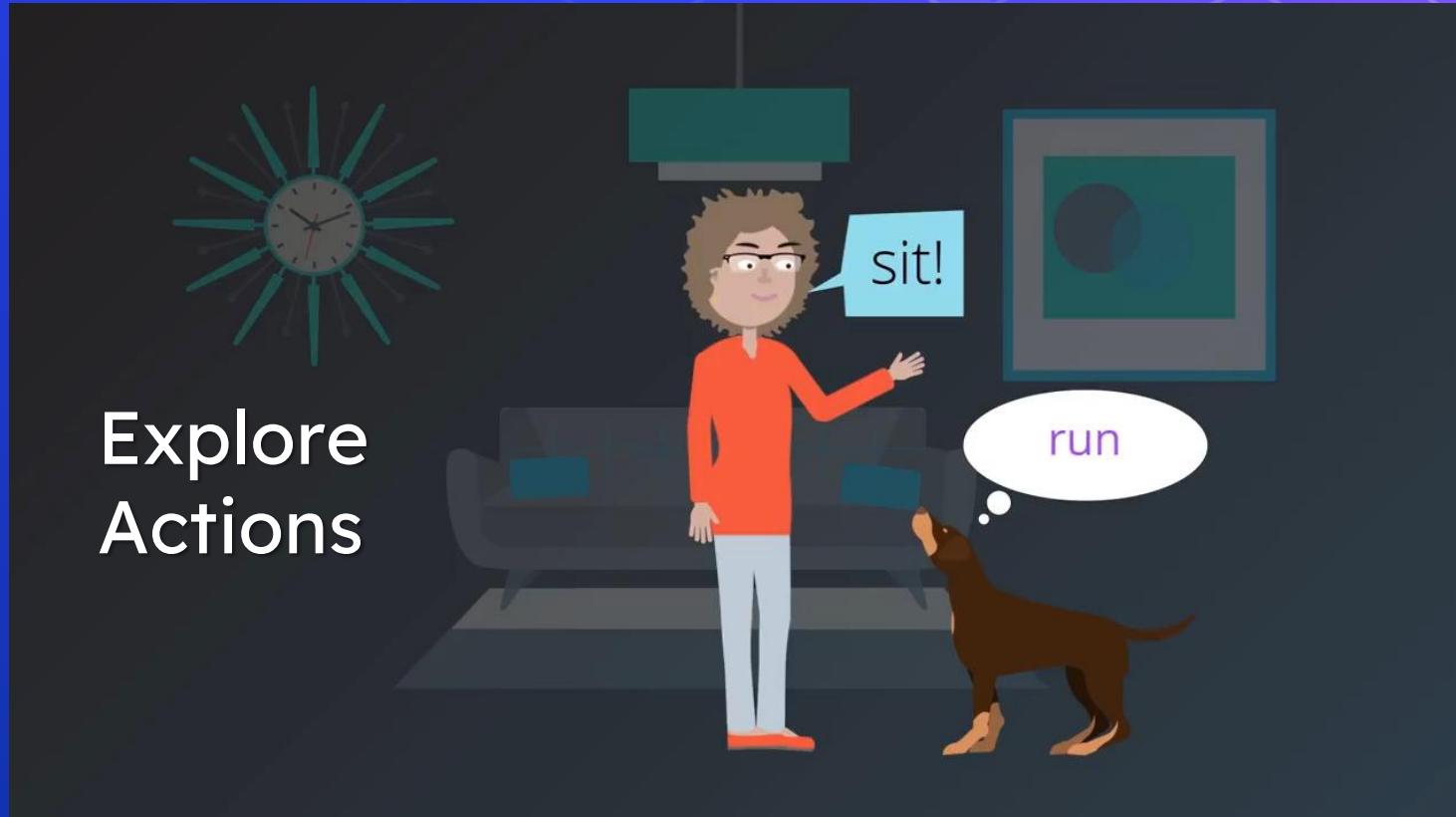
What is RL ?



What is RL ?

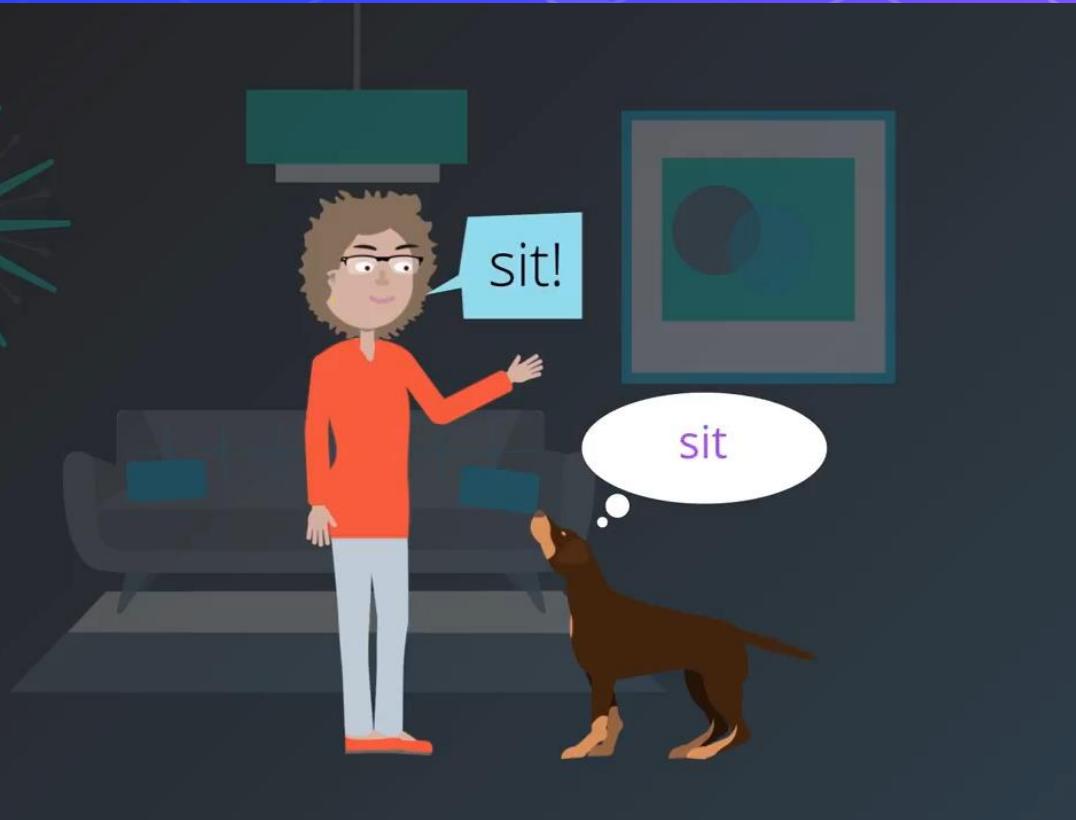


What is RL ?



What is RL ?

Explore
Actions

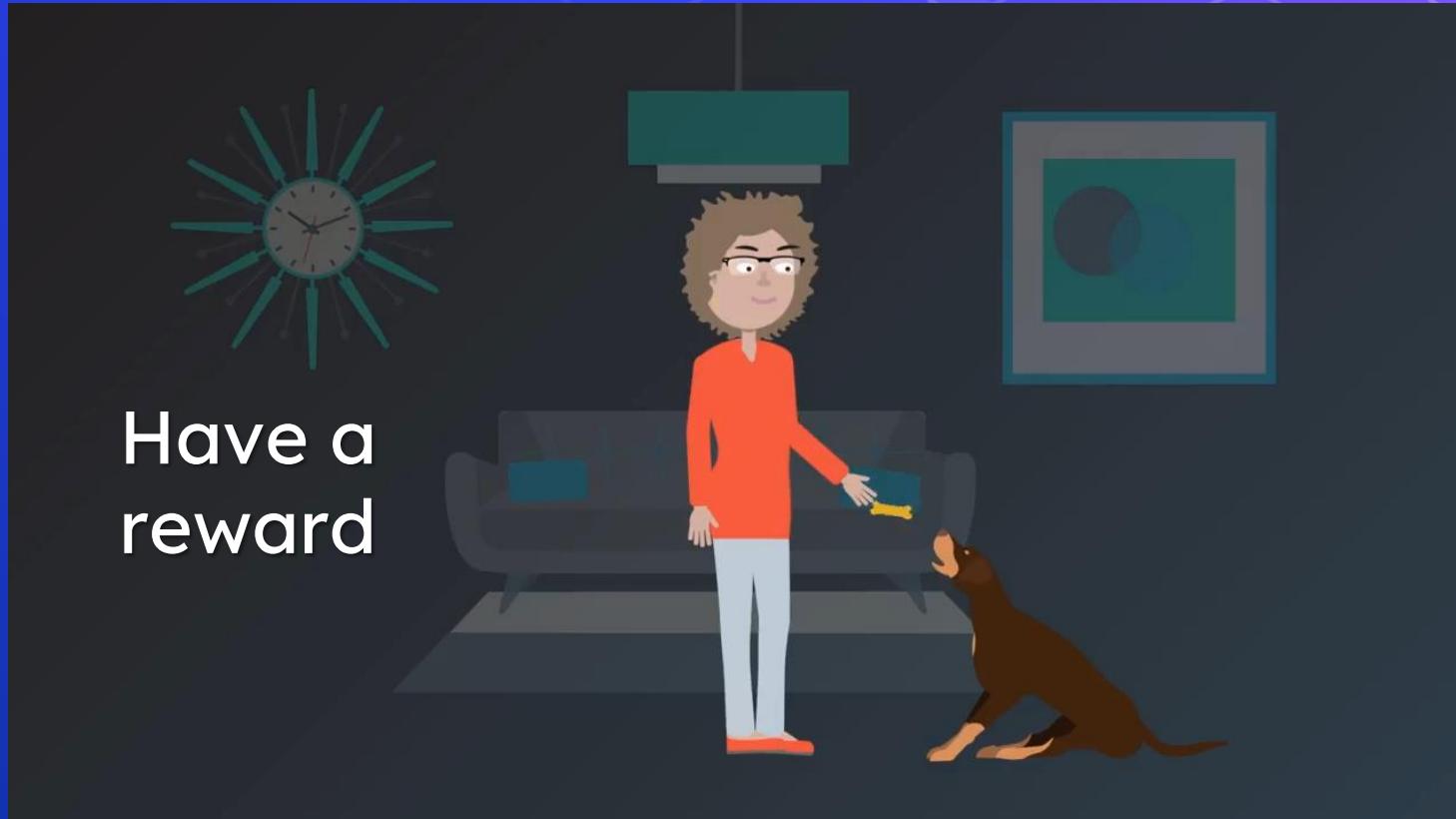


What is RL ?

