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

Agenda

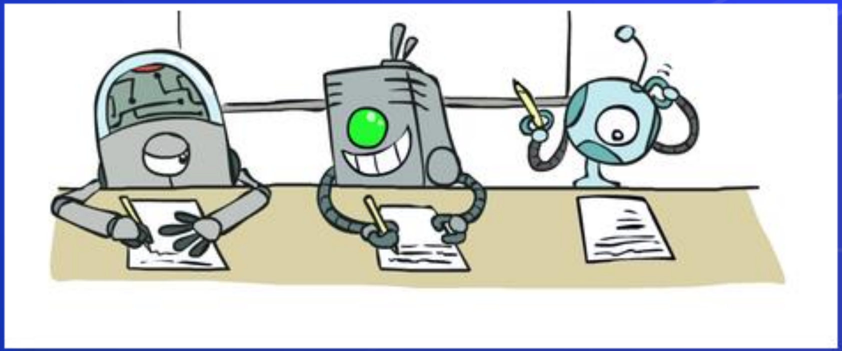
- What is Machine Learning
- Supervised Learning
  - Regression
  - Classification
- Unsupervised Learning
  - Clustering
  - Association rule mining
  - Dimension Reduction
- Model Selection & Evaluation
  - Cross Validation
  - Hyperparameter Tuning
- Reinforcement Learning

Agenda

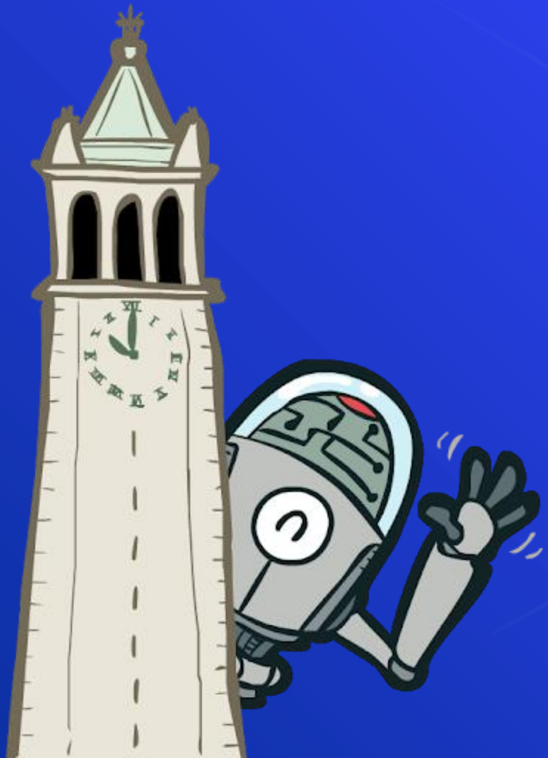
- **What is Machine Learning**
- Supervised Learning
  - Regression
  - Classification
- Unsupervised Learning
  - Clustering
  - Association rule mining
  - Dimension Reduction
- Model Selection & Evaluation
  - Cross Validation
  - Hyperparameter Tuning
- Reinforcement Learning

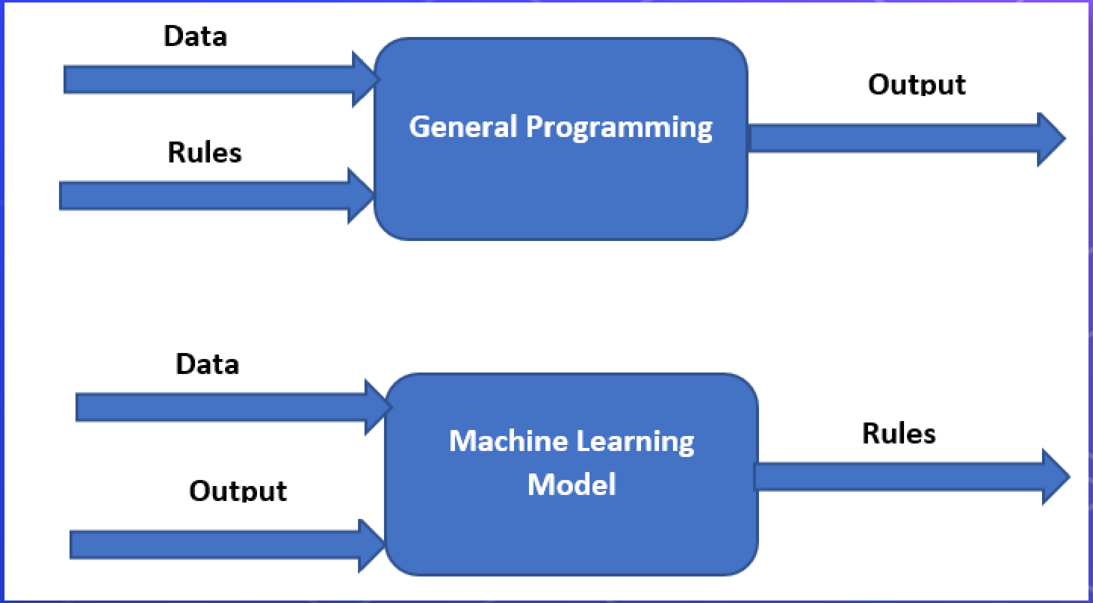
What is Machine Learning

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.

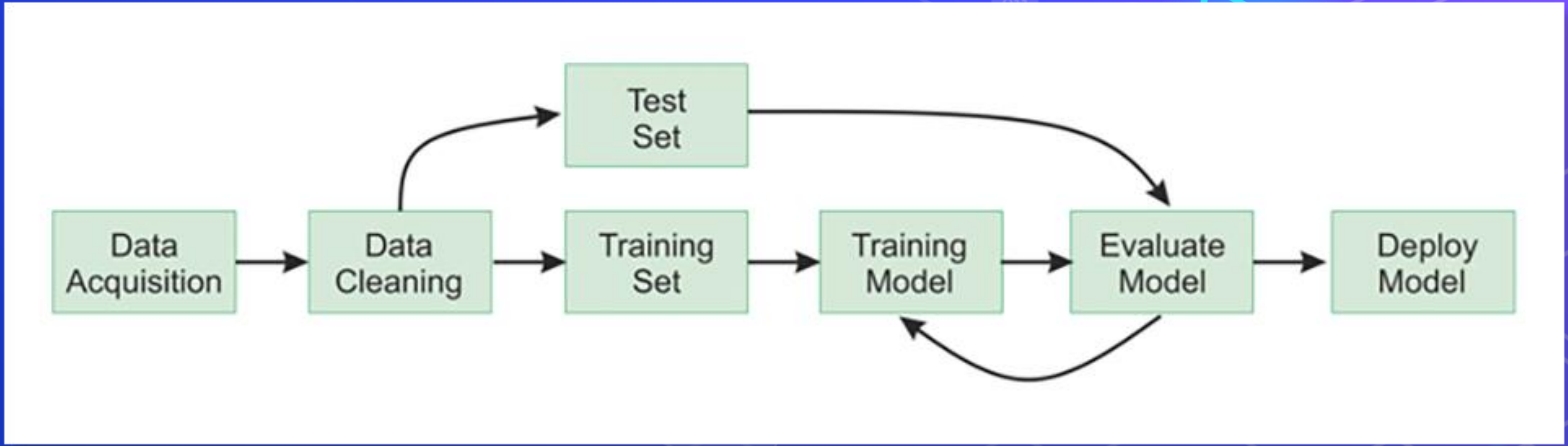


What is Machine Learning

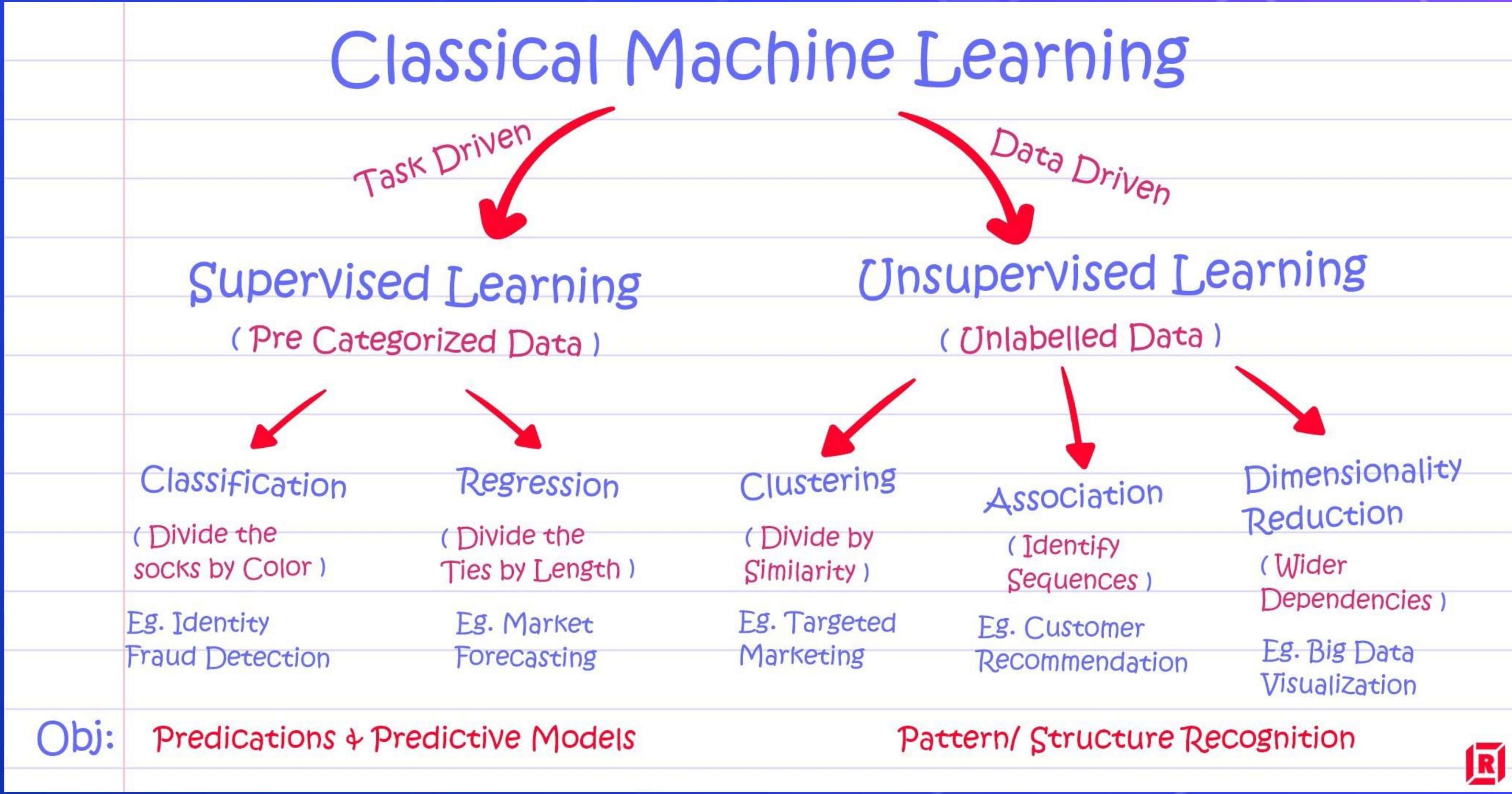




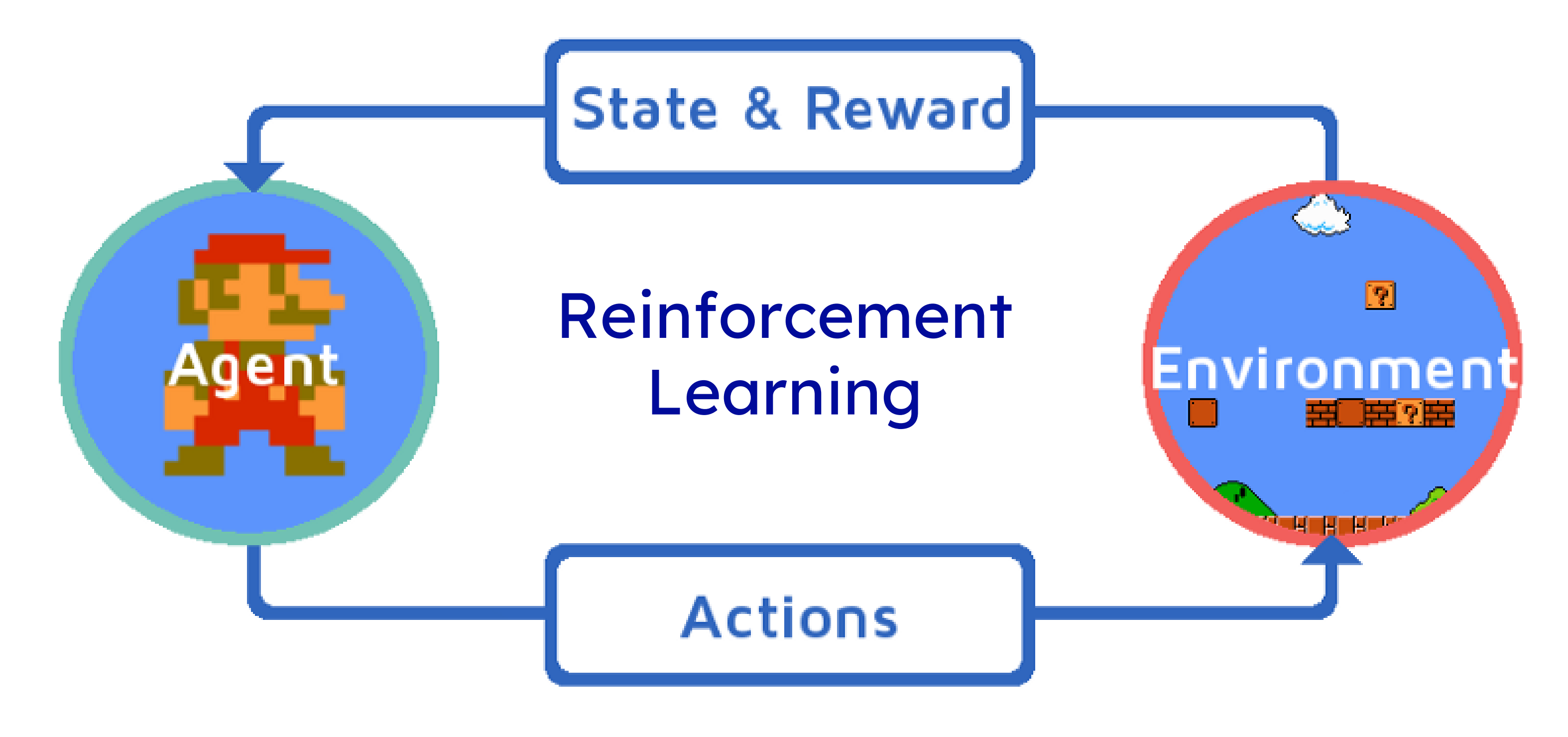
What is Machine Learning



What is Machine Learning



What is Machine Learning



What is 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.