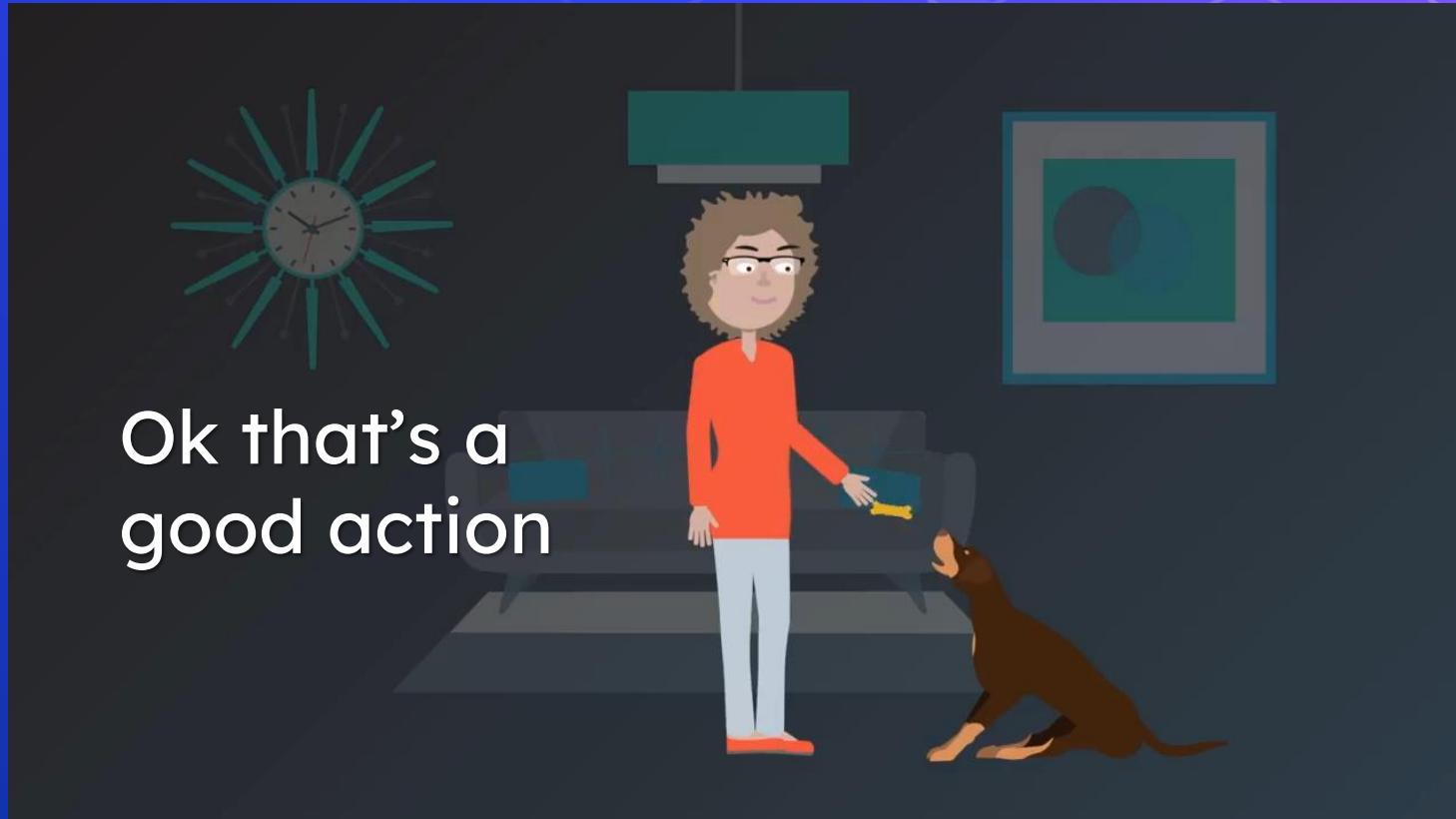
Choose
Random
Action
(Sit)



What is RL ?



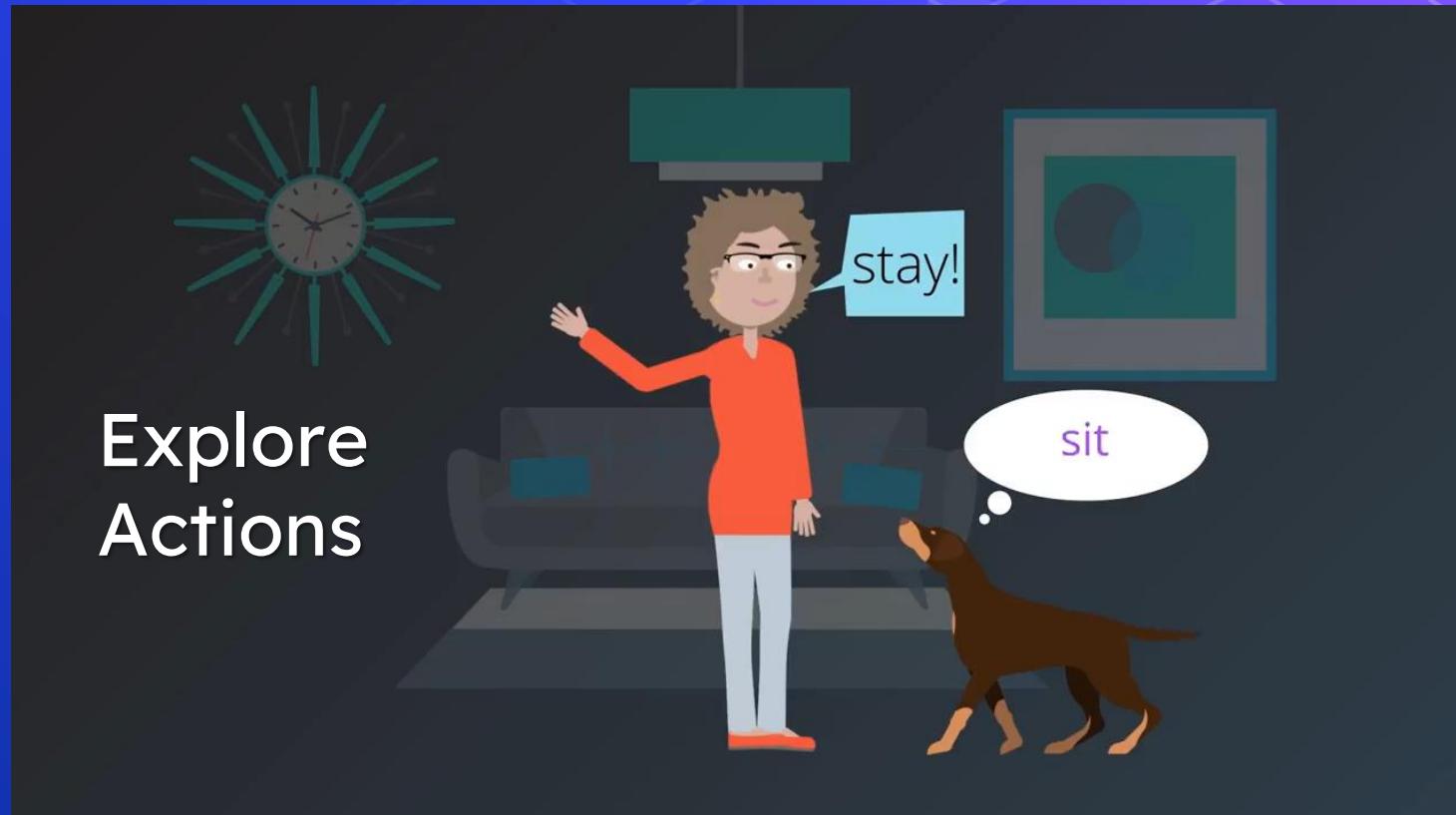
What is RL ?



What is RL ?



What is RL ?



What is RL ?

Explore
Actions



What is RL ?

Explore
Actions

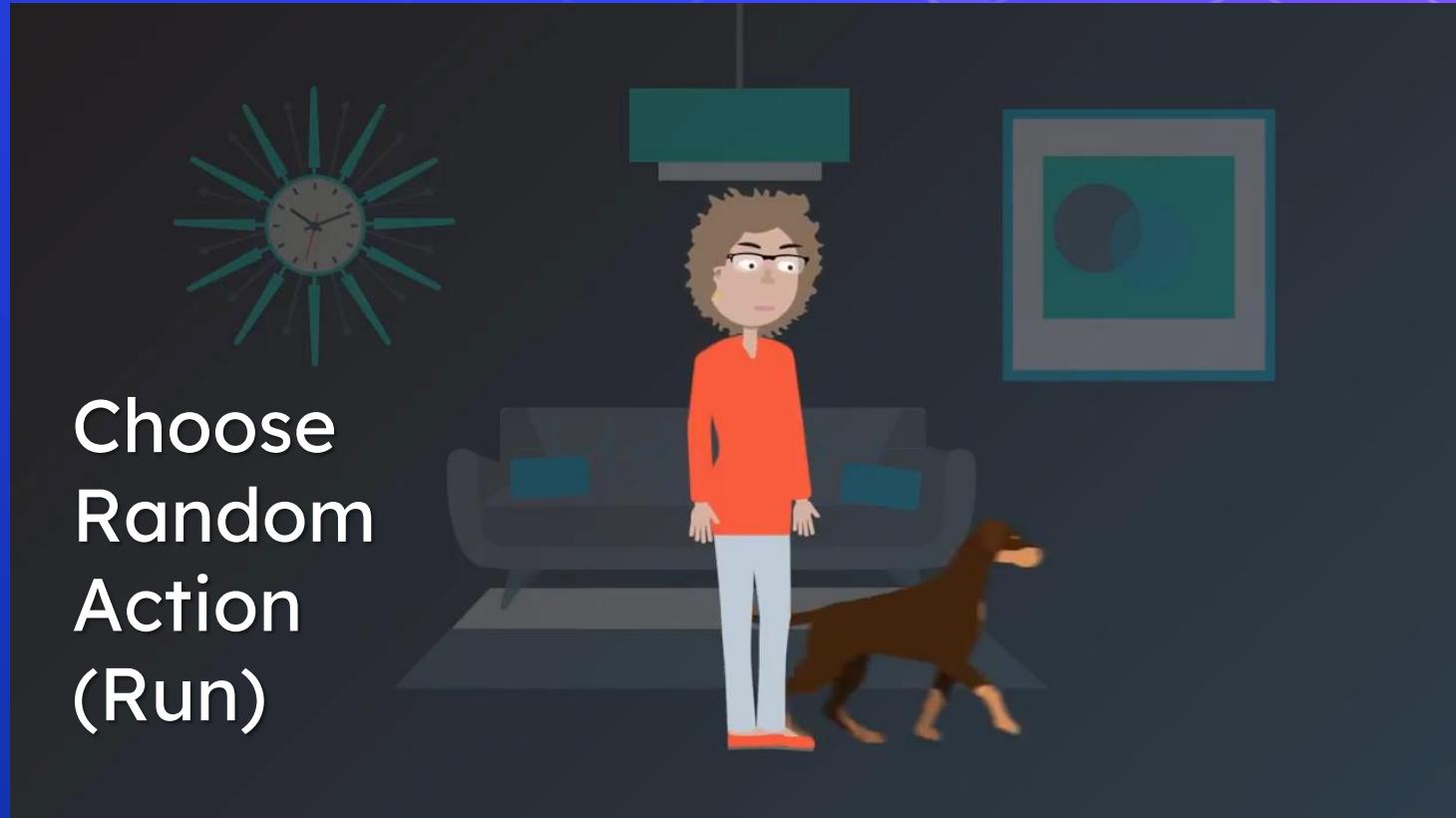


What is RL ?

Choose
Random
Action
(Run)



What is RL ?



What is RL ?



What is RL ?



What is RL ?

Again and again
for a lot of times
(Episodes)



Reinforcement Learning

- What is Reinforcement Learning.
- Applications of Reinforcement Learning.
- Why RL is Freaky.
- Markov Decision Process (MDP).
- MDP Examples.
- MDP Solution.
- Q-Learning Algorithm.
- Open AI Gym environment.
- Coding time >_



Reinforcement Learning Apps

- Robotics.
- Self Driving Cars.
- Game Playing.
- Finance.
- ...



Reinforcement Learning Apps

- Robotics.
- Self Driving Cars.
- Game Playing.
- Finance.
- ...



Reinforcement Learning Apps

- Robotics.
- Self Driving Cars.
- Game Playing.
- Finance.
- ...