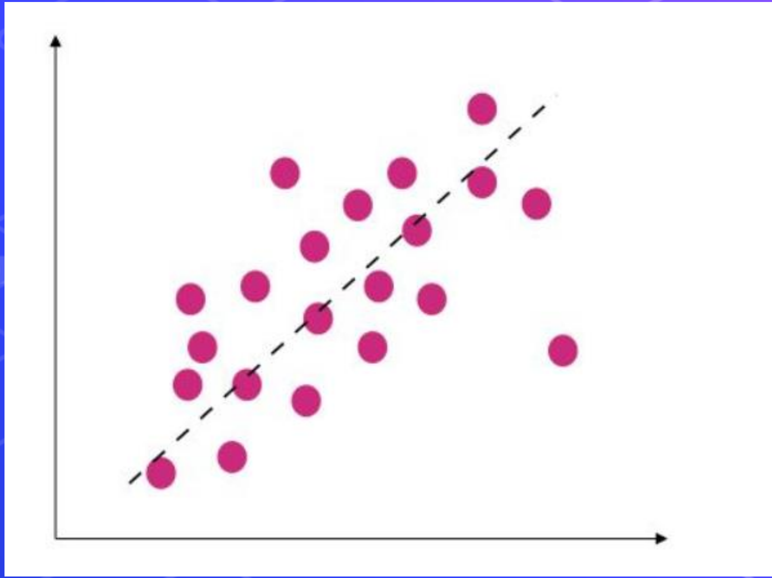
Agenda

- What is Machine Learning
- **Supervised Learning**
  - **Regression**
  - Classification
- Unsupervised Learning
  - Clustering
  - Association rule mining
  - Dimension Reduction
- Model Selection & Evaluation
  - Cross Validation
  - Hyperparameter Tuning
- Reinforcement Learning

Regression

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

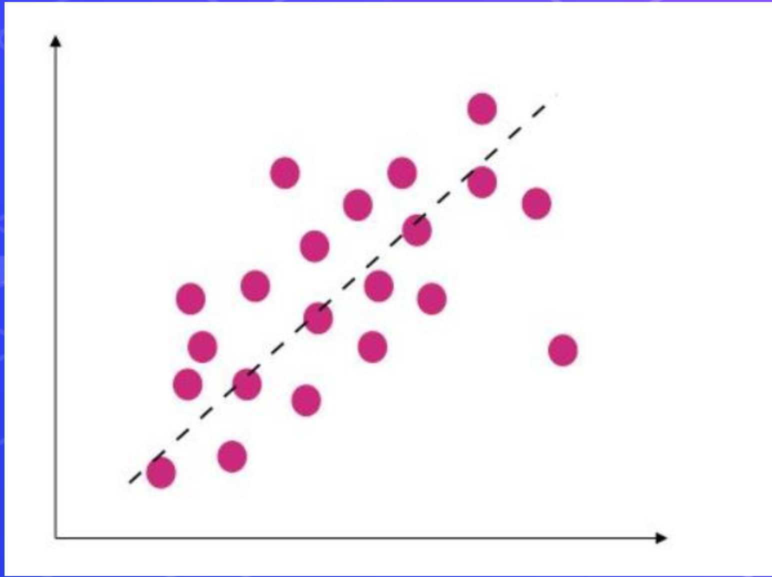
For example,
it can be used to quantify the relative impacts of age, gender, and diet (the predictors) on weight (the output or dependent).



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

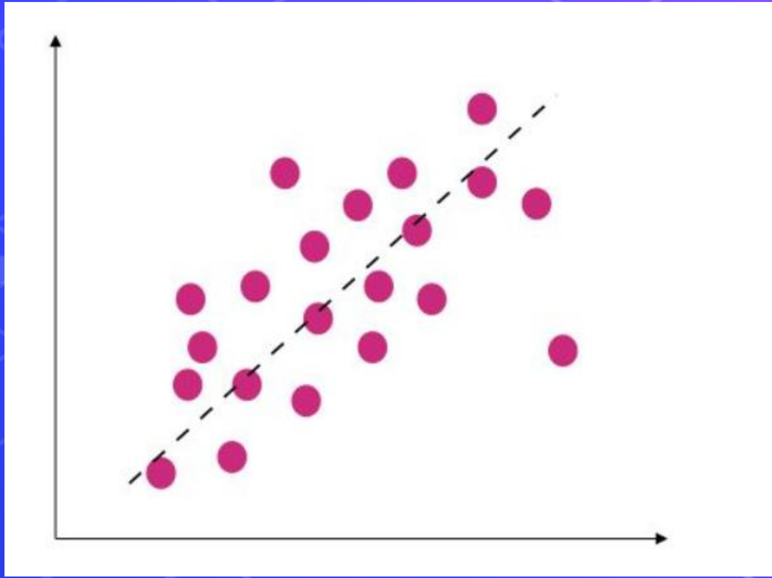
Regression

- Simple & Multiple Linear Regression
- Polynomial Regression
- Ridge, Lasso, ElasticNet Regression
- SVM
- Decision Trees & Random Forests
- XGBoost
- Evaluating Model Performance



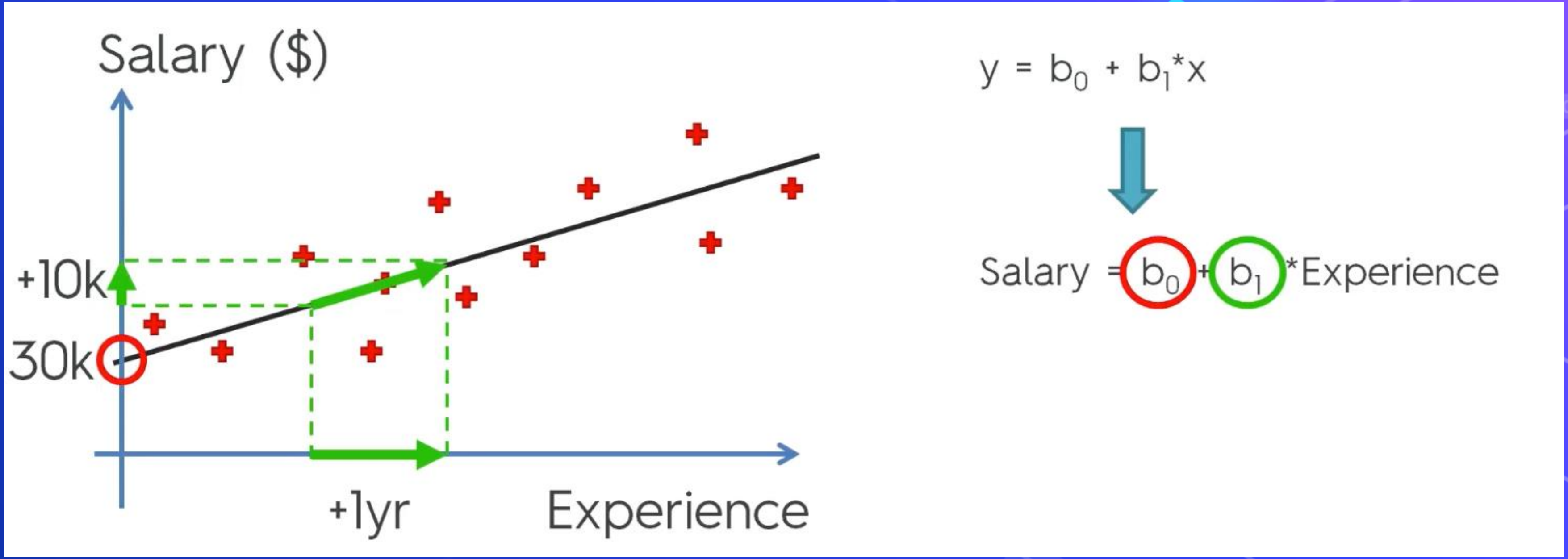
Regression

- Simple & Multiple Linear Regression
- Polynomial Regression
- Ridge, Lasso, ElasticNet Regression
- SVM
- Decision Trees & Random Forests
- XGBoost
- Evaluating Model Performance

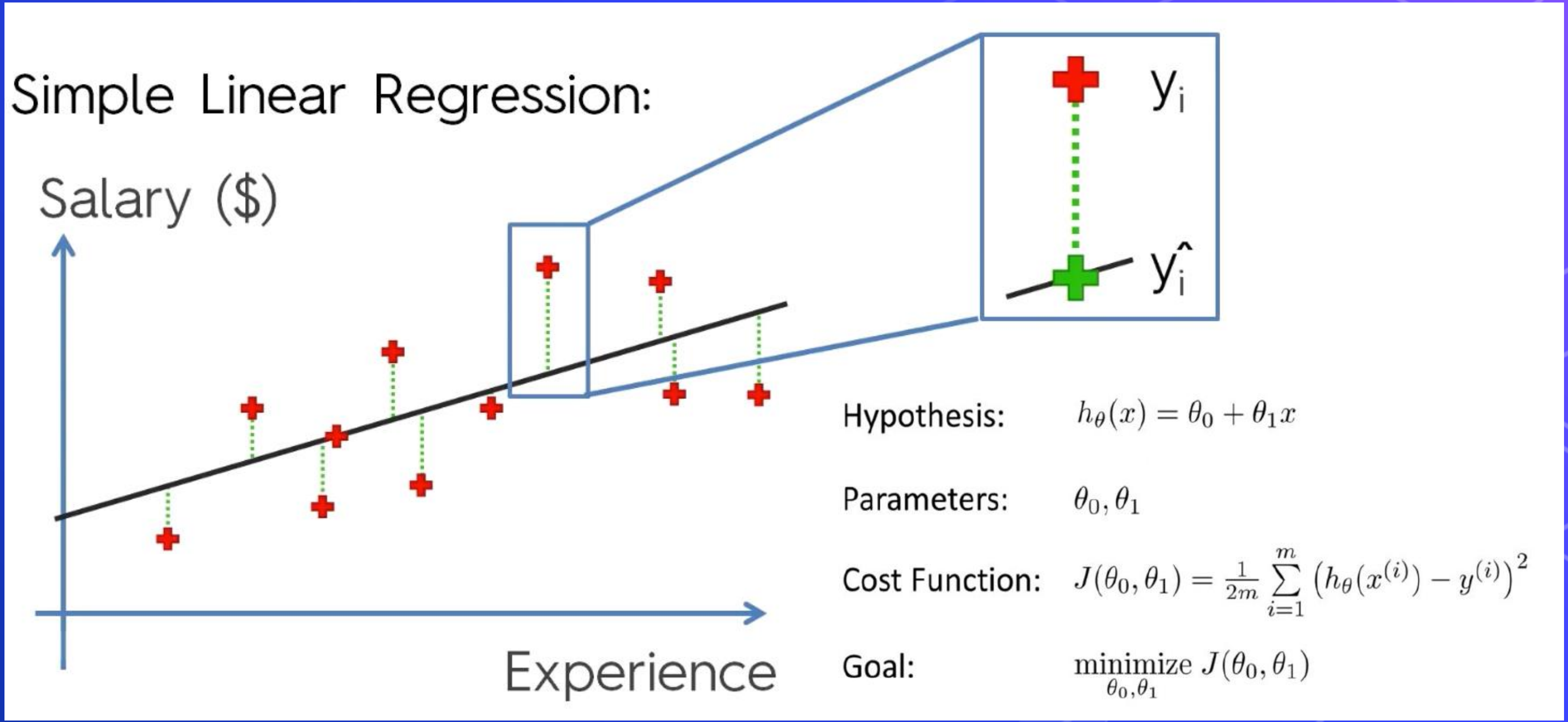


Simple Linear Regression (Equation of a straight line)

It also called ordinary least squares (OLS)



Simple Linear Regression (Calculate Cost Function)



Simple Linear Regression (Calculate Cost Function)

The goal is to minimize the sum of squares of residuals (y-y’) is as small as possible.

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

Parameters: θ_0, θ_1

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

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

Simple Linear Regression (Example)

Theta0 = 5 , theta 1 = 2 Equation $h(x) = 5 + 2x$

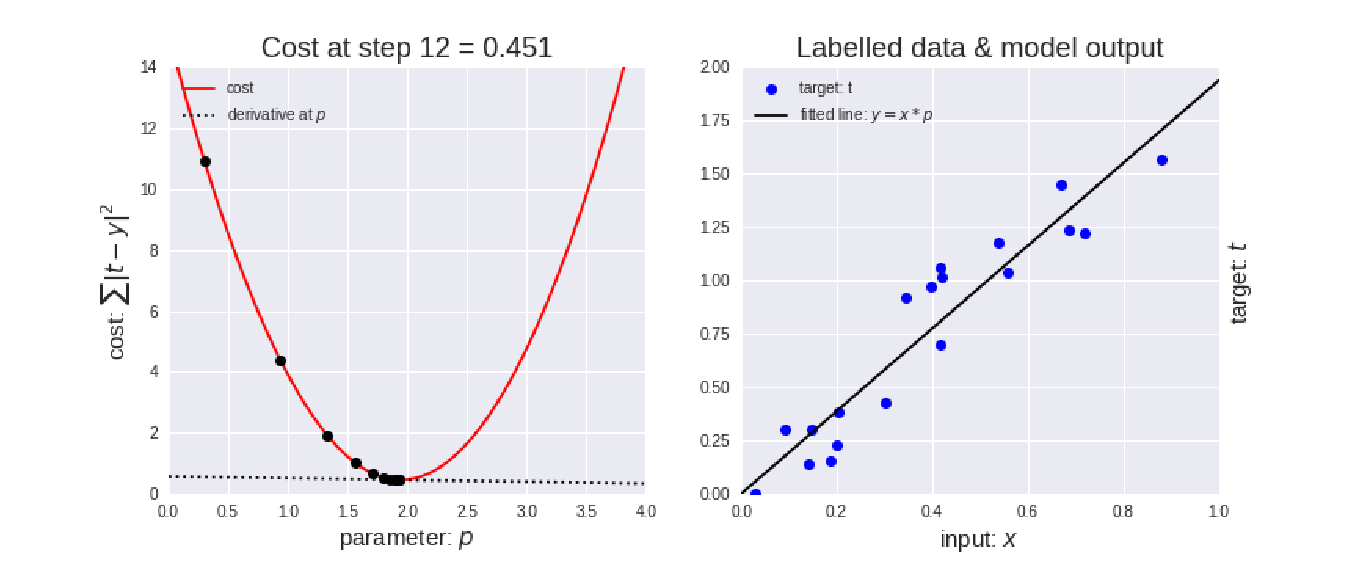
X	Y	h(x)	h(x) - y	$(h(x) - y)^2$
1	7	7	0	0
2	8	9	1	1
2	7	9	2	4
3	9	11	2	4
4	11	13	2	4
5	10	15	5	25
5	12	15	3	9

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

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

$$J\left(\Theta_0, \Theta_1\right) = \frac{1}{2m} \sum_{i=1}^{m} [\underbrace{h_\Theta(x_i)}_{\text{Predicted Value}} - \underbrace{y_i}_{\text{True Value}}]^2$$

Gradient Descent

$$\boxed{\Theta_j = \Theta_j - \underbrace{\alpha}_{\text{Learning Rate}} \frac{\partial}{\partial \Theta_j} J\left(\Theta_0, \Theta_1\right)}$$

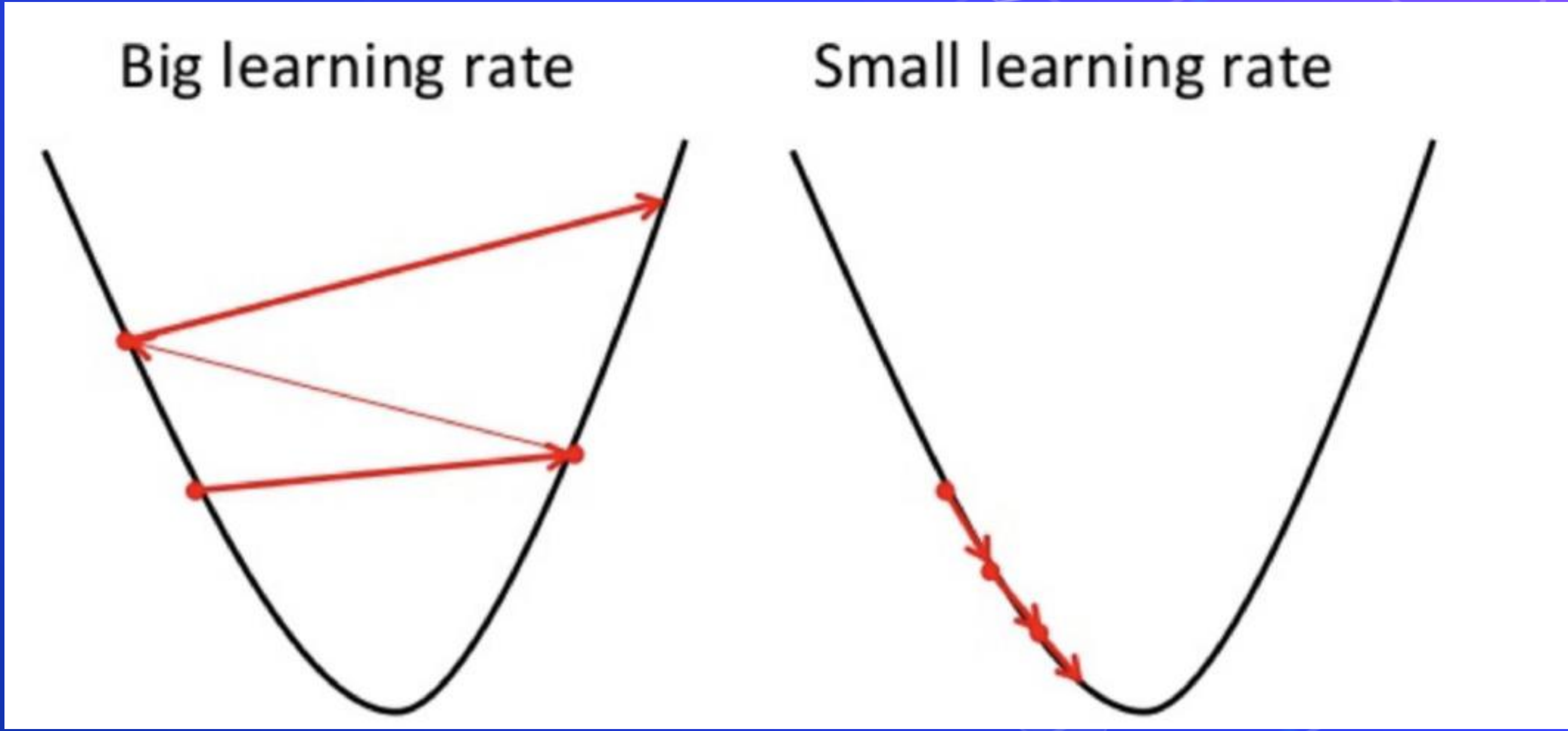
Now,

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

Therefore,

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

Simple Linear Regression (Alpha constant)



Multiple Linear Regression (Equation)

Simple Linear Regression

$$y = b_0 + b_1{}^*x_1$$

Multiple Linear Regression

Dependent variable (DV)

Independent variables (IVs)

$$y = b_0 + b_1{}^*x_1 + b_2{}^*x_2 + \ldots + b_n{}^*x_n$$

Constant

Coefficients

Multiple Linear Regression (Update theta values)

repeat until convergence: {

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

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

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

...