Reinforcement Learning Apps

- Robotics.
- Self Driving Cars.
- Game Playing.
- Finance.
- ...



Reinforcement Learning Apps

- Robotics.
- Self Driving Cars.
- Game Playing.
- Finance.**
- ...

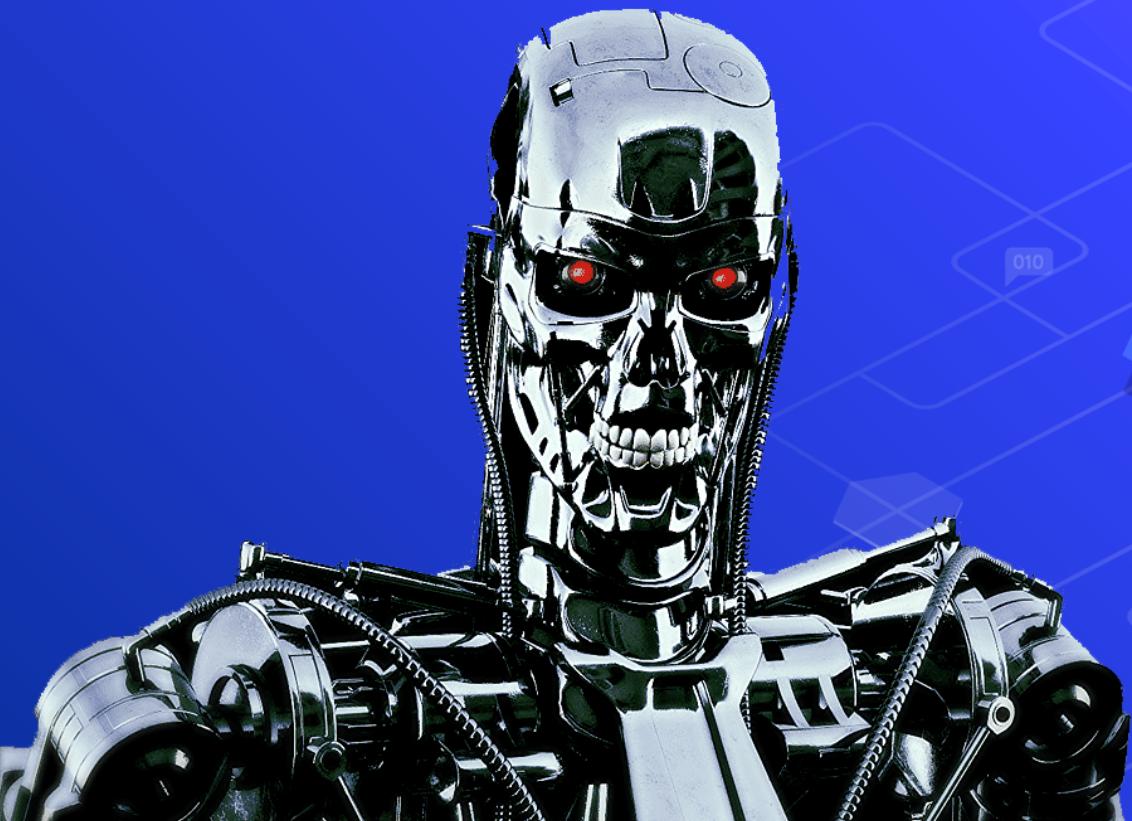


Reinforcement Learning

- What is Reinforcement Learning.
- Applications of Reinforcement Learning.
- Why RL is Freaky.
- Markov Decision Process (MDP).
- MDP Examples.
- MDP Solution.
- Q-Learning Algorithm.
- Open AI Gym environment.
- Coding time >_



Why RL is Creepy ?



Why RL is Creepy ?



Why RL is Creepy ?



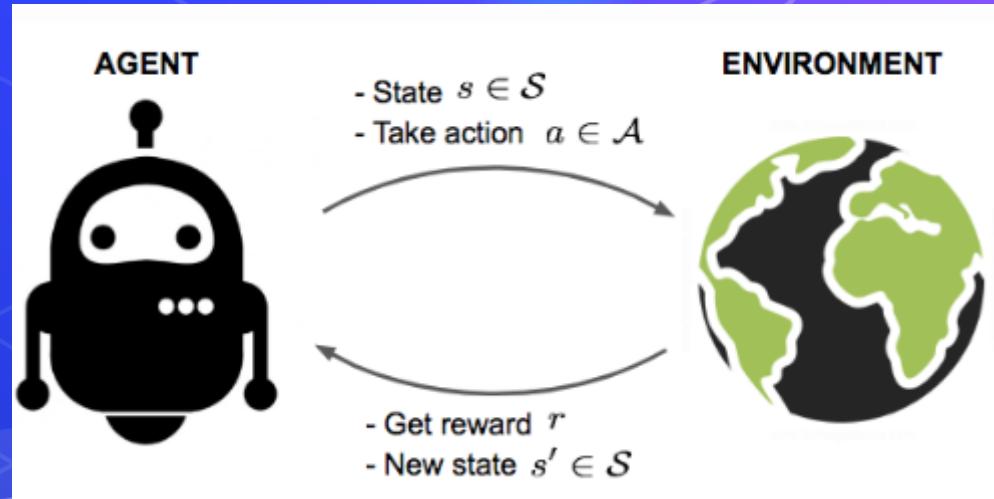
Reinforcement Learning

- What is Reinforcement Learning.
- Applications of Reinforcement Learning.
- Why RL is Freaky.
- Markov Decision Process (MDP).
- MDP Examples.
- MDP Solution.
- Q-Learning Algorithm.
- Open AI Gym environment.
- Coding time >_



Markov Decision process (MDP)

- Agent
- Environment
- Set of States [S]
- Set of Actions [A]
- Expected Rewards for taking an action in a state [$R(s, a)$]
- Goal is to find the optimal action taken in any state to get the maximum rewards (Optimal Policy).



Reinforcement Learning

- What is Reinforcement Learning.
- Applications of Reinforcement Learning.
- Why RL is Freaky.
- Markov Decision Process (MDP).
- **MDP Examples.**
- MDP Solution.
- Q-Learning Algorithm.
- Open AI Gym environment.
- Coding time >_



Markov Decision process (MDP) - Maze

- Set of States

- Robot position in grid

- Set of Actions

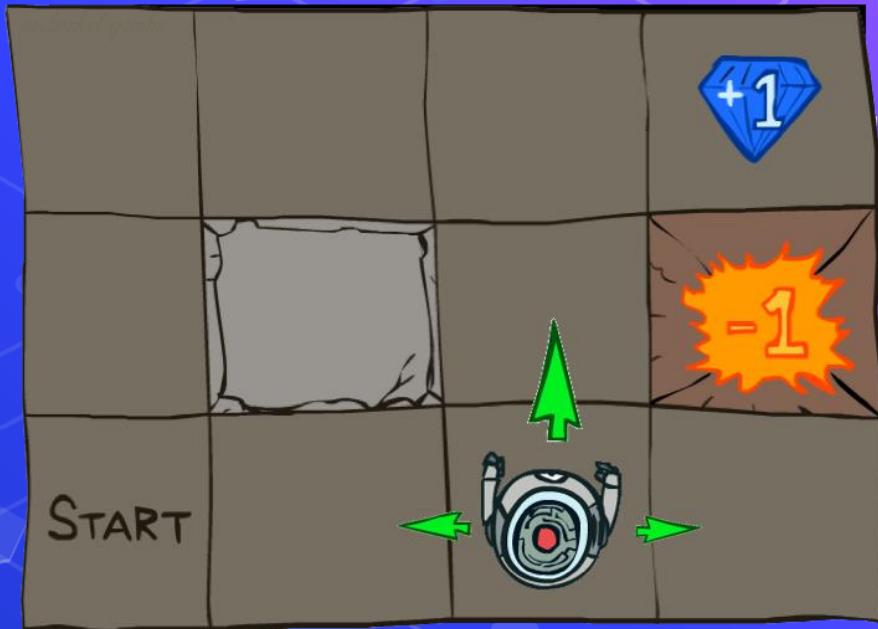
- Up, Down, Right, Left

- Rewards

- $(+1)$ if get the diamond.

- (-1) if go to the fire.

- ...



Markov Decision process (MDP) - Robot

- Set of States
 - Camera and Sensors.

- Set of Actions
 - Torque on the joints to walk or jump.

- Rewards
 - (+10) if not fall while jumping.
 - (-5) if apply high torque on joints.
 - ...



Markov Decision process (MDP) - Self driving car

- Set of States

- Camera and Sensors.

- Set of Actions

- Steer, Throttle, Break.

- Rewards

- (-1000) if hit human.

- (-500) if hit a car.

- ...



Reinforcement Learning

- What is Reinforcement Learning.
- Applications of Reinforcement Learning.
- Why RL is Freaky.
- Markov Decision Process (MDP).
- MDP Examples.
- **MDP Solution.**
- Q-Learning Algorithm.
- Open AI Gym environment.
- Coding time >_



MDP Solution.

- Policy.
- Exploration vs Exploitation.
- Reward Function.
- Discounted Reward Factor.
- Q-Values.



MDP Solution.

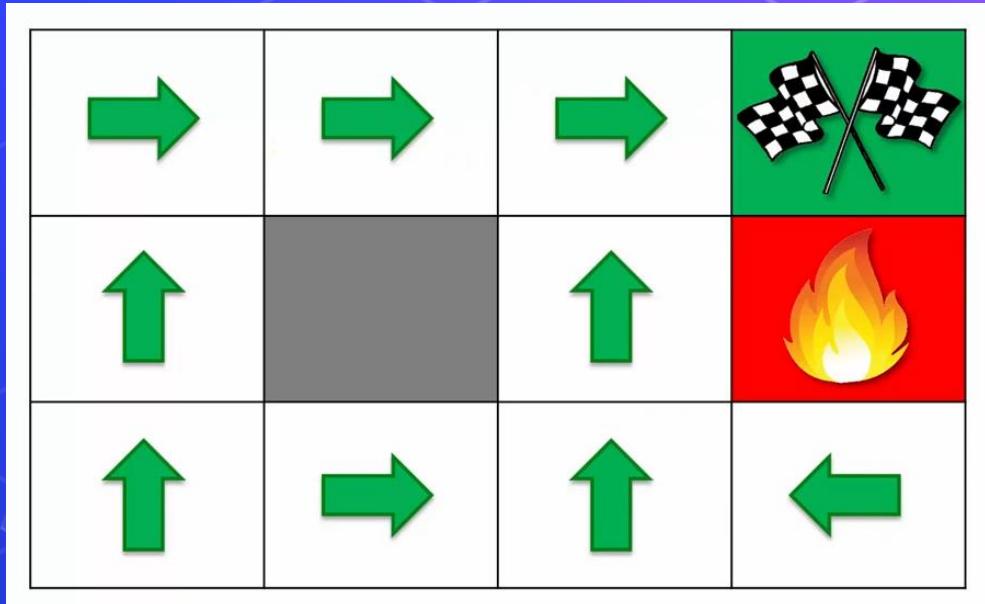
- Policy.
- Exploration vs Exploitation.
- Reward Function.
- Discounted Reward Factor.
- Q-Values.



Policy

The set of actions for all the states that solves the MDP and get to the Goal.

The **Optimal** policy is the best actions led to the maximum reward for solving the goal.



MDP Solution.

- Policy.
- Exploration vs Exploitation.
- Reward Function.
- Discounted Reward Factor.
- Q-Values.



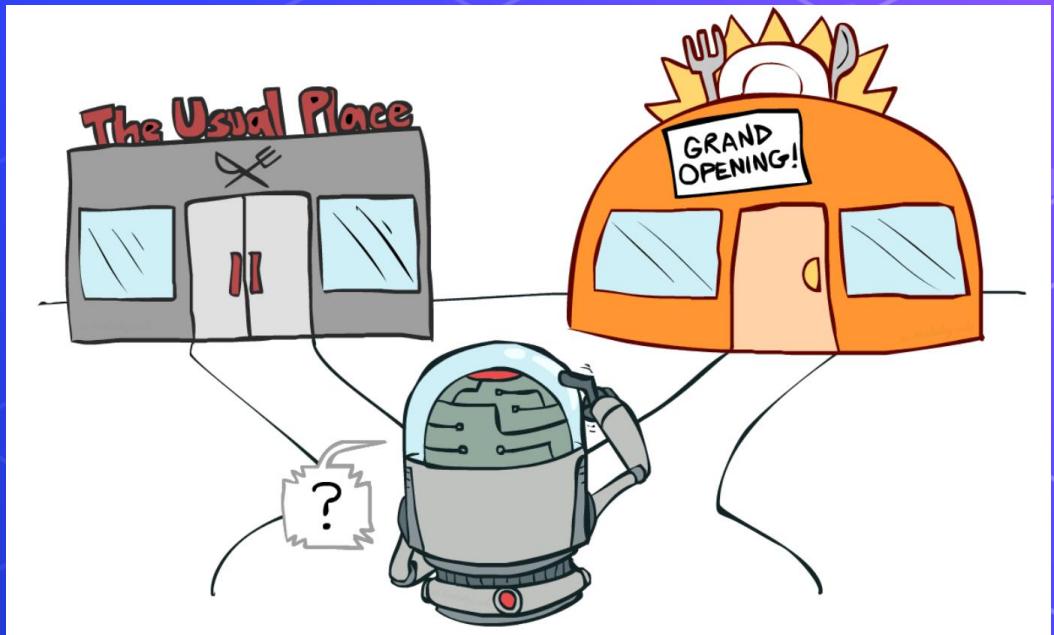
Exploration vs Exploitation

Exploration

Keep taking random actions to gain more knowledge about the environment.

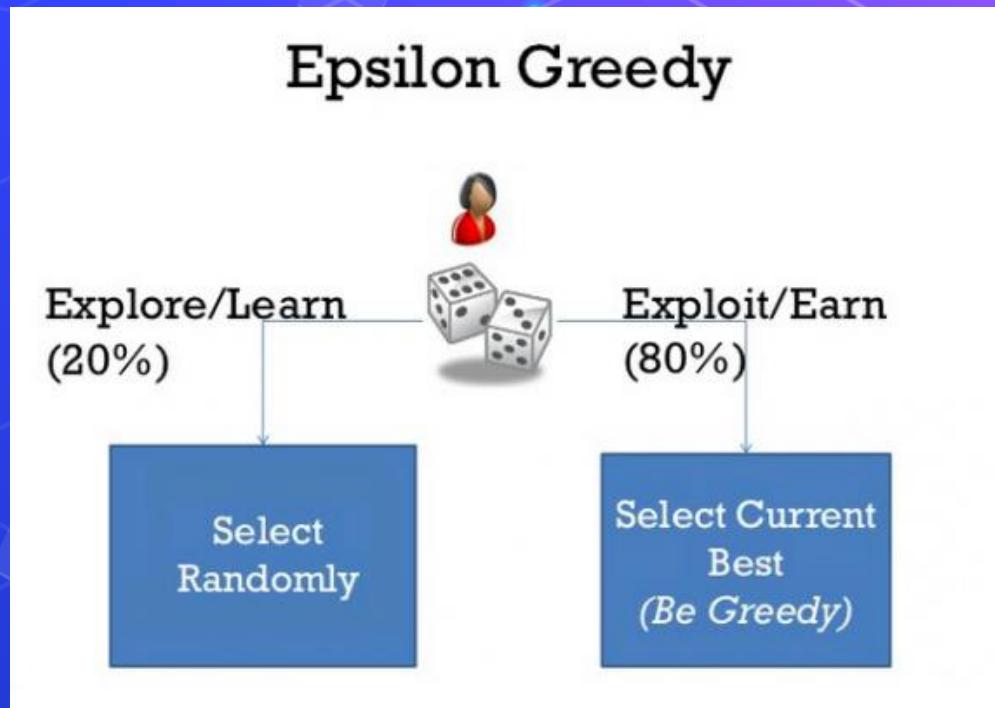
Exploitation

Use your gained knowledge to take actions based on your collected experience.



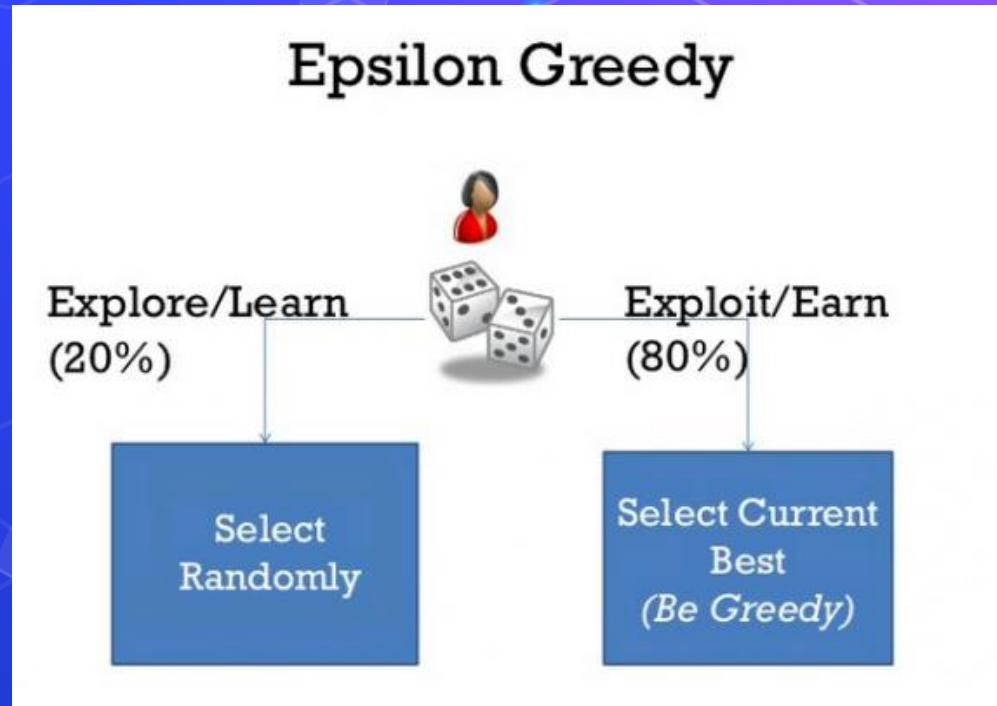
Exploration vs Exploitation (Epsilon-Greedy algorithm)

- Epsilon = 0.2 (20%) having random action.
- $(1 - \text{Epsilon}) = 0.8$ (80%) the agent became **greedy** for having an action from collected experience.



Exploration vs Exploitation (Epsilon-Greedy algorithm)

- At first we set epsilon to 1 (100%) so at first the agent will explore the environment (taking random actions).
- Then gradually decaying the epsilon value (start explore then gradually exploit while collecting knowledge).
- At the end epsilon will be ~ 0 so that the agent now has a great knowledge about the environment and no need to take random actions any more.



Exploration vs Exploitation (Epsilon-Greedy algorithm)

Implementation

- 1- Generate a random number between 0 & 1.
- 2- If the number is > current epsilon value then exploit the knowledge.
- 3- If the number is < current epsilon value then explore random action.
- 4- Decay the epsilon over time.



```
1 if random_num > epsilon:  
2     # choose action via exploitation  
3 else:  
4     # choose action via exploration
```



MDP Solution

- Policy.
- Exploration vs Exploitation.
- Reward Function.
- Discounted Reward Factor.
- Q-Values.



Reward Function

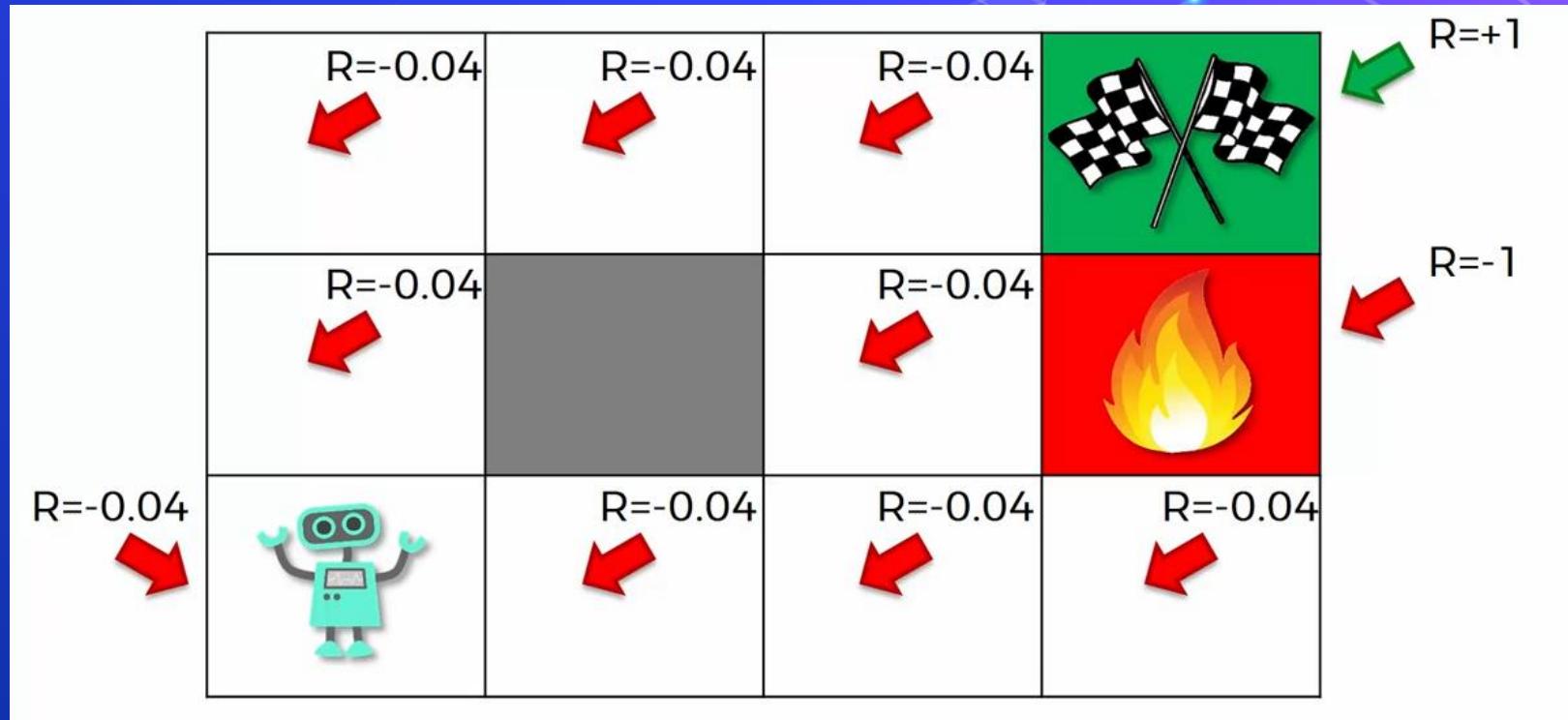
The goal of an agent in an MDP is to maximize its cumulative rewards. For this, we introduce the concept of the expected return of the rewards at a given time step.