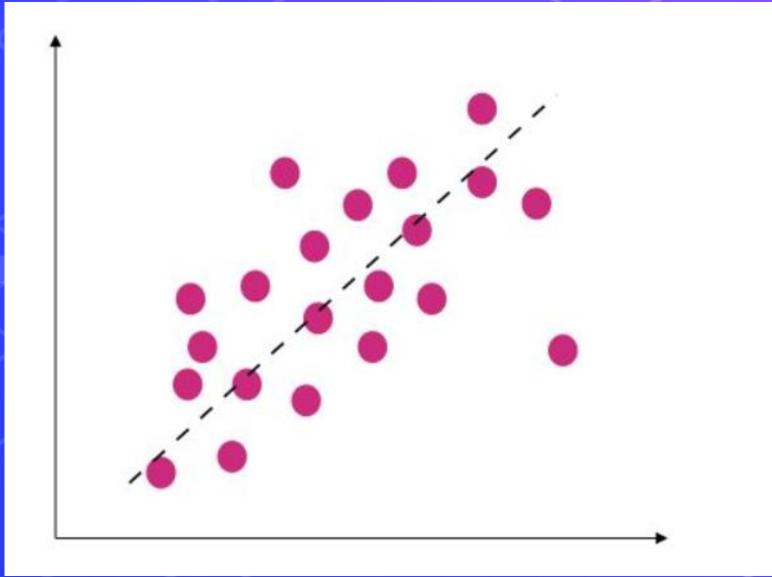
}

Linear Regression (Code)

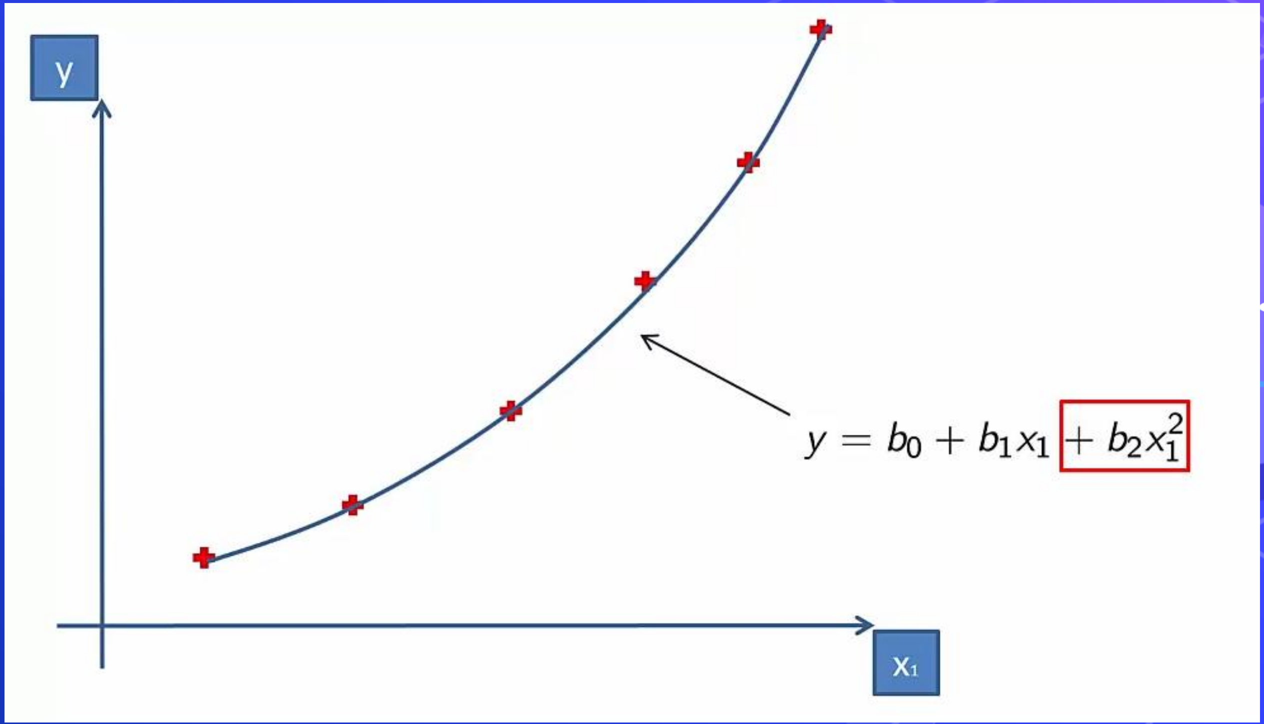
```python
from sklearn.linear_model import LinearRegression

# make model object
model = LinearRegression()

# train
model.fit(x_train, y_train)

# test
model.predict(x_test)

# calculate R2 score on training data
model.score(x_train, y_train)

# calculate R2 score on testing data
model.score(x_test, y_test)

# get model parameters
model.coef_
model.intercept_
```

Regression

- Simple & Multiple Linear Regression
- <u>Polynomial Regression</u>
- Ridge, Lasso, ElasticNet Regression
- SVM
- Decision Trees & Random Forests
- XGBoost
- Evaluating Model Performance



Polynomial Regression (Poly equation)



Polynomial Regression (Poly equation)

Polynomial Linear Regression

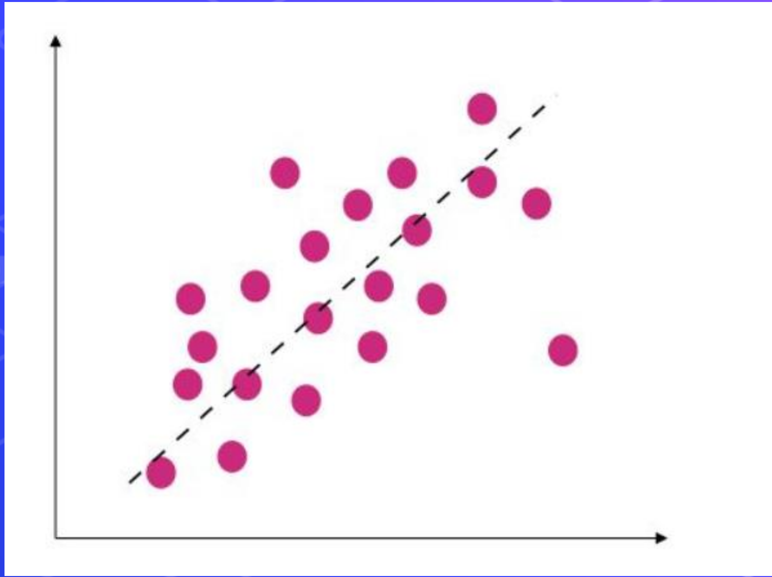
$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_n x_1^n$$

Polynomial Regression (Code)

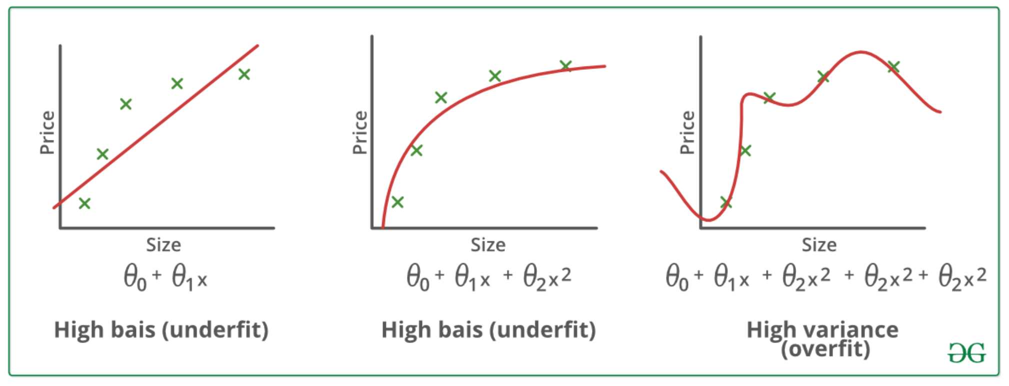
```python
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# create poly features for a feature
poly = PolynomialFeatures(degree=2)
x_train_poly = poly.fit_transform(x_train)
x_test_poly = poly.fit_transform(x_test)

# make model object
model = LinearRegression()

# train
model.fit(x_train_poly, y_train)

# test
model.predict(x_test_poly)

# calculate R2 score on training data
model.score(x_train_poly, y_train)

# calculate R2 score on testing data
model.score(x_train_poly, y_train)

# get model parameters
model.coef_
model.intercept_
```

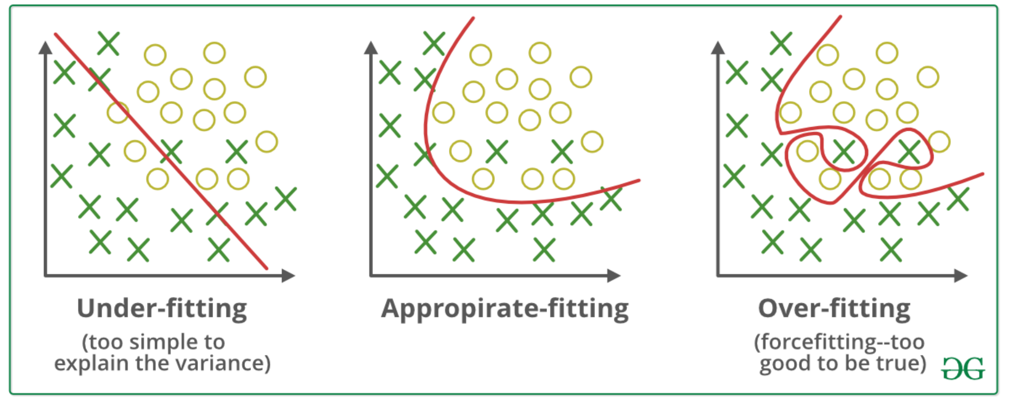
Regression

- Simple & Multiple Linear Regression
- Polynomial Regression
- <u>Ridge, Lasso, ElasticNet Regression</u>
- SVM
- Decision Trees & Random Forests
- XGBoost
- Evaluating Model Performance



Ridge, Lasso, ElasticNet Regression





Ridge, Lasso, ElasticNet Regression

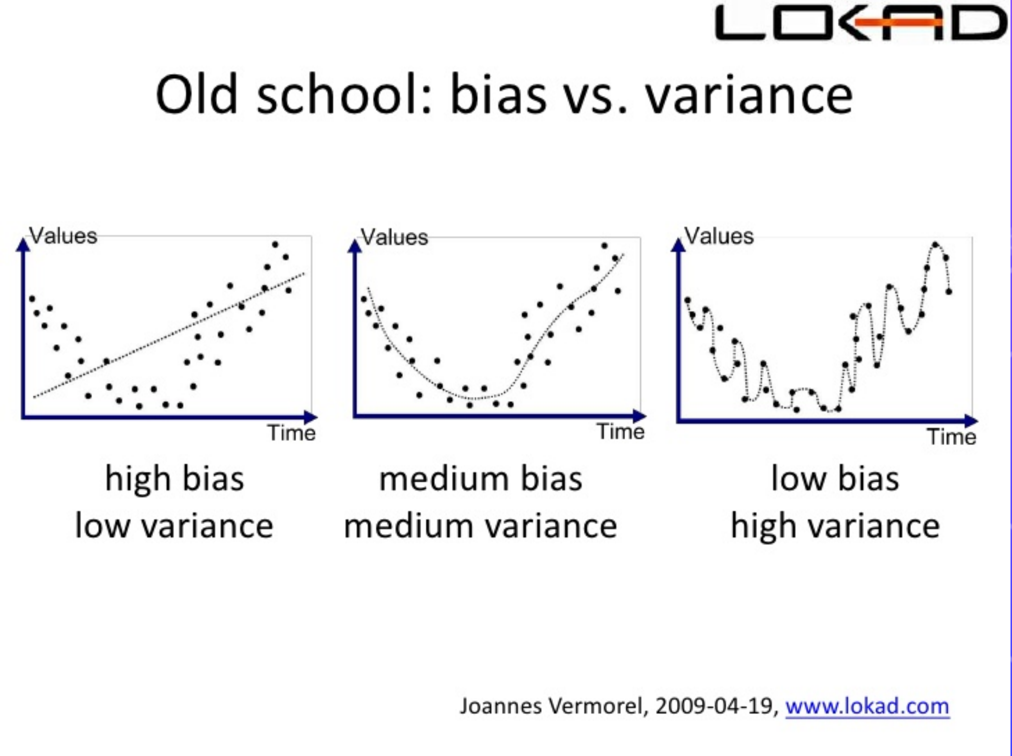
The more features you add the accuracy improves but this can lead to **Overfitting** that means the model memorizes the data too much and not applying **Generalization**.

So we should know about the trade-off between **Bias** and **Variance**.

- A model with a high bias error **underfits** data and makes very simplistic assumptions on it.

- A model with a high variance error **overfits** the data and learns too much from it.