The expected return of the rewards is what's driving the agent to make the decisions it makes.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

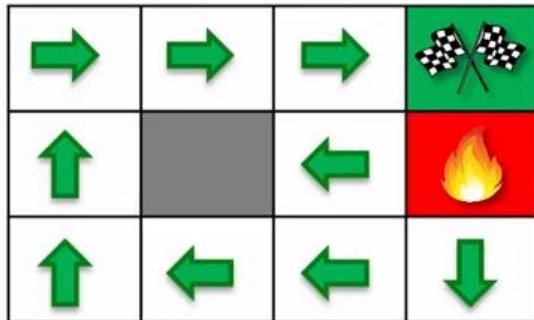


Reward Function

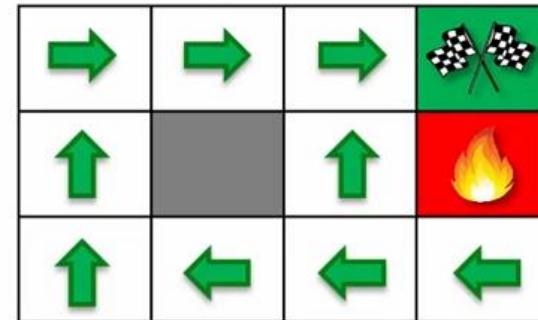


Reward Function

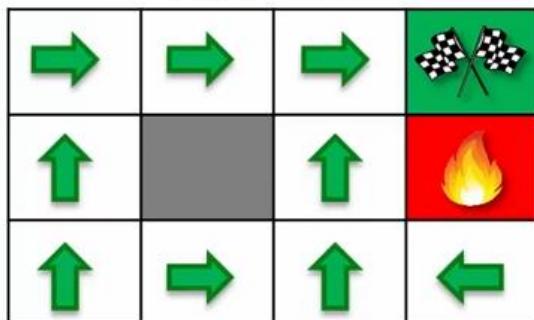
$R(s)=0$



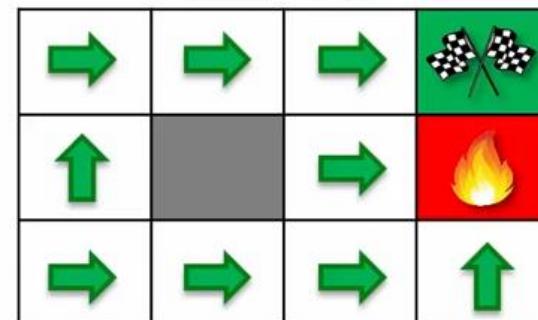
$R(s)=-0.04$



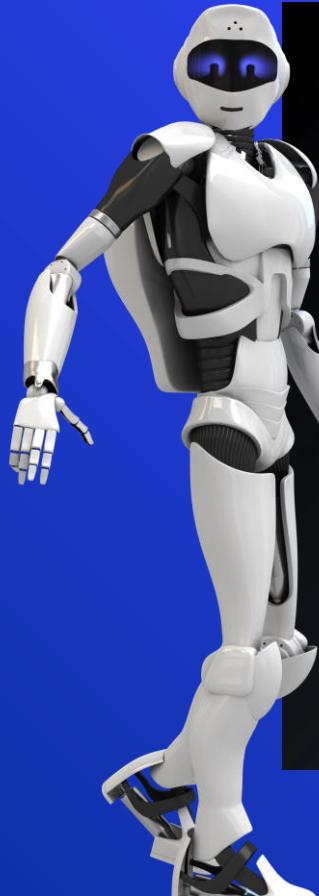
$R(s)=-0.5$



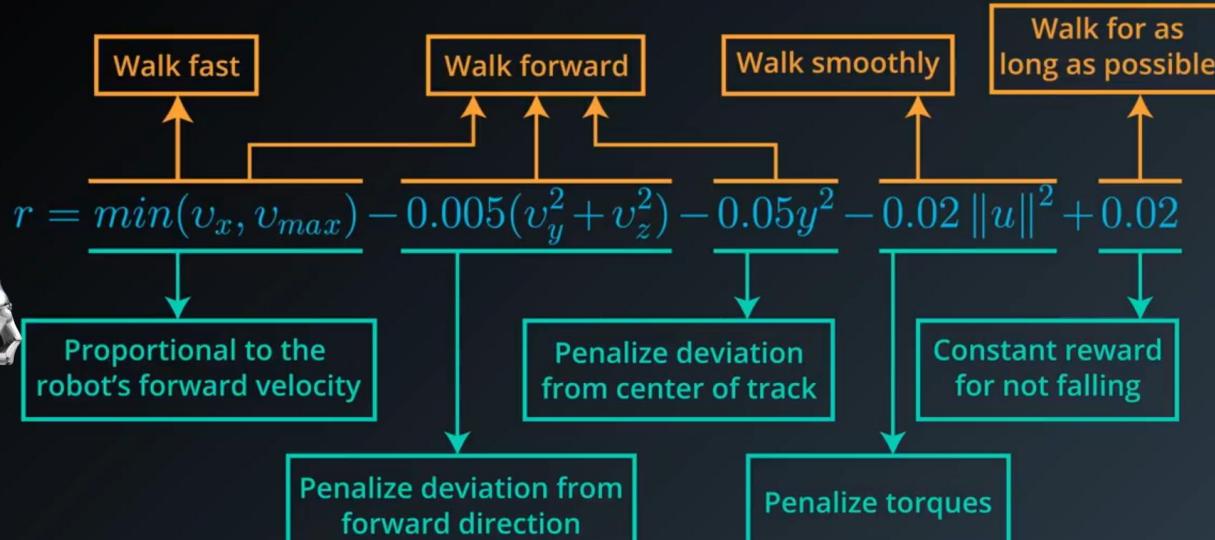
$R(s)=-2.0$



Reward Function



What are the rewards?



MDP Solution

- Policy.
- Exploration vs Exploitation.
- Reward Function.
- Discounted Reward Factor.**
- Q-Values.



Discounted Reward Factor

Do you prefer to have one Million
Dollars now or tomorrow ?!



Discounted Reward Factor

For sure you will choose having the money now rather than tomorrow, because immediate rewards are the most valuable and more important than the next future rewards.

So to make the agent cares a lot of immediate rewards more than the future rewards, we added the discount factor ($\gamma = 0.9$).

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\ &= R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$



MDP Solution

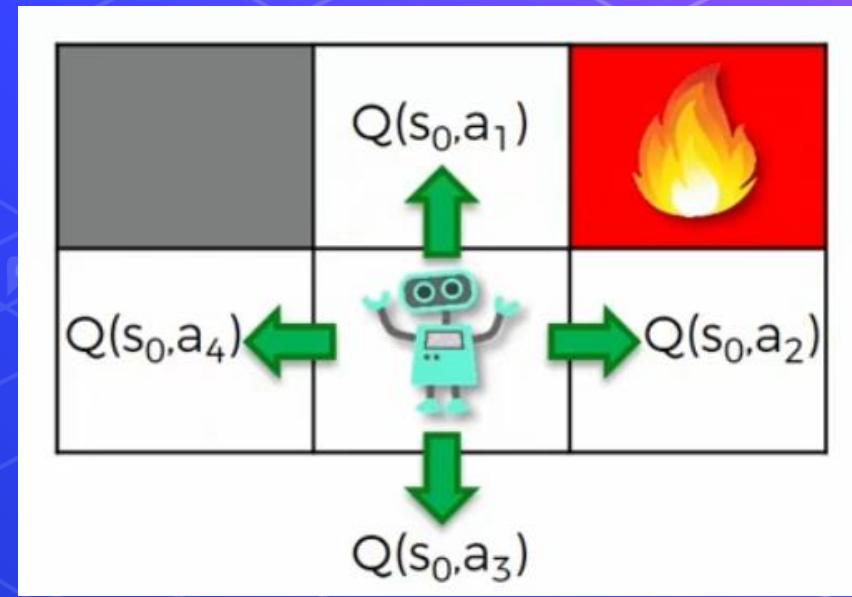
- Policy.
- Exploration vs Exploitation.
- Reward Function.
- Discounted Reward Factor.
- Q-Values.**



Q-Values

Q-Values for a state are the probabilities of taking an action between set of actions in a given state.

For example: in this state the robot have 4 actions (up, down, right, left) for this state, we make 4 q-values one for each action, the highest q-value for an action is the optimal action to take.



Reinforcement Learning

- What is Reinforcement Learning.
- Applications of Reinforcement Learning.
- Why RL is Freaky.
- Markov Decision Process (MDP).
- MDP Examples.
- MDP Solution.
- [Q-Learning Algorithm.](#)
- Open AI Gym environment.
- Coding time >_



Q Learning Algorithm

- 1. Initialize $Q(s, a)$ by setting all of the Q-Values for each state equal to zero, $\epsilon = 1$.
- 2. Observe the current state.
- 3. Based on the epsilon-greedy policy, choose an action (explore or exploit).
- 4. Take action A and observe the resulting reward, and the new state of the environment S'.
- 5. Update $Q(s, a)$ based on the update rule.

$$q^{new}(s, a) = (1 - \alpha) \underbrace{q(s, a)}_{\text{old value}} + \alpha \overbrace{\left(R_{t+1} + \gamma \max_{a'} q(s', a') \right)}^{\text{learned value}}$$

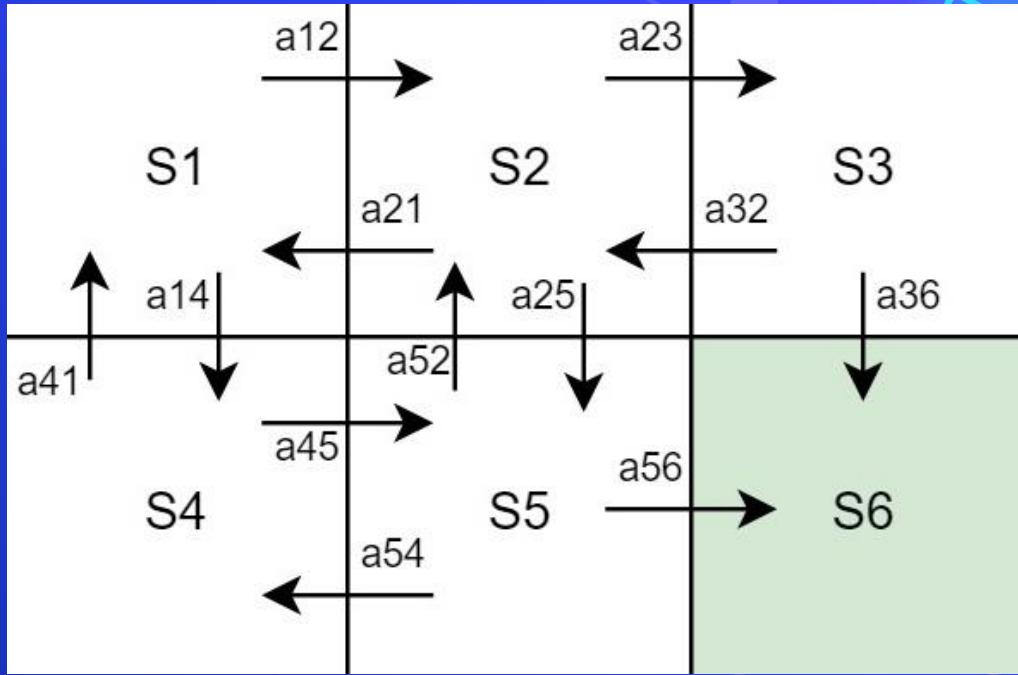
Where $R[t+1]$ is the reward, alpha is the learning rate and gamma is the discount rate = 0.9, $Q(s', a')$ is the maximum q-value for an action at the next state.

- 6. $S = S'$ & Decay ϵ then Repeat steps 2–6 until convergence.



Q Learning Algorithm (Grid World)

Consider a grid world environment that a robot wants to reach the flag at state 6.



Q Learning Algorithm (Grid World)

1. Initialize problem by this settings:

- All of the Q-Values for each state equal to zero.
- Epsilon = 1.
- Gamma = 0.5.
- Alpha = 1.
- The flag reward is +100 at state S6 otherwise zero.

When alpha = 1 for simplicity, the equation became

$$R_{t+1} + \gamma \max_{a'} q(s', a')$$

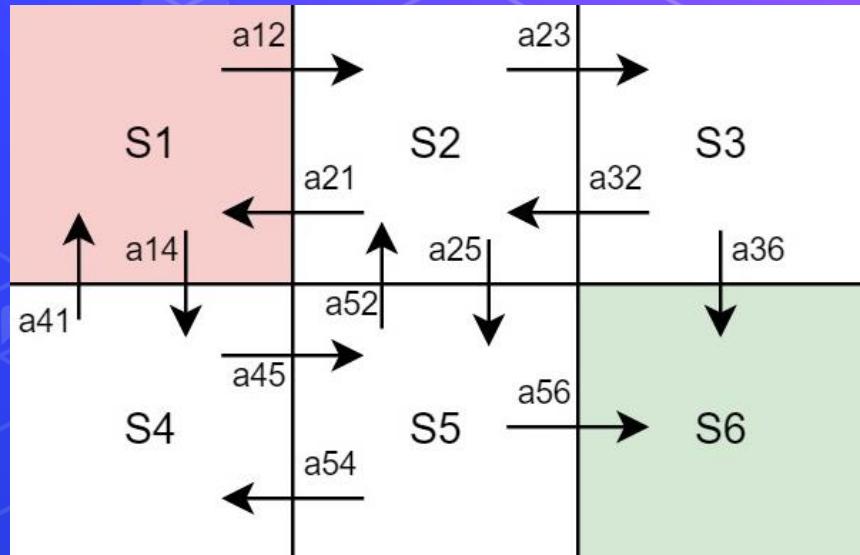
Q(S, A)	Value
Q(S1, A12)	0
Q(S1, A14)	0
Q(S2, A21)	0
Q(S2, A23)	0
Q(S2, A25)	0
Q(S3, A32)	0
Q(S3, A36)	0
Q(S4, A41)	0
Q(S4, A45)	0
Q(S5, A54)	0
Q(S5, A52)	0
Q(S5, A56)	0

Q Learning Algorithm (Grid World)

2. Observe the current state S1 (the red state), available actions are (a12, a14) lets say we took action a12 so we became at state S' = S2.

3. Calculate q_new(S1, a12) with the equation using q-values from the table.

$$R_{t+1} + \gamma \max_{a'} q(s', a')$$

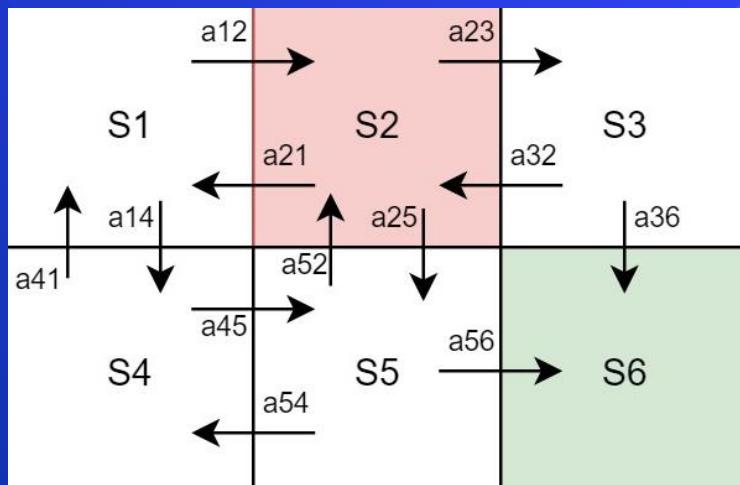


$$\begin{aligned} q_{\text{new}}(S1, a12) &= r + 0.5 * \max[q(S2, a21), q(S2, a23), q(S2, a25)] \\ &= 0 + 0.5 * 0 = 0 \end{aligned}$$

Q Learning Algorithm (Grid World)

4- Update the q table with new $Q(S1, A12)$ value = 0.

5- New State became S2.



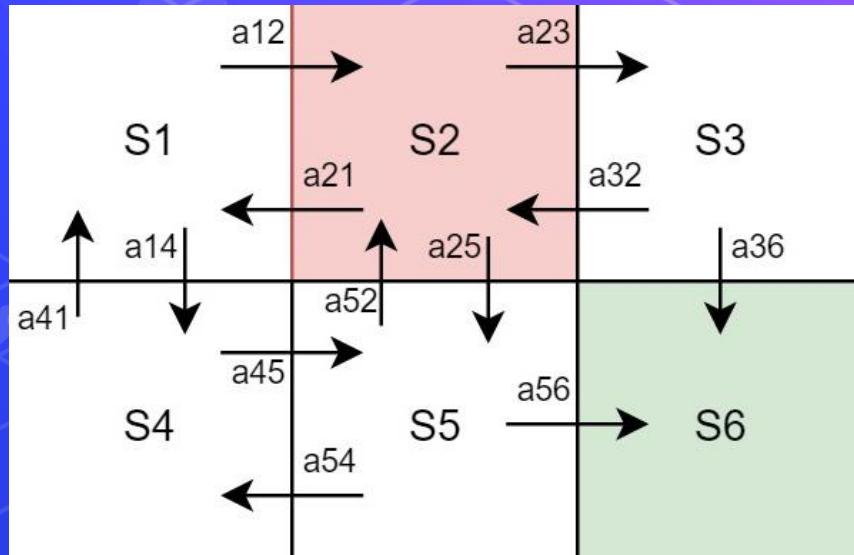
$Q(S, A)$	Value
$Q(S1, A12)$	0
$Q(S1, A14)$	0
$Q(S2, A21)$	0
$Q(S2, A23)$	0
$Q(S2, A25)$	0
$Q(S3, A32)$	0
$Q(S3, A36)$	0
$Q(S4, A41)$	0
$Q(S4, A45)$	0
$Q(S5, A54)$	0
$Q(S5, A52)$	0
$Q(S5, A56)$	0

Q Learning Algorithm (Grid World)

6. Observe the current state S2 (the red state), available actions are (a21, a23, a25) lets say we took action a23 so we became at state S' = S3.

7. Calculate q_new(S2, a23) with the equation using q-values from the table.

$$R_{t+1} + \gamma \max_{a'} q(s', a')$$

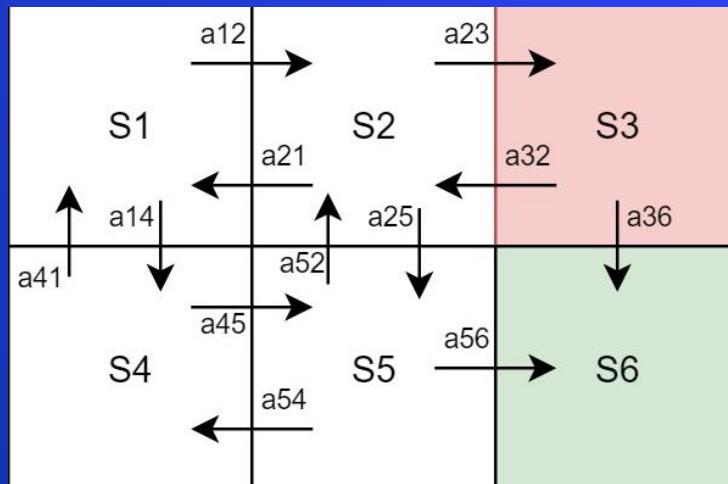


$$\begin{aligned} q_{\text{new}}(S2, a23) &= r + 0.5 * \max[q(S3, a32), q(S3, a36)] \\ &= 0 + 0.5 * 0 = 0 \end{aligned}$$

Q Learning Algorithm (Grid World)

8- Update the q table with new $Q(S_2, A_{23})$ value = 0.

9- New State became S_3 .



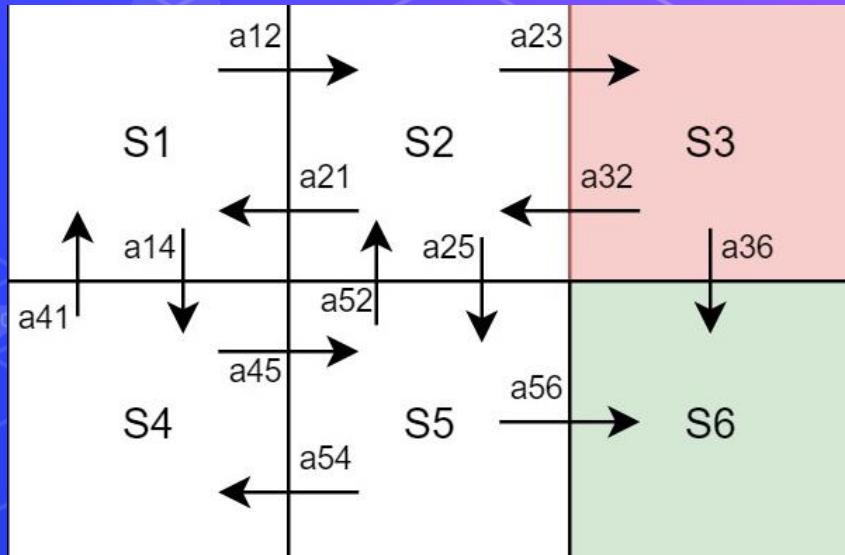
Q(S, A)	Value
Q(S1, A12)	0
Q(S1, A14)	0
Q(S2, A21)	0
Q(S2, A23)	0
Q(S2, A25)	0
Q(S3, A32)	0
Q(S3, A36)	0
Q(S4, A41)	0
Q(S4, A45)	0
Q(S5, A54)	0
Q(S5, A52)	0
Q(S5, A56)	0

Q Learning Algorithm (Grid World)

10. Observe the current state S3 (the red state), available actions are (a32, a36) lets say we took action a36 so we became at state S' = S6.

11. Calculate q_new(S3, a36) with the equation using q-values from the table.

$$R_{t+1} + \gamma \max_{a'} q(s', a')$$

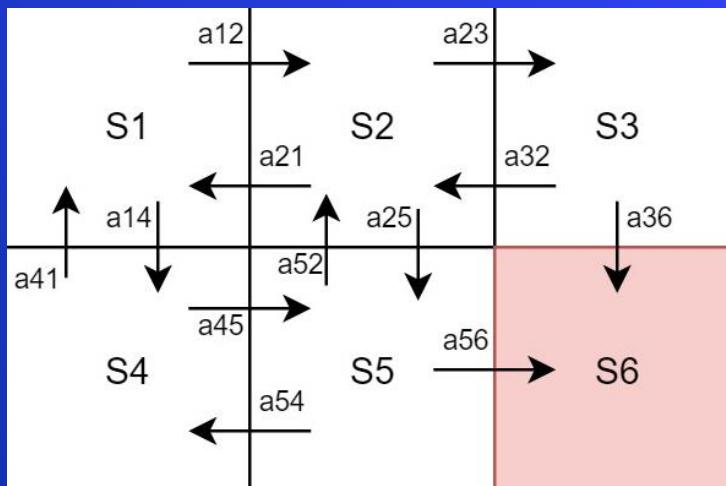


q_new(S3, a36) = r = 100, because there is no actions in S' -> S6 (Goal)

Q Learning Algorithm (Grid World)

12- Update the q table with new $Q(S_3, A_{36})$ value = 100.