- A good model is where both Bias and Variance errors are balanced

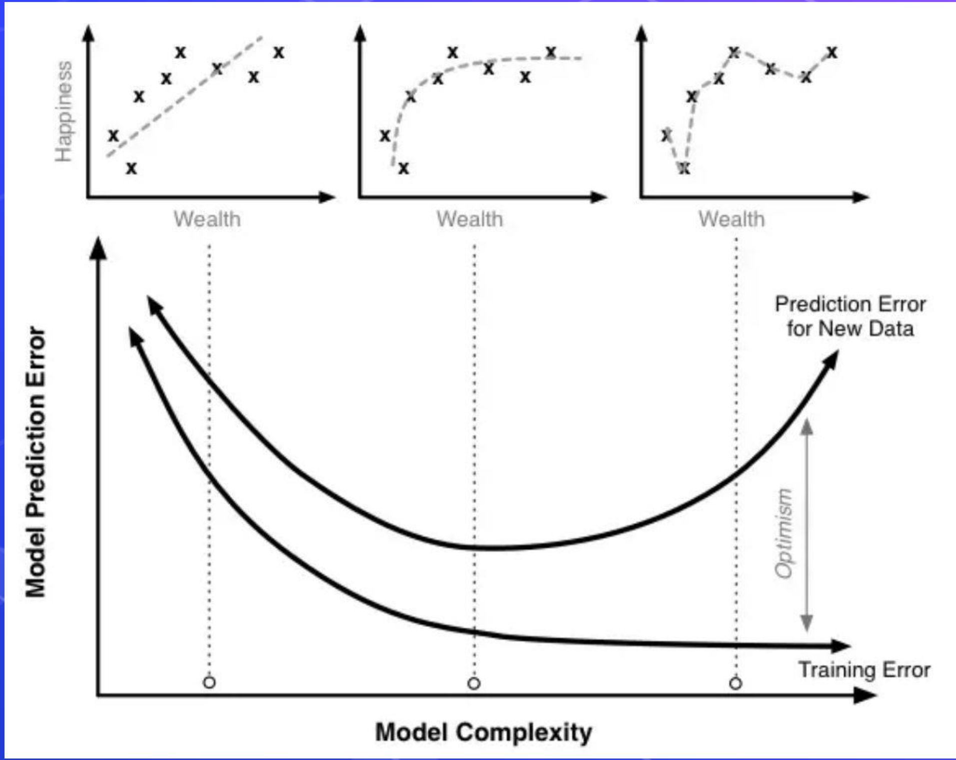


Ridge, Lasso, ElasticNet Regression

Underfitting (High Bias, Low Variance)
Model is so simple and weak to understand and represent data.
High Train Error + High Test Error

Overfitting (Low Bias, High Variance)
Model is so complex so it memorizes training data and can't apply generalization.
Low Train Error + High Test Error

Optimal (Low Bias, Low Variance)
Model is not simple nor complex and apply generalization well.
Low Train Error + Low Test Error



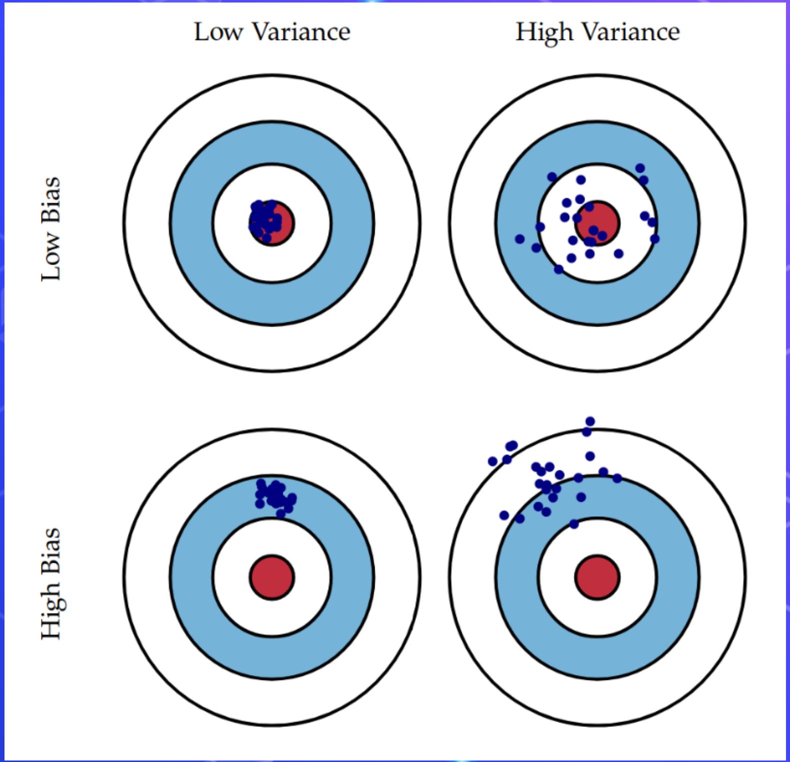
Ridge, Lasso, ElasticNet Regression

Techniques to reduce underfitting

- Increase model complexity.
- Increase number of features.
- Performing feature engineering.
- Remove noise from the data.

Techniques to reduce overfitting

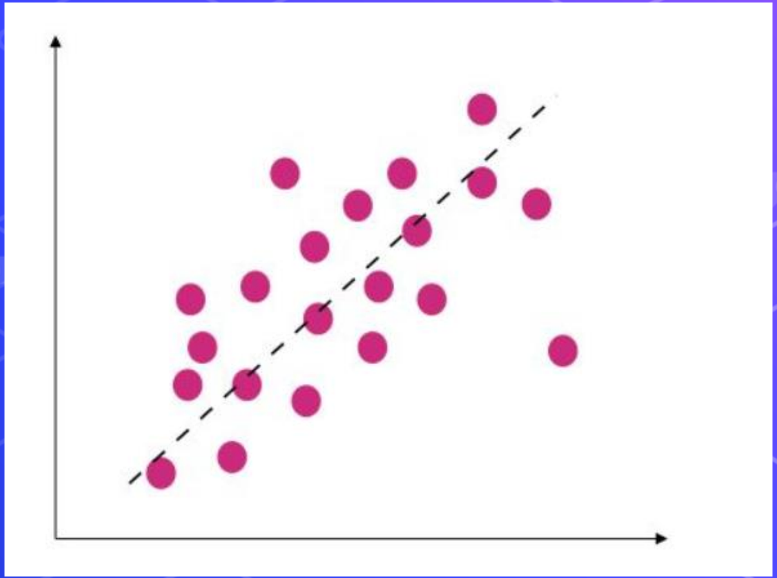
- Increase training data.
- Remove correlated features.
- Use simpler model.
- Regularization (add penalty bias).



And in this topic we will talk about **Regularization**.

Regression

- Simple & Multiple Linear Regression
- Polynomial Regression
- Ridge, Lasso, ElasticNet Regression
- SVM
- Decision Trees & Random Forests
- XGBoost
- Evaluating Model Performance



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	

$$R^2 = SSR/SST$$

$$= 1844.12/3028.77$$

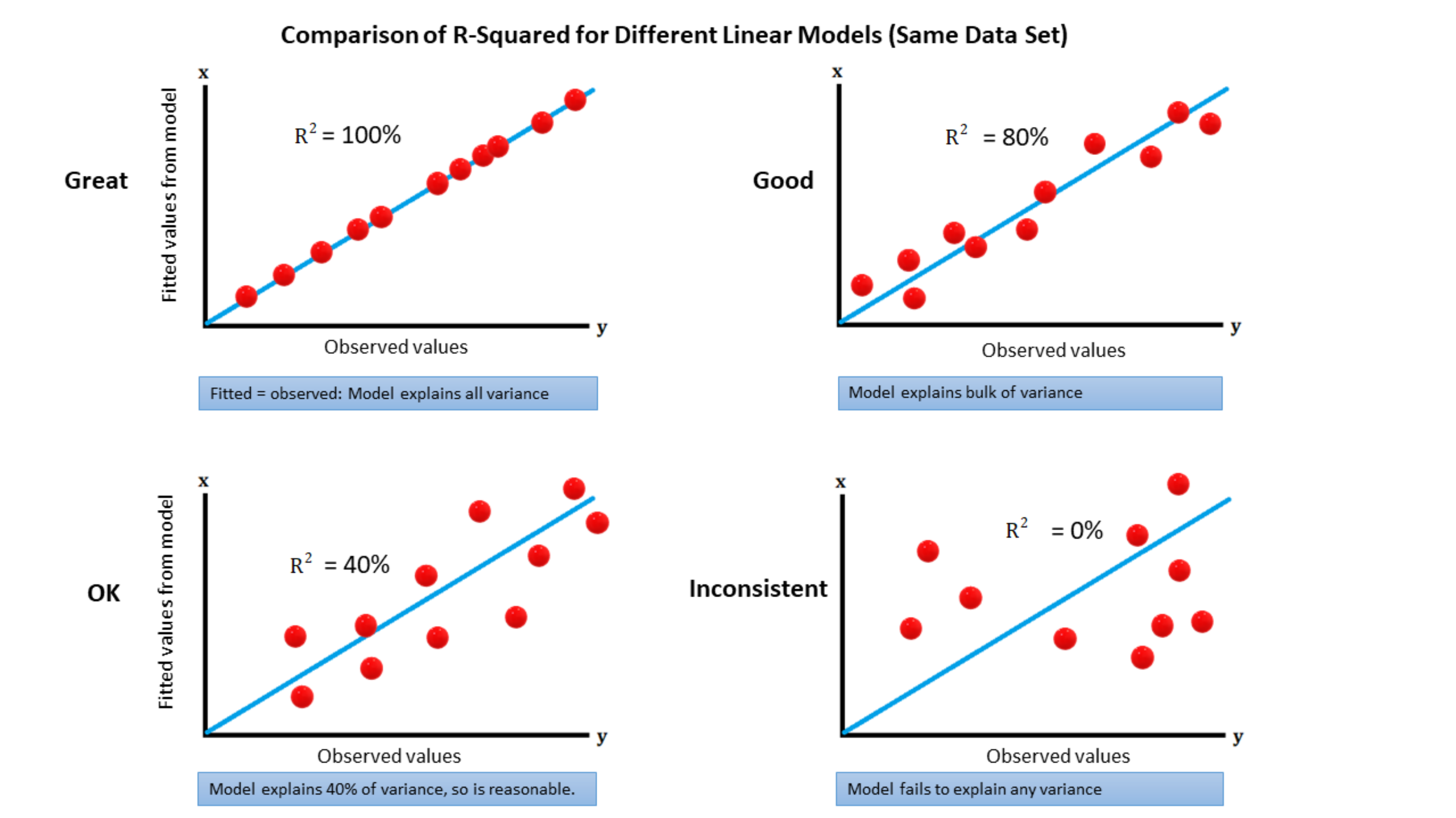
$$= 0.60886$$

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

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

$(Y - \bar{Y})^2$	$(\hat{Y} - \bar{Y})^2$
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 (R2)



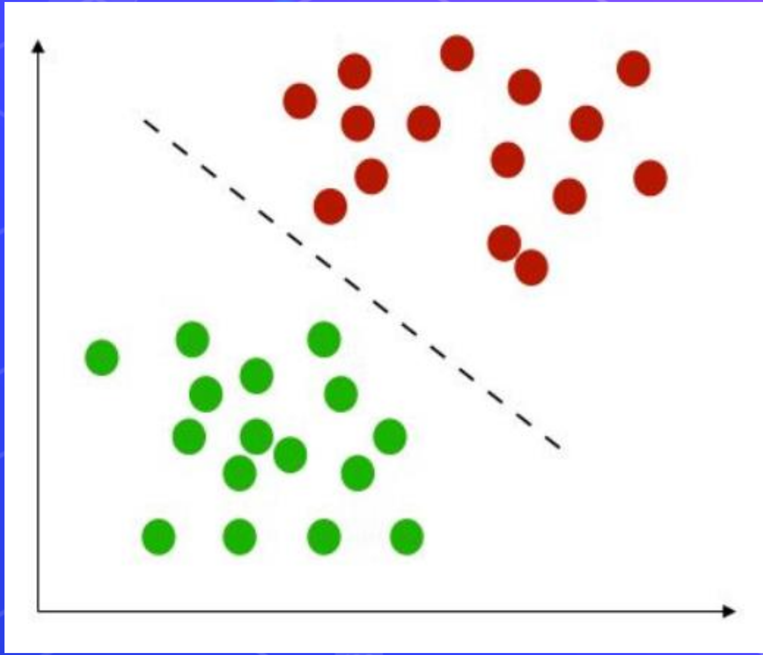
Agenda

- What is Machine Learning
- **Supervised Learning**
  - Regression
  - **Classification**
- Unsupervised Learning
  - Clustering
  - Association rule mining
  - Dimension Reduction
- Model Selection & Evaluation
  - Cross Validation
  - Hyperparameter Tuning
- 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.

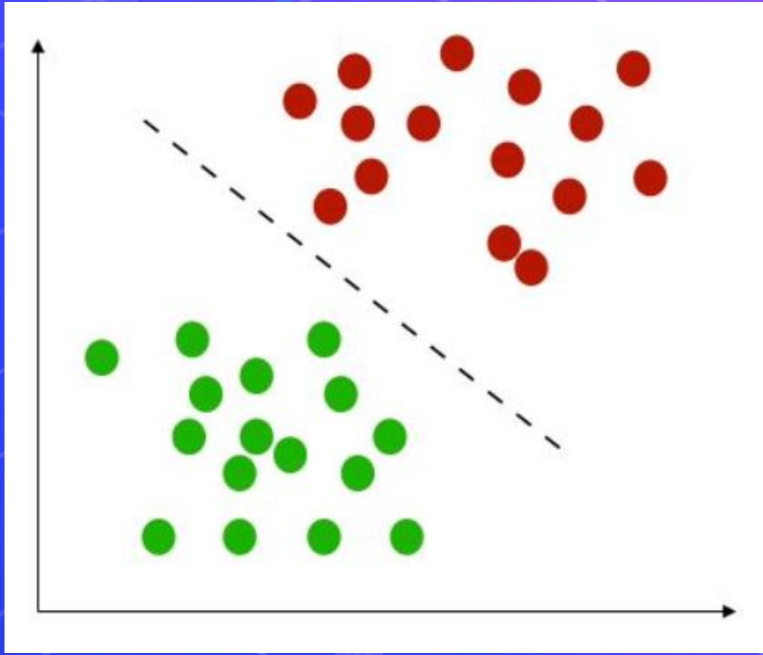
For example,
it can be used to classify the animal in an image, it's either a cat or a dog.