13- New State became S_6 (Goal or Terminal State) and the episode ends.



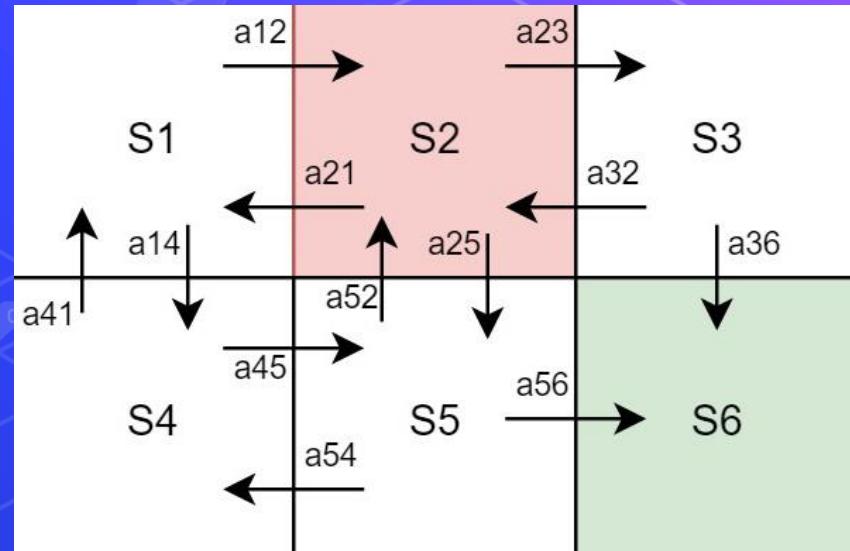
Q(S, A)	Value
$Q(S_1, A_{12})$	0
$Q(S_1, A_{14})$	0
$Q(S_2, A_{21})$	0
$Q(S_2, A_{23})$	0
$Q(S_2, A_{25})$	0
$Q(S_3, A_{32})$	0
$Q(S_3, A_{36})$	100
$Q(S_4, A_{41})$	0
$Q(S_4, A_{45})$	0
$Q(S_5, A_{54})$	0
$Q(S_5, A_{52})$	0
$Q(S_5, A_{56})$	0

Q Learning Algorithm (Grid World)

14. After Episode 1 ends, Start Episode 2 with Random State for example S2 (the red state), available actions are (a21, a23, a25) lets say we took action a23 so we became at state S' = S3.

11. Calculate q_new(S2, a23) with the equation using q-values from the table.

$$R_{t+1} + \gamma \max_{a'} q(s', a')$$

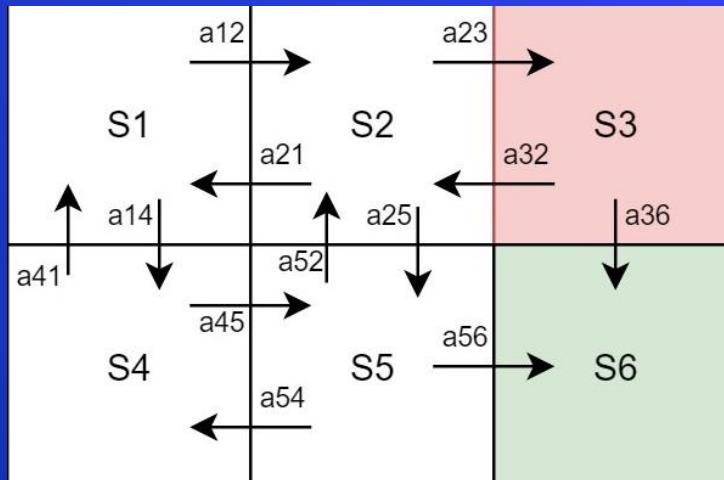


$$\begin{aligned} q_{\text{new}}(S2, a23) &= r + 0.5 * \max[q(S3, a32), q(S3, a36)] \\ &= 0 + 0.5 * 100 = 50 \end{aligned}$$

Q Learning Algorithm (Grid World)

12- Update the q table with new $Q(S_2, A_{23})$ value = 50.

13- New State became S_3 , and So on for N episodes.



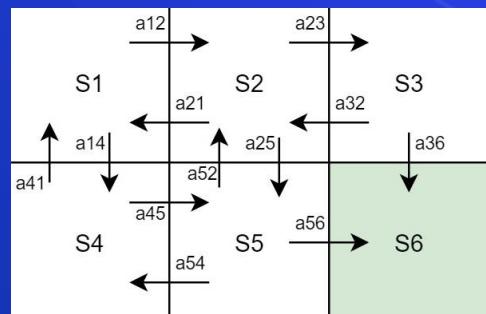
$Q(S, A)$	Value
$Q(S_1, A_{12})$	0
$Q(S_1, A_{14})$	0
$Q(S_2, A_{21})$	0
$Q(S_2, A_{23})$	50
$Q(S_2, A_{25})$	0
$Q(S_3, A_{32})$	0
$Q(S_3, A_{36})$	100
$Q(S_4, A_{41})$	0
$Q(S_4, A_{45})$	0
$Q(S_5, A_{54})$	0
$Q(S_5, A_{52})$	0
$Q(S_5, A_{56})$	0

Q Learning Algorithm (Grid World)

After a lot of episodes of updating the Q-Values Table,

Lets say after 200 episode we have these values on the left, so we finally know the policy to solve the grid world problem.

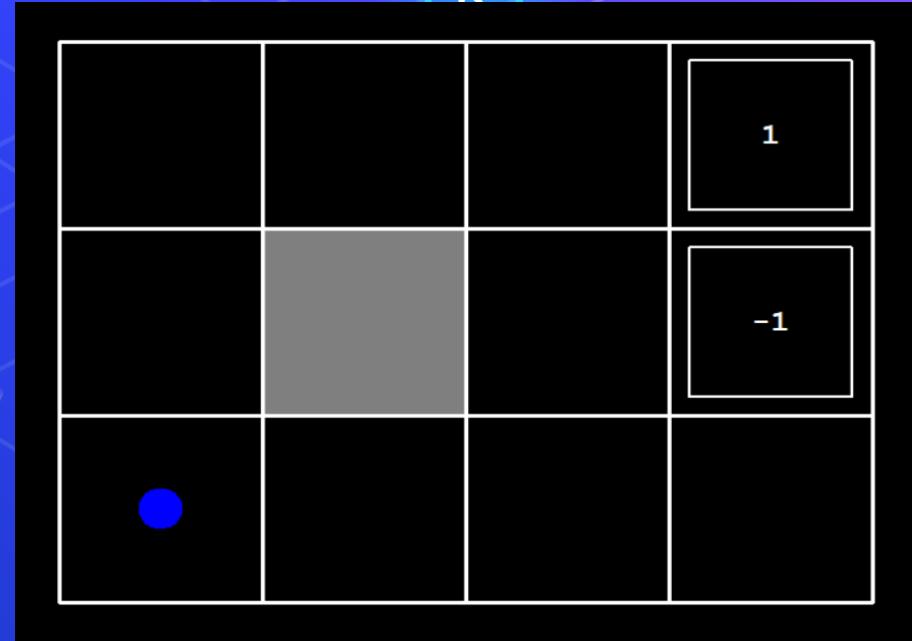
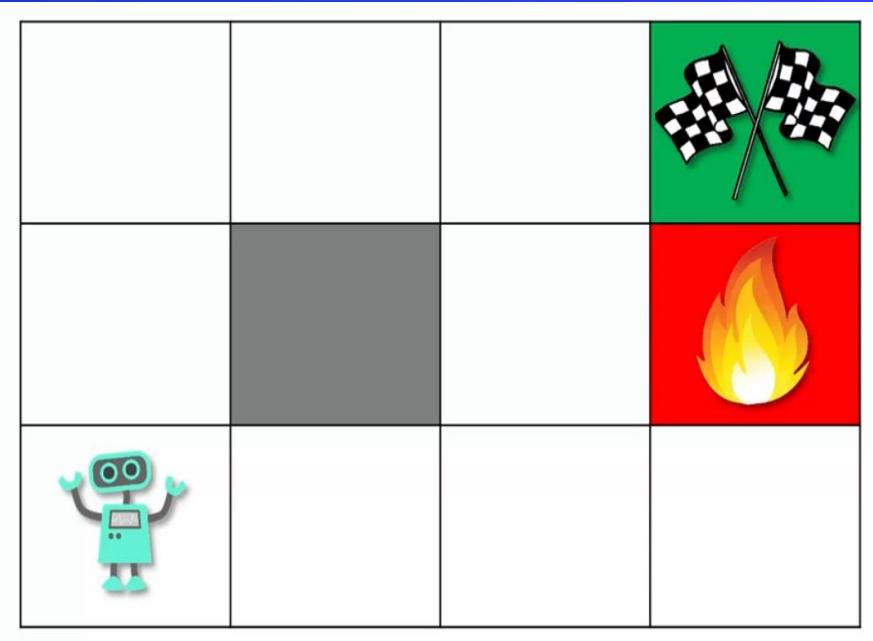
- At State 1 choose Action A12 or A14 = 25.
- At State 2 choose Action A23 = 50.
- At State 3 choose Action A36 = 100.
- At State 4 choose Action A45 = 50.
- At State 5 choose Action A56 = 100.



Q(S, A)	Value
Q(S1, A12)	25
Q(S1, A14)	25
Q(S2, A21)	12.5
Q(S2, A23)	50
Q(S2, A25)	25
Q(S3, A32)	25
Q(S3, A36)	100
Q(S4, A41)	12.5
Q(S4, A45)	50
Q(S5, A54)	25
Q(S5, A52)	25
Q(S5, A56)	100

Q Learning Algorithm (Another Grid World)

Consider a grid world environment that a robot wants to reach the flag and stay away from fire.



Q Learning Algorithm (Another Grid World)

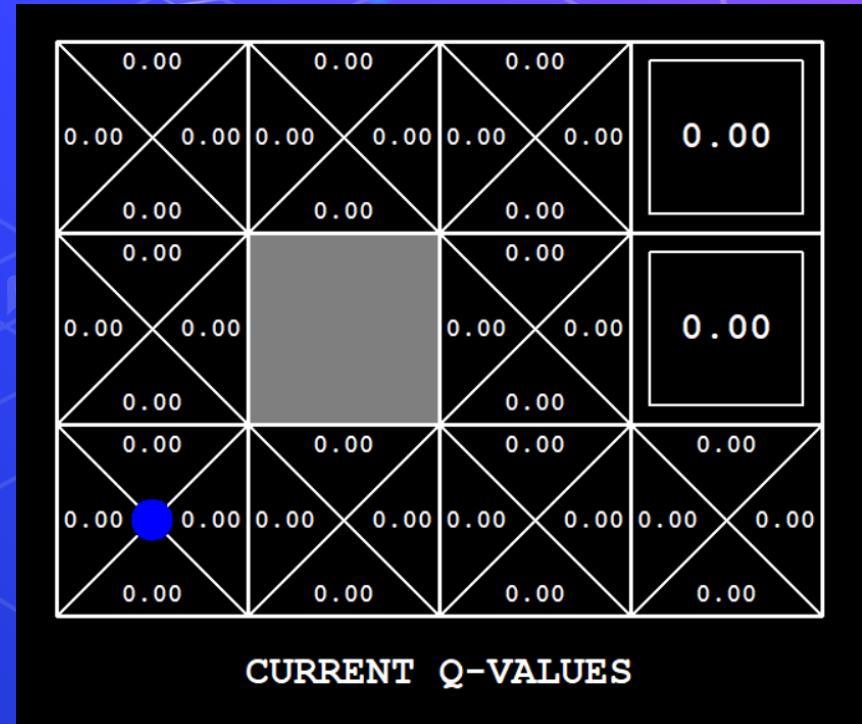
1. Initialize $Q(s, a)$ by setting all of the Q-Values for each state equal to zero, ϵ = 1, γ = 0.9, α = 0.7, the flag reward is +1 & fire penalty is -1 and every step has penalty 0, number of episodes = 400.