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

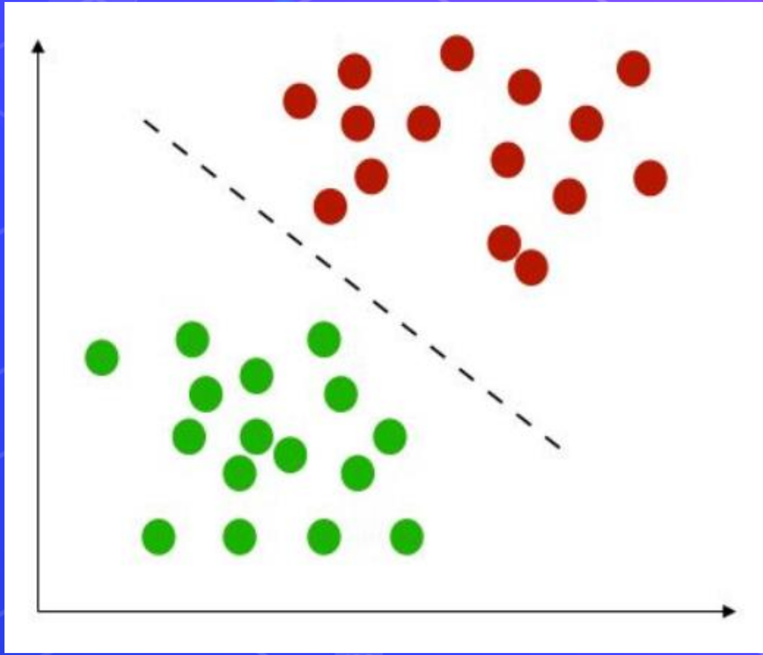
Classification

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



Logistic Regression

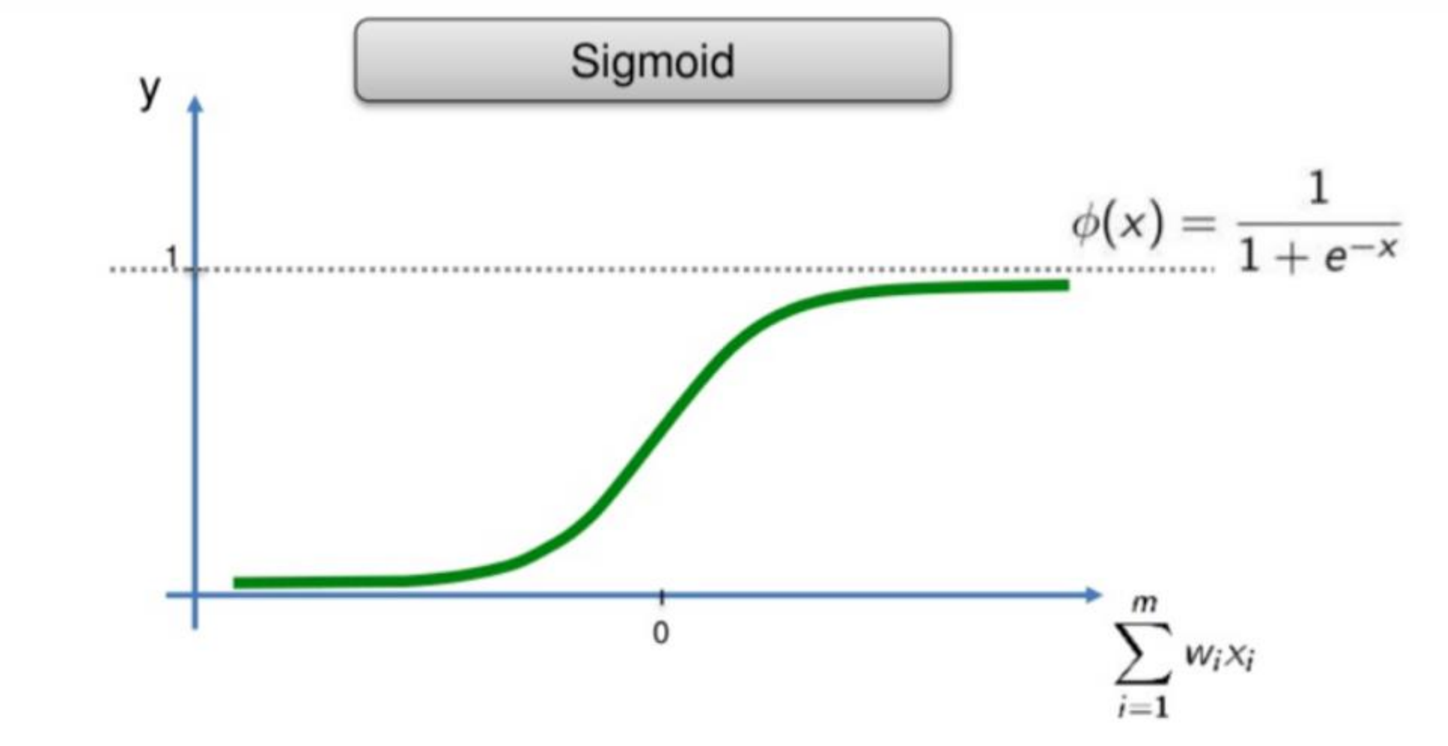
The Logistic Function

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

where

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

Logistic Regression

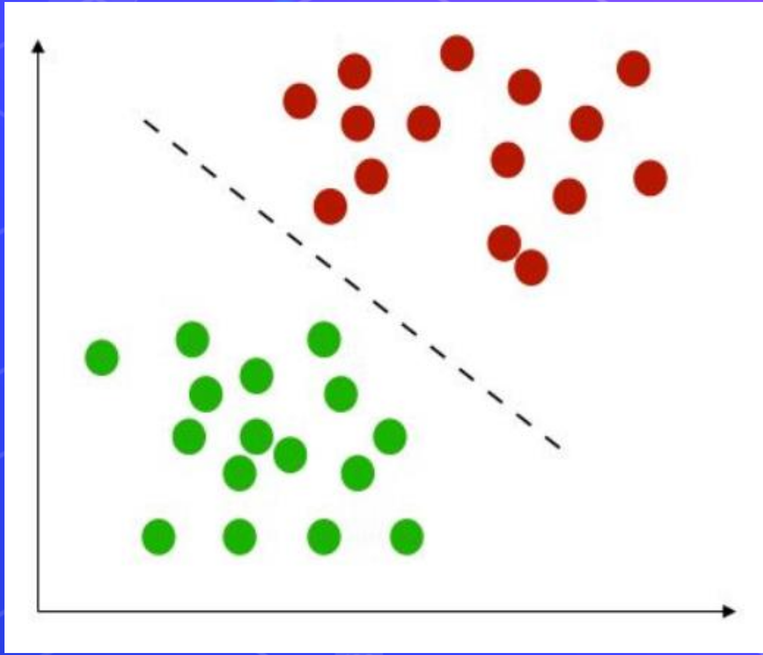


Logistic Regression

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix


model = LogisticRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

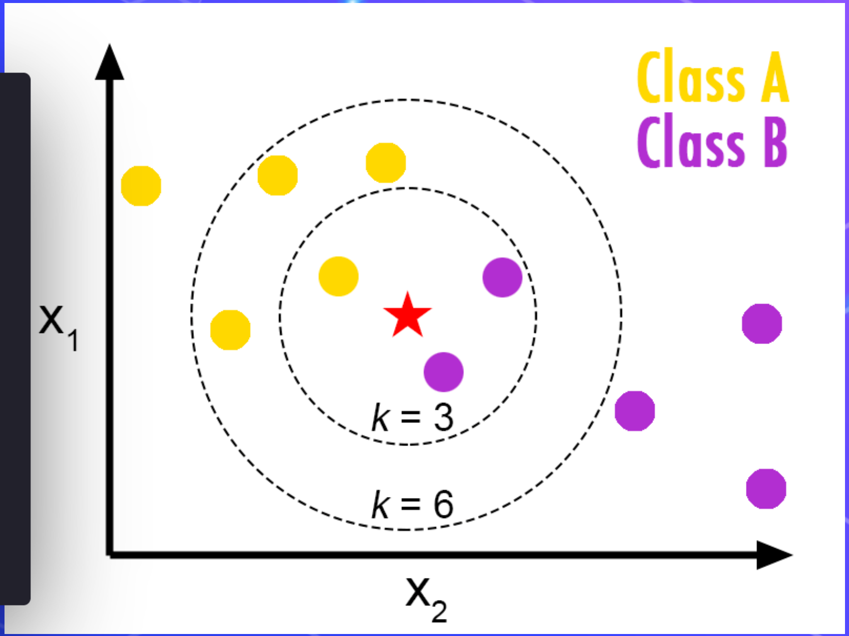
Classification

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



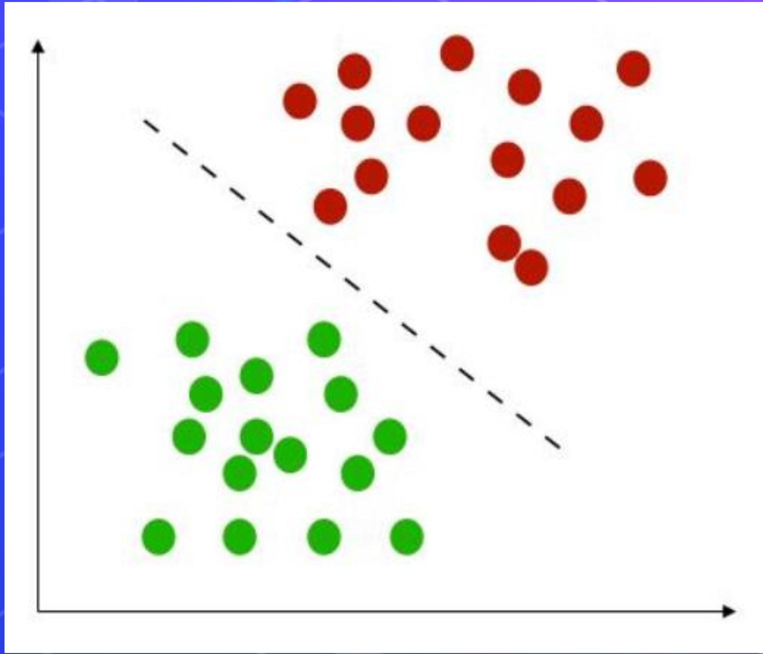
K-Nearest Neighbors (KNN)

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix


model = KNeighborsClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

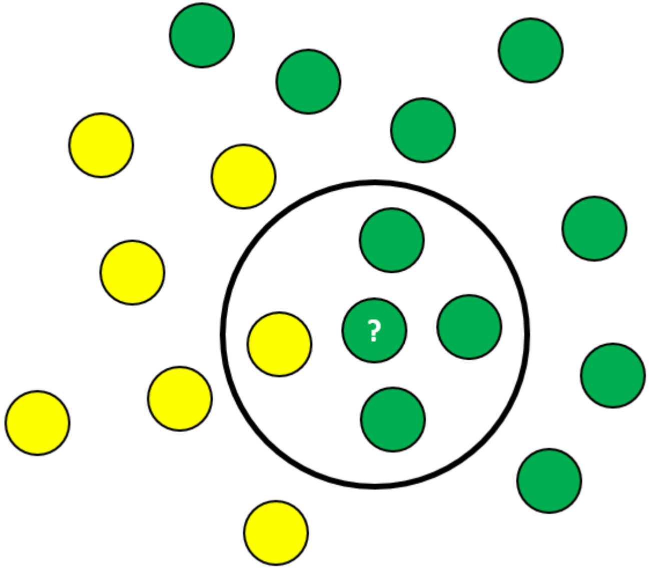


Classification

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



Naive Bayes



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

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

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

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

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

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

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

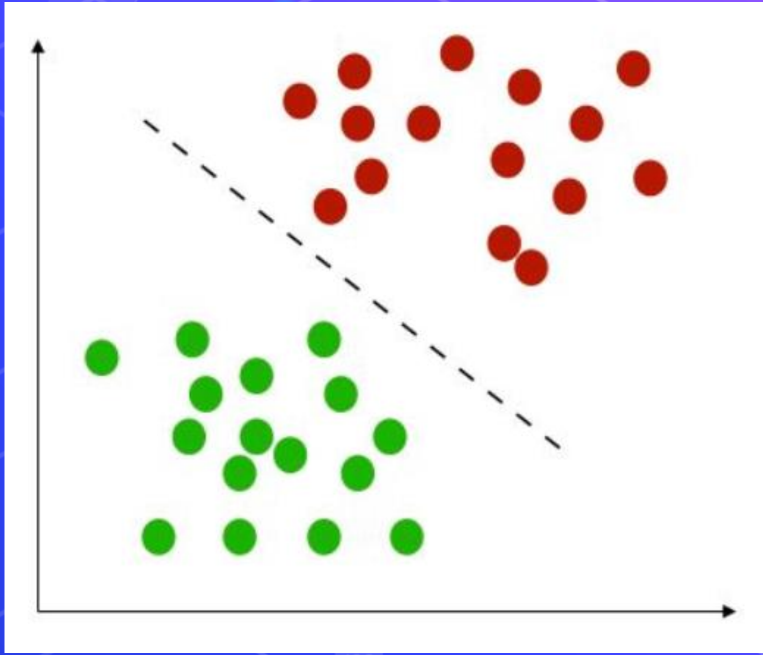
posterior probability $P''(\text{? is yellow}) = P(yellow) * P'(? \mid yellow) = \frac{7}{17} * \frac{1}{7} = \frac{7}{119}$

Naive Bayes

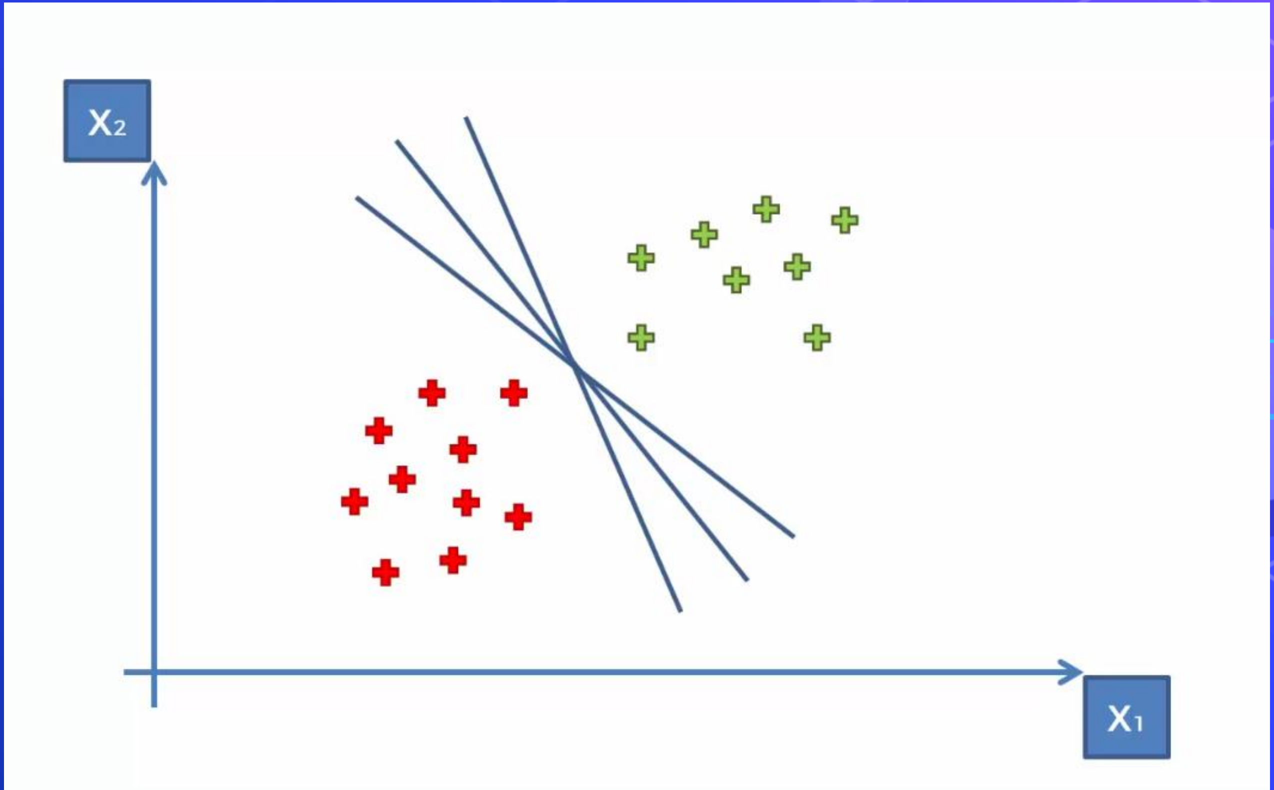
```python
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix


model = MultinomialNB()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Classification

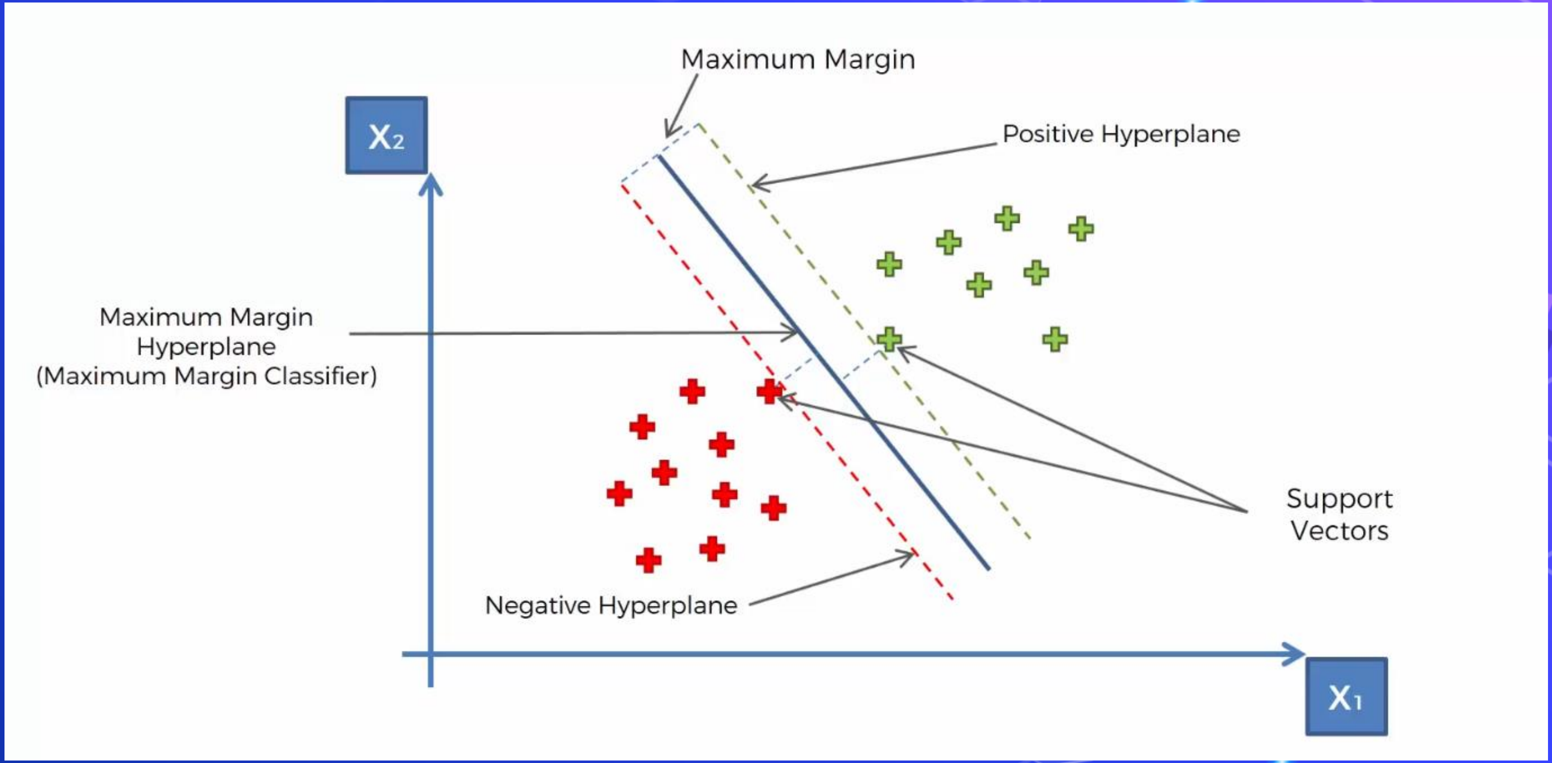
- Logistic Regression
- K-Nearest Neighbors (KNN)
- Naive Bayes
- <u>SVM</u>
- Decision Trees
- Ensemble Methods
  - What is Bagging & Boosting
  - Random Forests
  - XGBoost
- Evaluating Model Performance



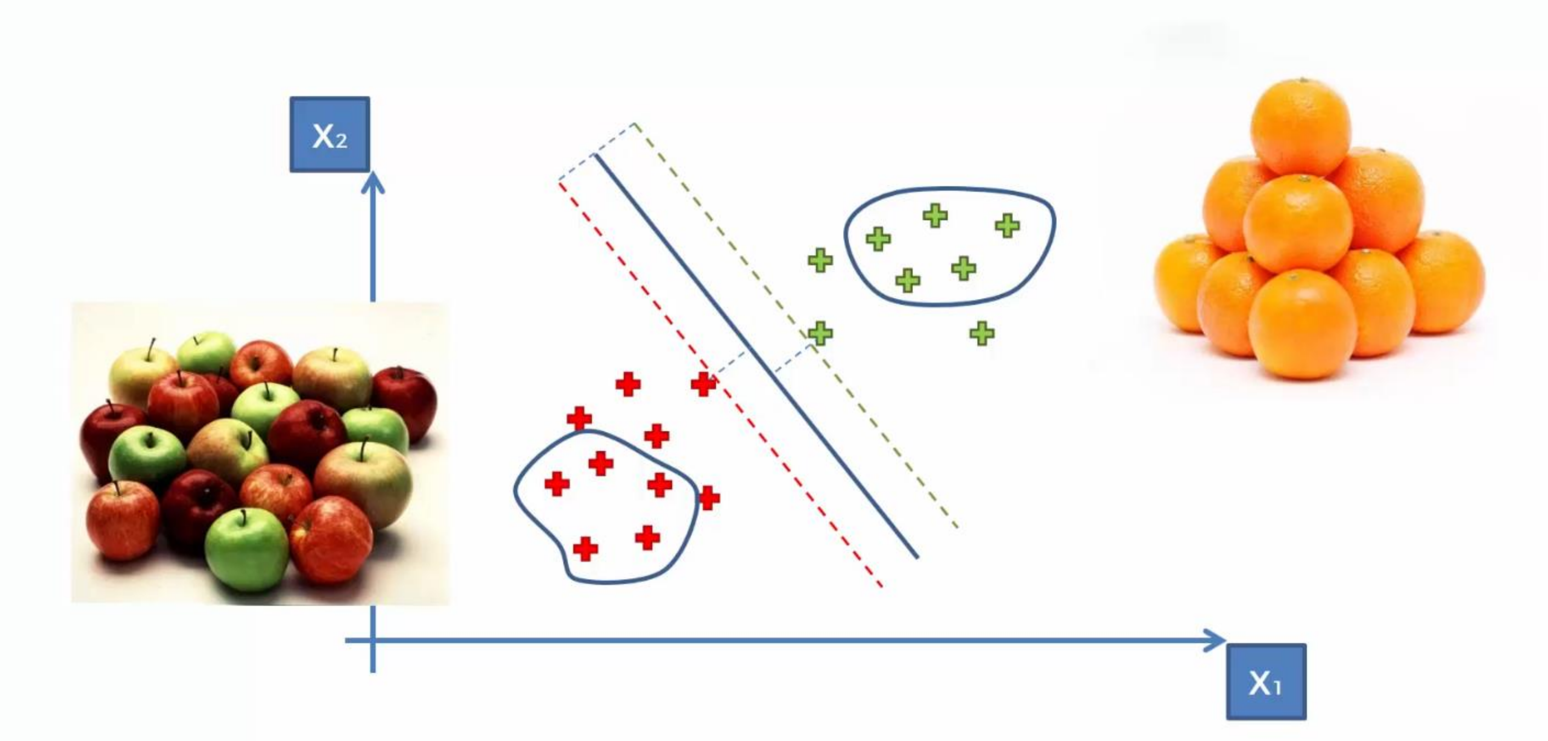
SVM (Support Vector Machine)



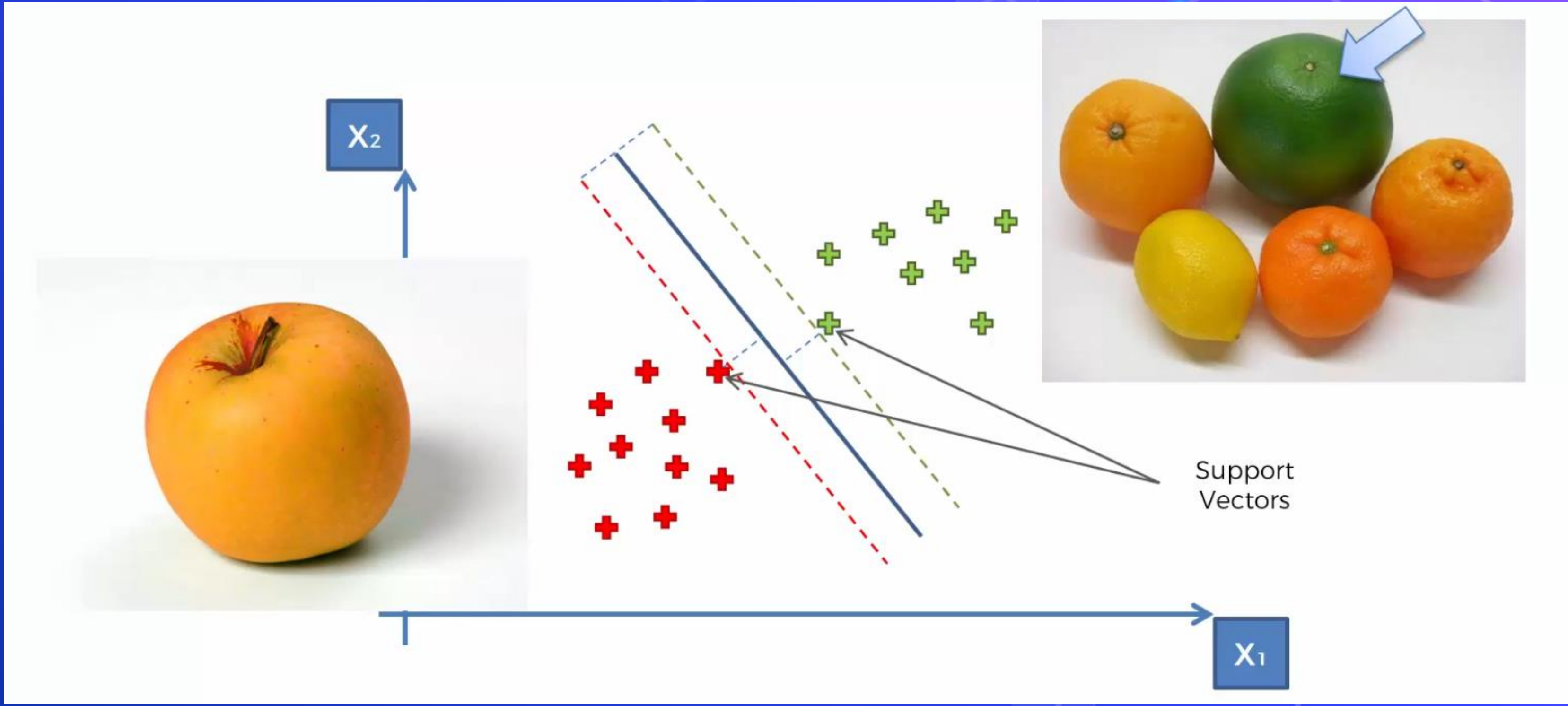
SVM (Support Vector Machine)



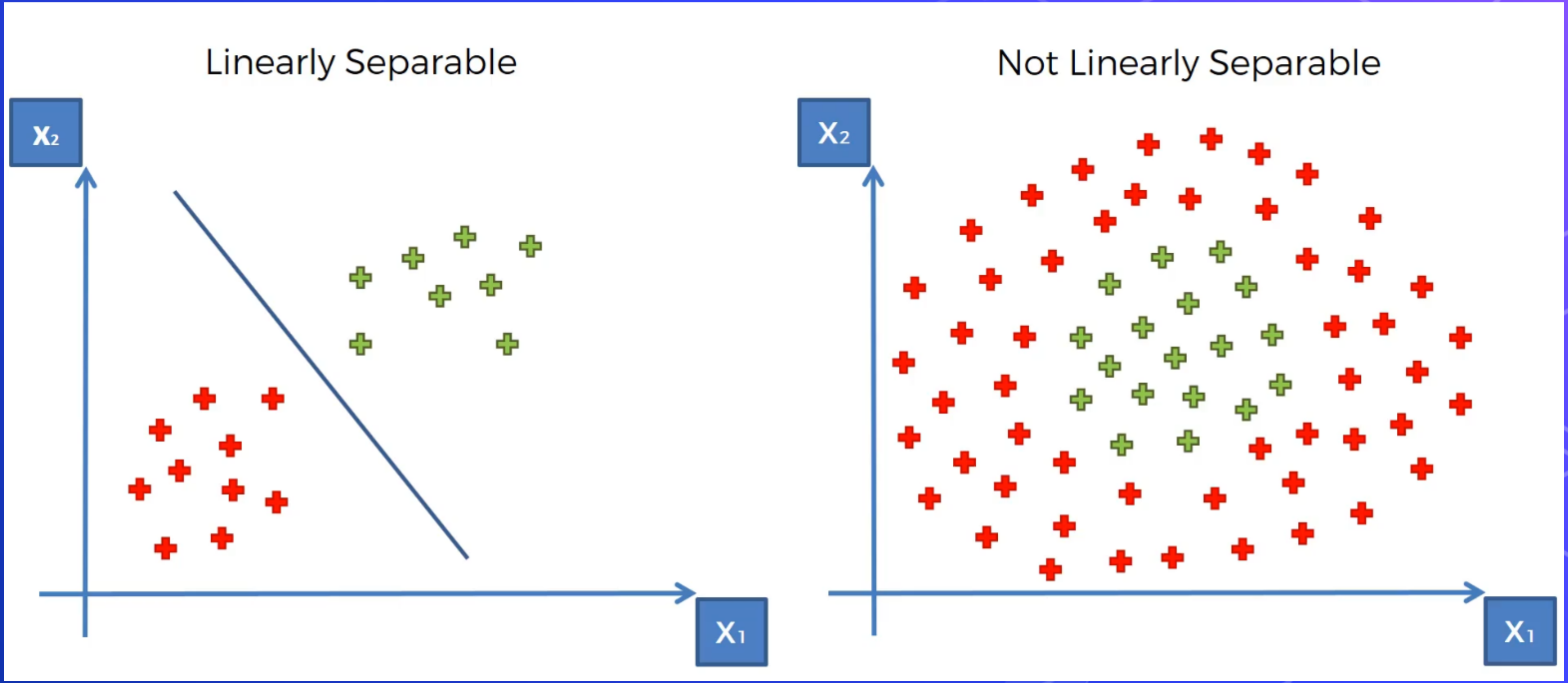
SVM (Support Vector Machine)



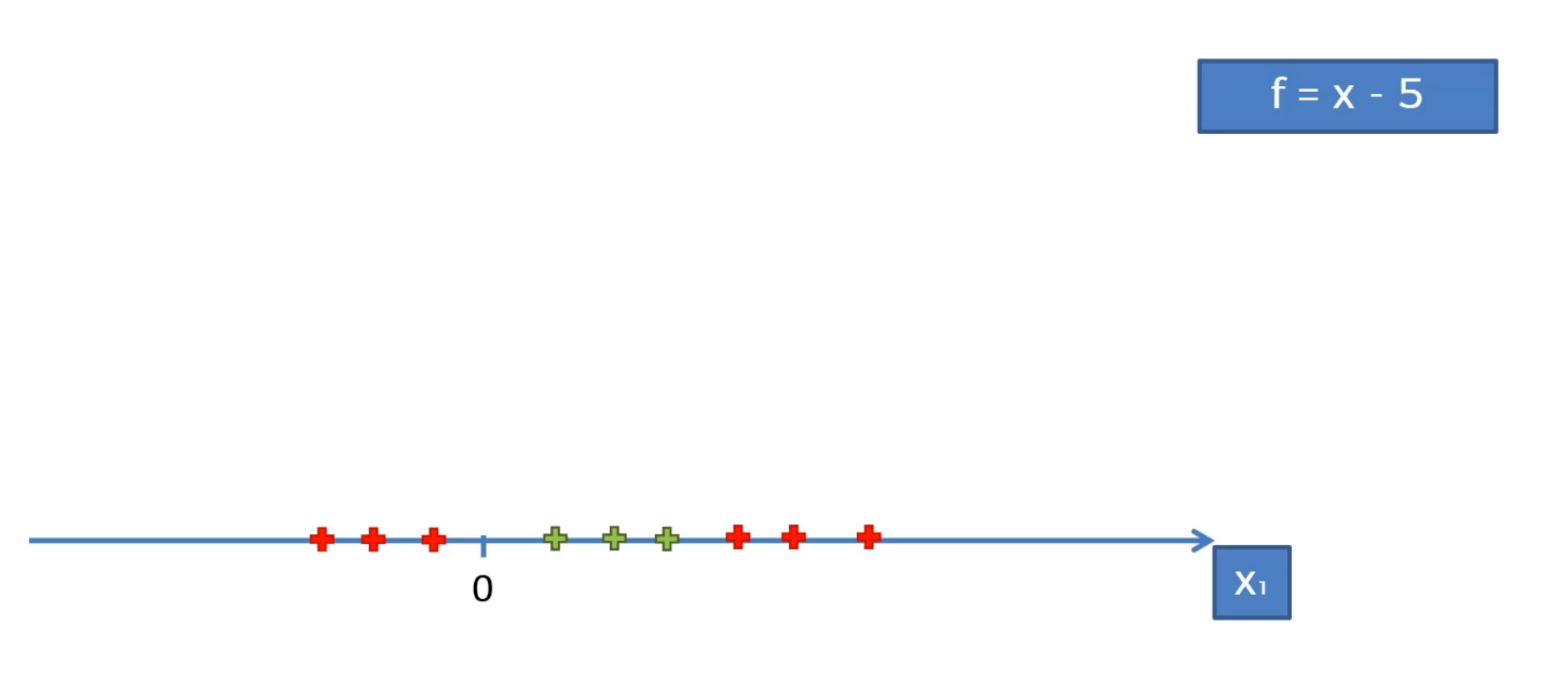
SVM (Support Vector Machine)



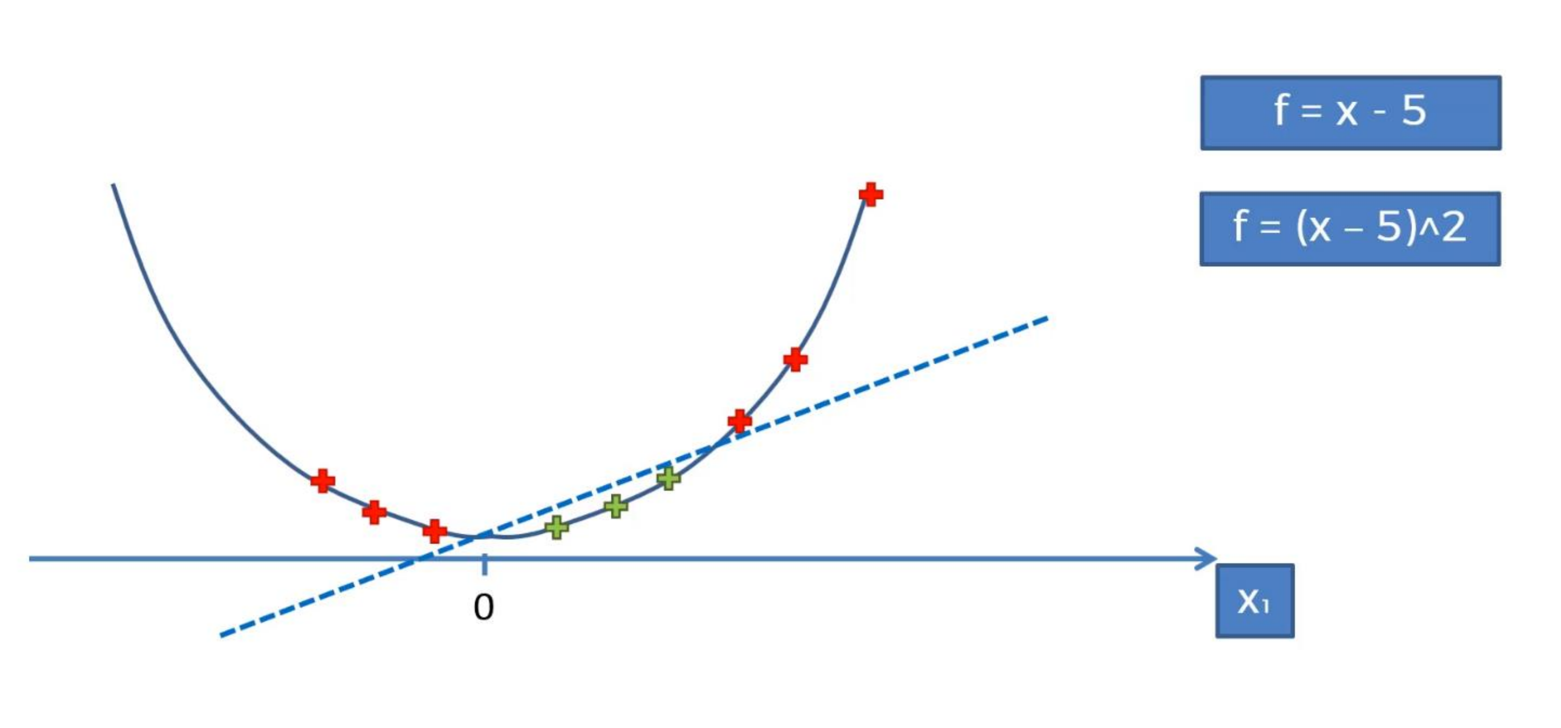