2. Observe the current state (the blue circle).

3. Based on the epsilon-greedy policy, choose an action, At first ϵ = 1, so the agent will take random actions for the first episodes.

4. after a while the agent gains some knowledge so it should be greedy and exploit its knowledge, after a lot of episodes ϵ became very small thus a little explored random actions is taken.

5. After taking an action, observe the new state and the resulting reward.



Q Learning Algorithm (Another Grid World)

6. The episode ends when the agent reaches the flag or the fire, lets say the agent is in state [0, 2] it chooses action right then it reaches the flag and gains the reward 1, so it knows that the action (right) led to the flag, it's a good action so updates the q-value of the action with the equation below

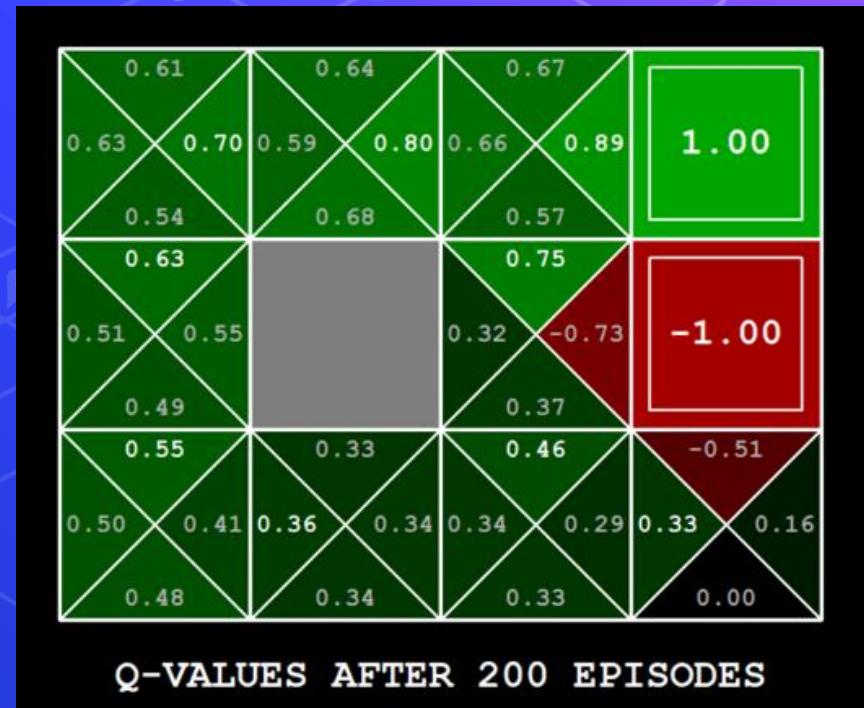
$$q_{\text{new}}([0, 2], \text{Right}) = (1 - 0.7) * 0 + 0.7 (1) = 0.7$$

$$q^{new}(s, a) = (1 - \alpha) \underbrace{q(s, a)}_{\text{old value}} + \alpha \overbrace{\left(R_{t+1} + \gamma \max_{a'} q(s', a') \right)}^{\text{learned value}}$$

7. At a next step we ended for example at state [0, 1] and took action right, then the new q is

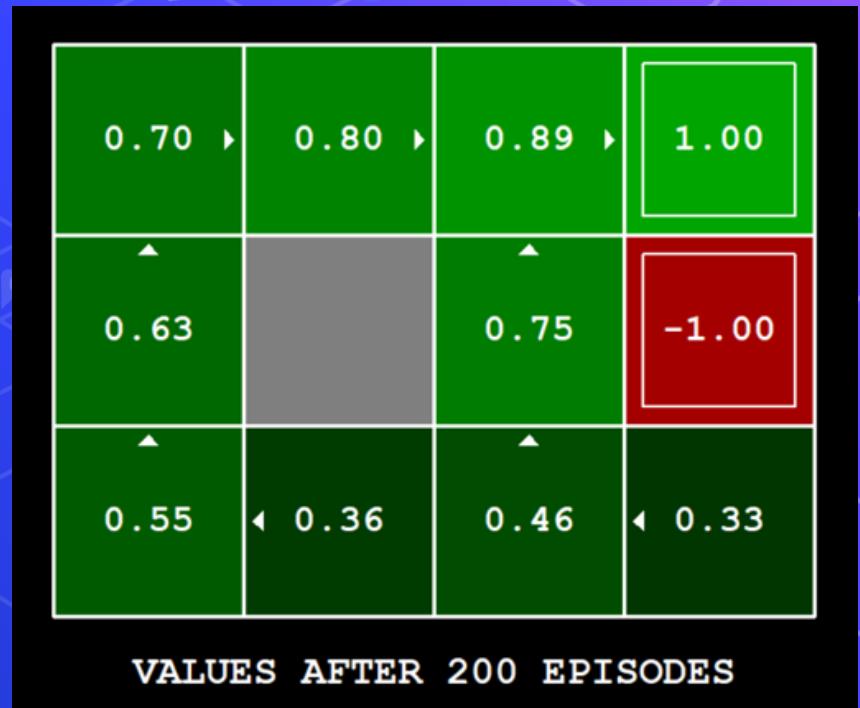
$$\begin{aligned} q_{\text{new}}([0, 1], \text{Right}) &= (1 - 0.7) * 0 + 0.7 (0 + 0.9 * 0.7) \\ &= 0.441 \end{aligned}$$

8. Repeat the steps for many episodes, and keep updating Q-values, after many episodes it will be like the image.



Q Learning Algorithm (Another Grid World)

So at the end we will have the optimal policy that can play the game with its own to have the maximum reward.



Q Learning Algorithm (Another Grid World)

To run the code.

```
1 python gridworld.py -k 400 -a q -s 10 -r -0.3
```

Reinforcement Learning

- What is Reinforcement Learning.
- Applications of Reinforcement Learning.
- Why RL is Freaky.
- Markov Decision Process (MDP).
- MDP Examples.
- MDP Solution.
- Q-Learning Algorithm.
- [Open AI Gym environment.](#)
- Coding time >_



Open AI Gym

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball.

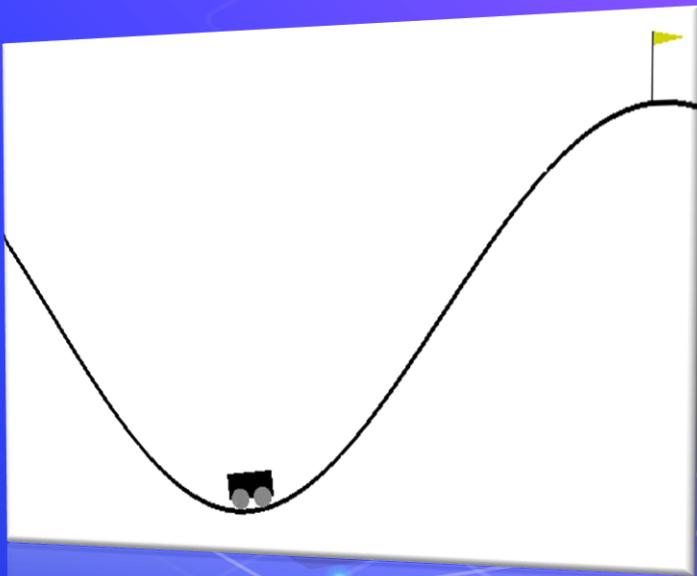
<https://gym.openai.com/>



OpenAI

Open AI Gym – Mountain Car

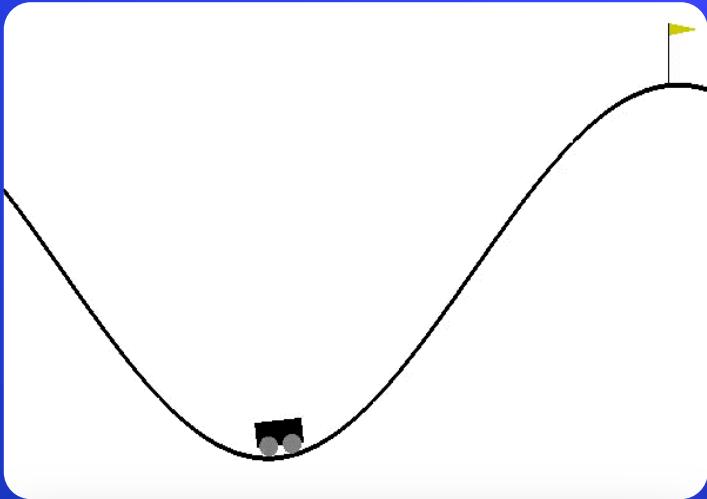
```
● ● ●  
1 import gym  
2  
3 env = gym.make('MountainCar-v0')  
4  
5 for i_episode in range(400):  
6     observation = env.reset() # get random state S0  
7     action = env.action_space.sample() # take random action  
8     observation, reward, done, info = env.step(action)  
9     env.render()  
10    if done:  
11        break  
12  
13 env.close()
```



Reinforcement Learning

- What is Reinforcement Learning.
- Applications of Reinforcement Learning.
- Why RL is Freaky.
- Markov Decision Process (MDP).
- MDP Examples.
- MDP Solution.
- Q-Learning Algorithm.
- Open AI Gym environment.
- Coding time >





```
1 import gym
2 import numpy as np
3
4 env = gym.make("MountainCar-v0")
5
6 lr = 0.5
7 DISCOUNT = 0.95
8 EPISODES = 80
9 START_EPS_DECAY = 1
10 END_EPS_DECAY = EPISODES//2
11 epsilon = 1
12 eps_decay = epsilon/(END_EPS_DECAY - START_EPS_DECAY)
13
14 q_table = np.zeros(([10,10] + [env.action_space.n]))
15
```

```
17 episode = 0
18 while episode < EPISODES:
19
20     done = False
21     discrete_state = calc_discrete_state(env.reset())
22
23     while not done:
24
25         # Exploit or explore
26         if np.random.random() > epsilon:
27             # Exploit from q-table
28             action = np.argmax(q_table[discrete_state])
29         else:
30             # Explore - t
31             action = np.random.randint(0, env.action_space.n)
32
33
34     new_state, reward, done, _ = env.step(action)
35
36     # Update q-table
37     max_future_q = np.max(q_table[new_state_disc])
38     current_q = q_table[discrete_state + (action,)]
39     reward_fun = reward + DISCOUNT * max_future_q
40     new_q = (1 - lr) * current_q + lr * reward_fun
41     q_table[discrete_state + (action,)] = new_q
42
43     discrete_state = calc_discrete_state(new_state_disc)
44
45     env.render()
46
47     if END_EPS_DECAY >= episode >= START_EPS_DECAY:
48         epsilon -= epsilon_change
49
50     episode += 1
```

Questions ?!



Thanks!

>_ Live long and prosper