SVM (Support Vector Machine)



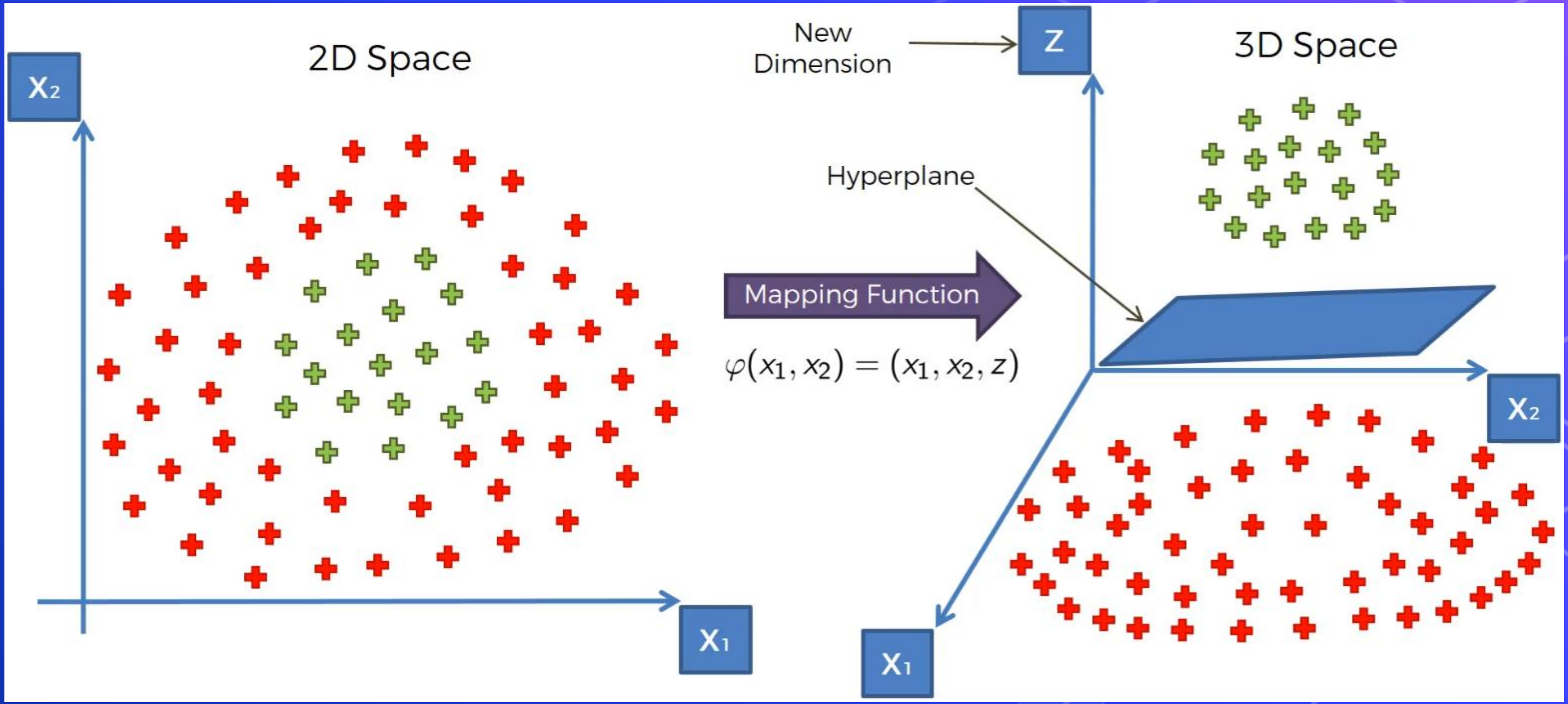
SVM (Support Vector Machine)



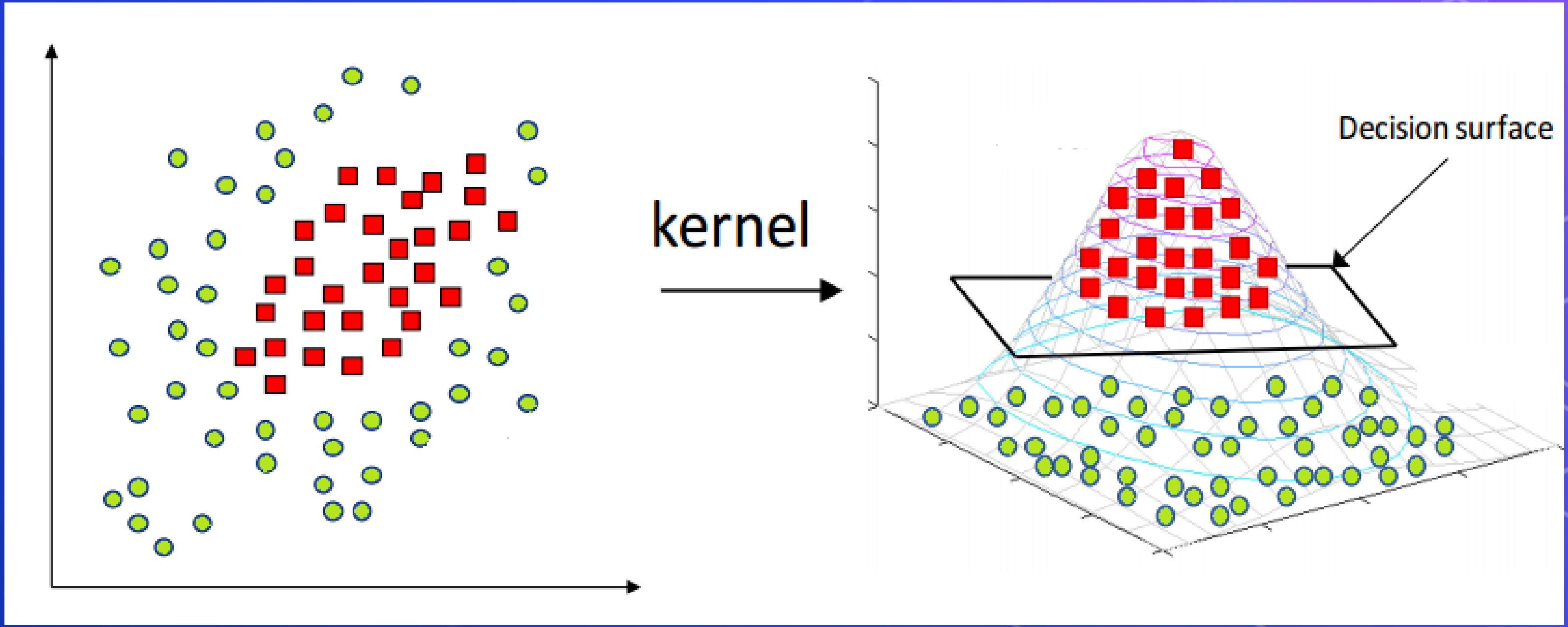
SVM (Support Vector Machine)



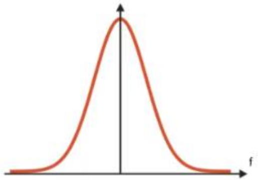
SVM (Support Vector Machine)



SVM (Support Vector Machine)

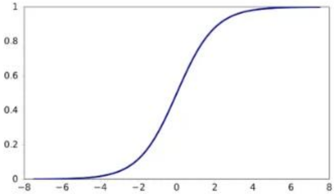


SVM (Support Vector Machine)



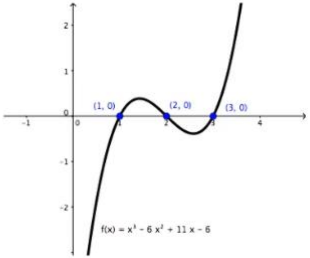
Gaussian RBF Kernel

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



Sigmoid Kernel

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



Polynomial Kernel

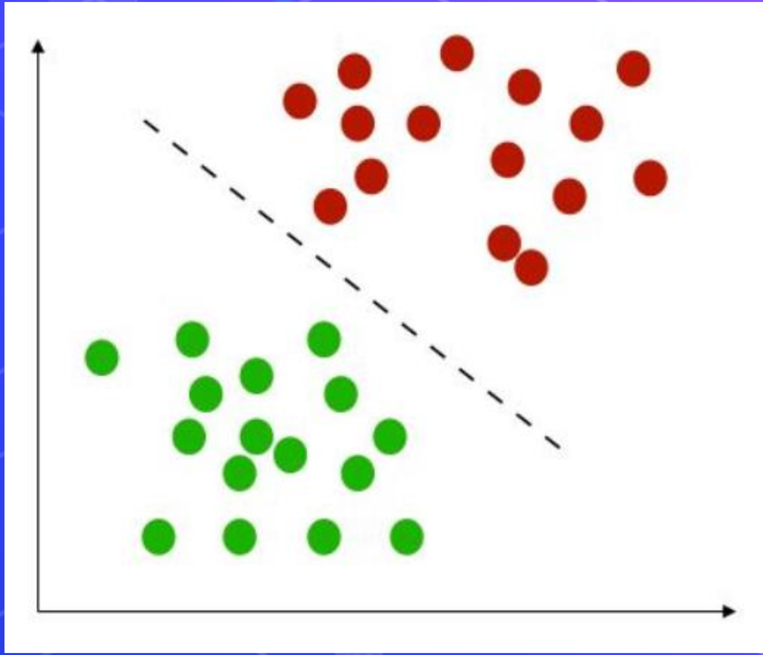
$$K(X,Y) = (\gamma \cdot X^T Y + r)^d, \gamma > 0$$

SVM (Support Vector Machine)

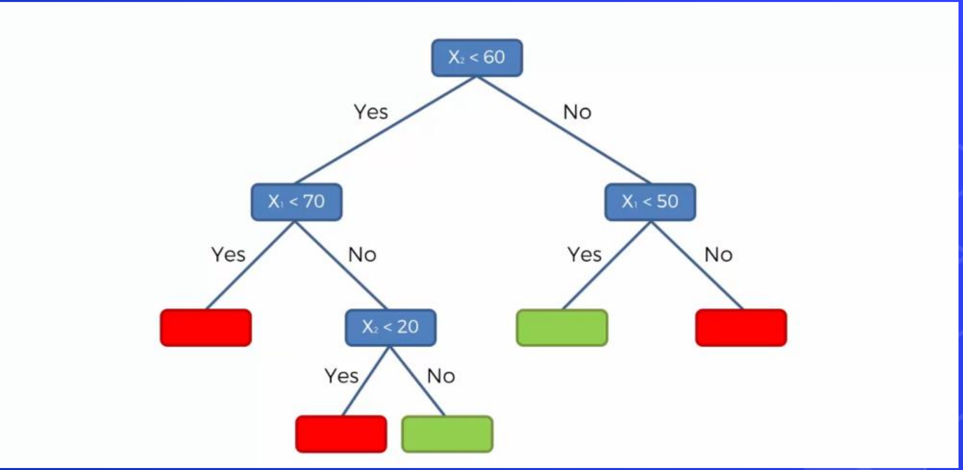
```python
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix


model = SVC()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

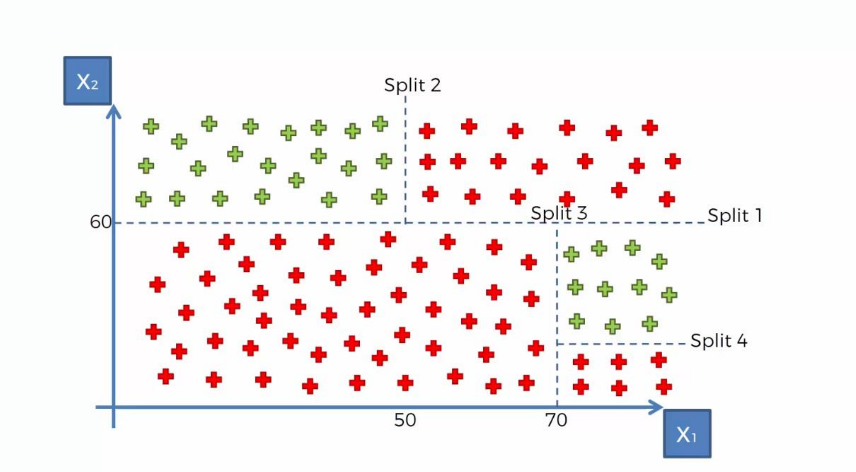
Classification

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



Decision Trees



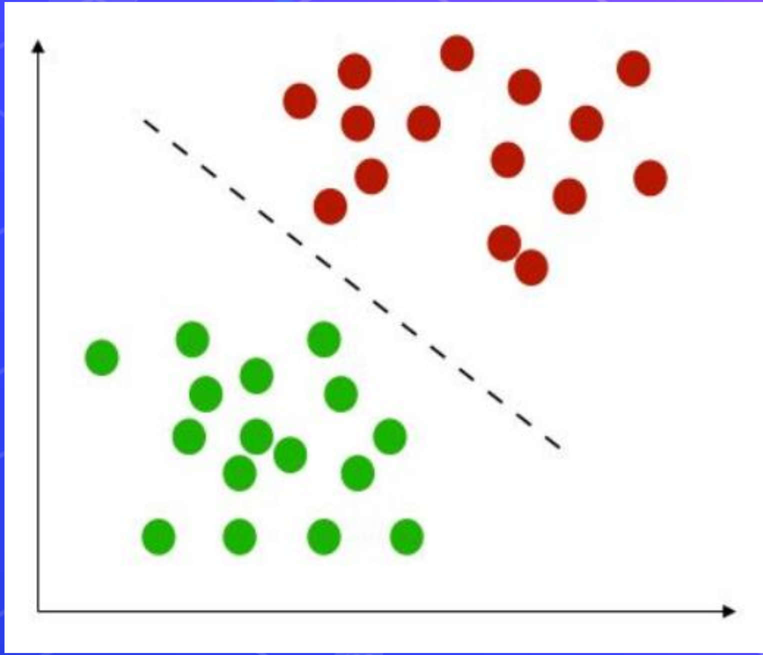


Decision Trees

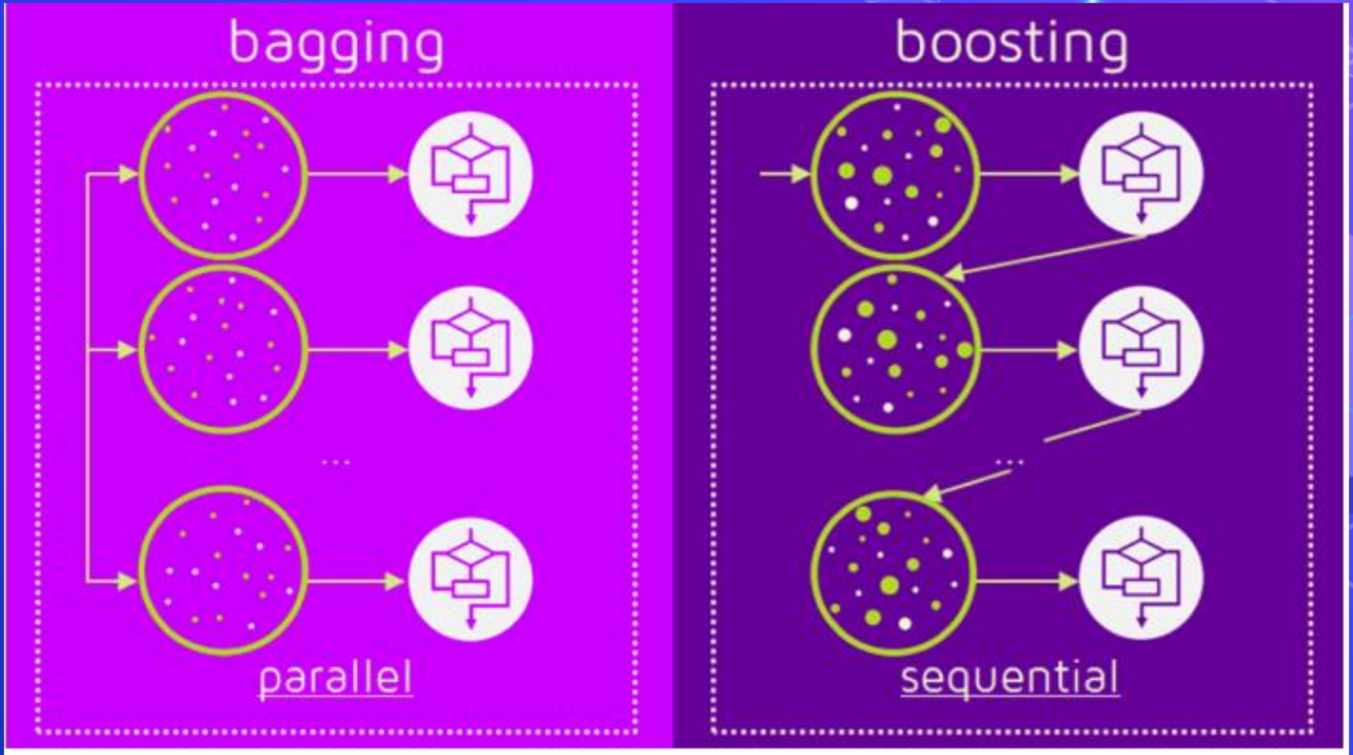
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix


model = DecisionTreeClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Classification

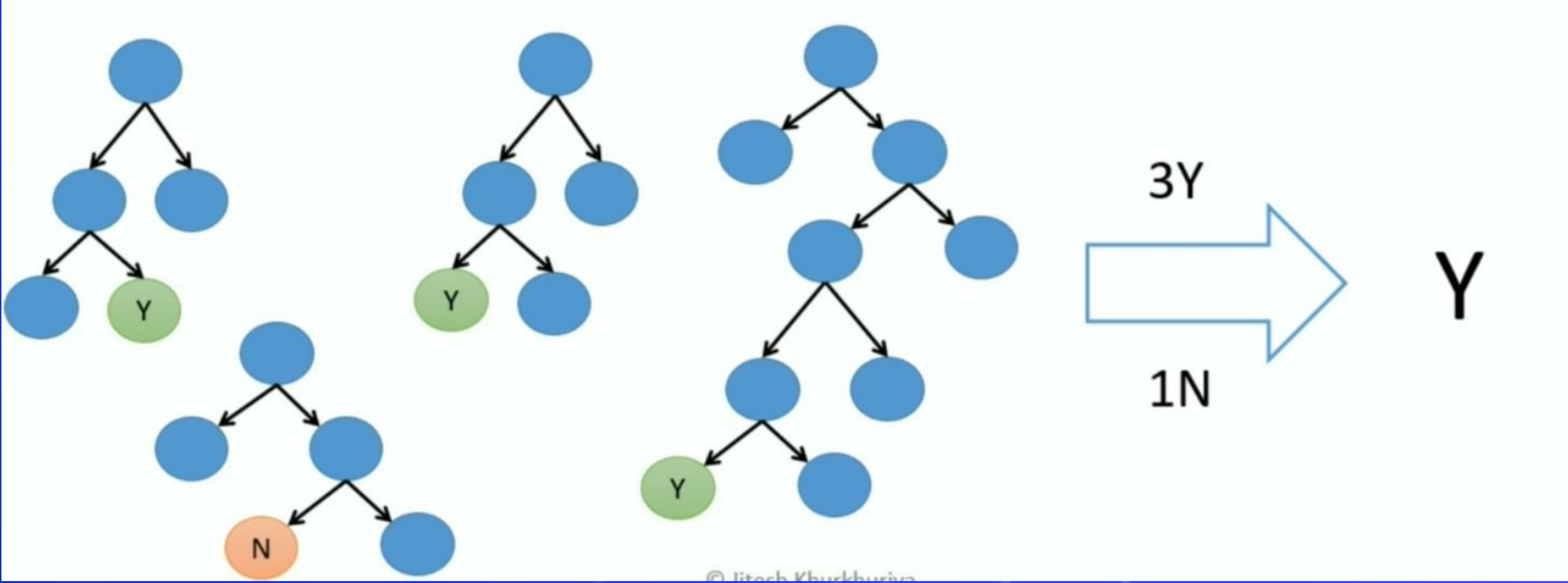
- Logistic Regression
- K-Nearest Neighbors (KNN)
- Naive Bayes
- SVM
- Decision Trees
- **Ensemble Methods**
  - What is Bagging & Boosting
  - Random Forests
  - XGBoost
- Evaluating Model Performance



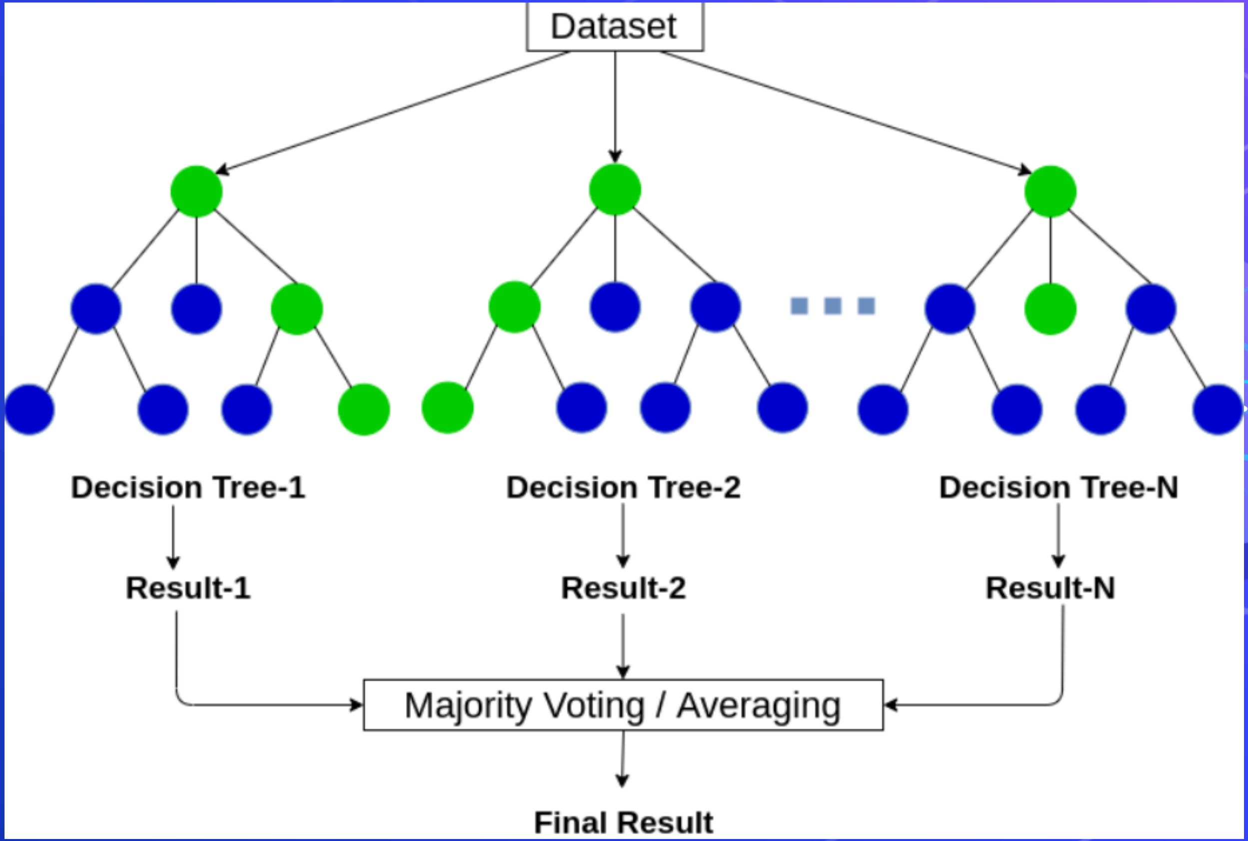
What is Bagging & Boosting



What is Bagging & Boosting



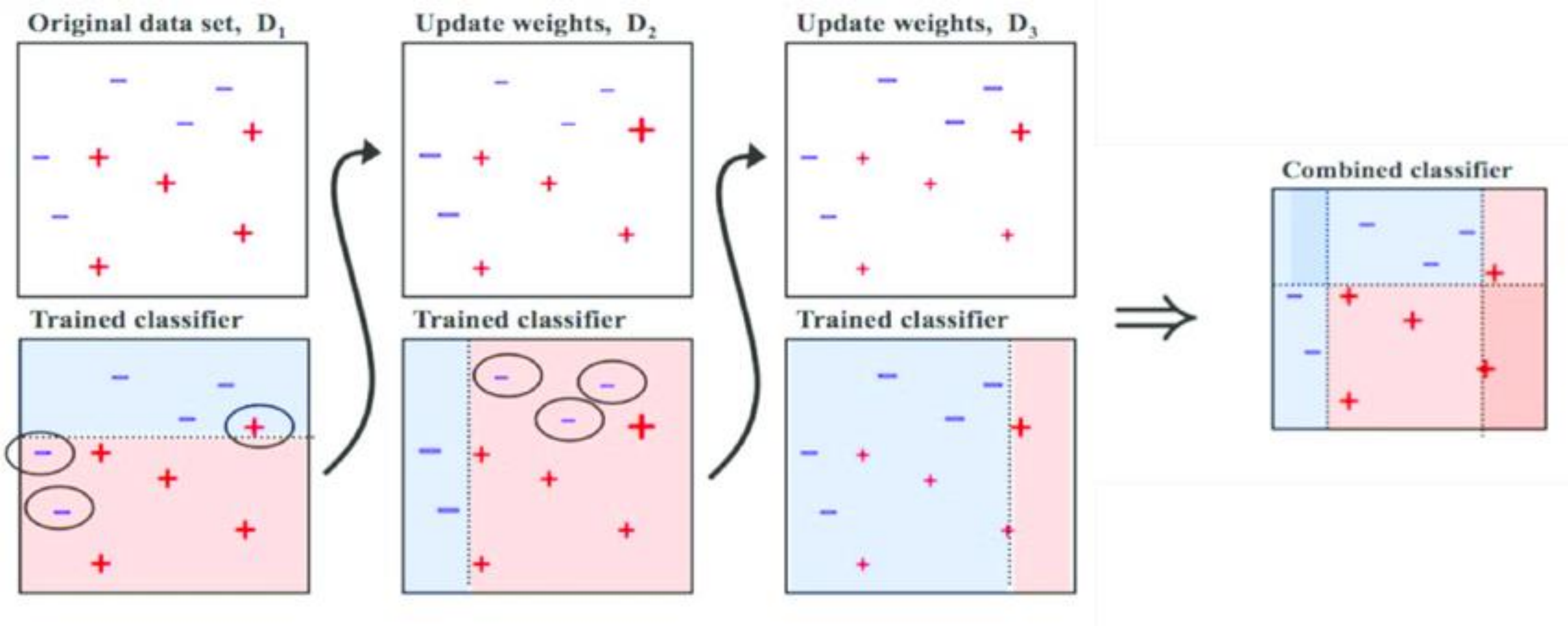
Random Forests



Random Forests

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix


model = RandomForestClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

What is Bagging & Boosting

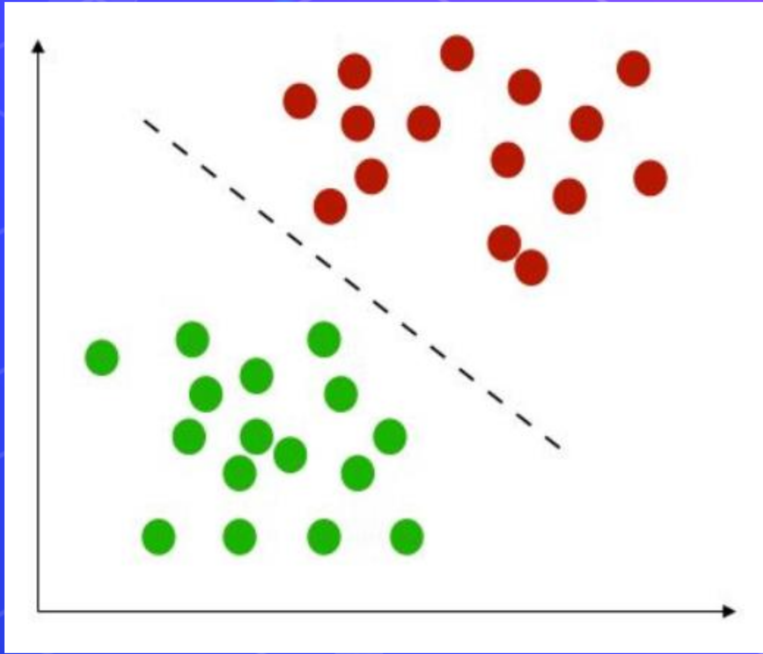


XGBoost

```python
import xgboost as xgb
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix


model = xgb.XGBClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

print(confusion_matrix(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Classification

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

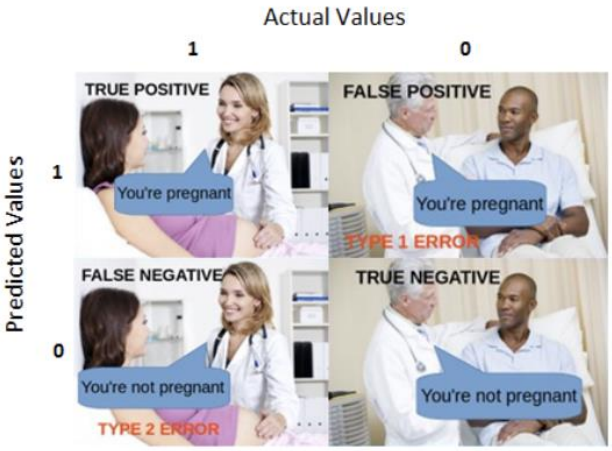


Evaluating Model Performance (Confusion Matrix)

		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)

```python
from sklearn.metrics import confusion_matrix


confusion_matrix(y_pred, y_true)

"""
array([[43,  2],
       [11, 87]], dtype=int64)
"""
```

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:

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

F_1 score:

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

Evaluating Model Performance (Accuracy)

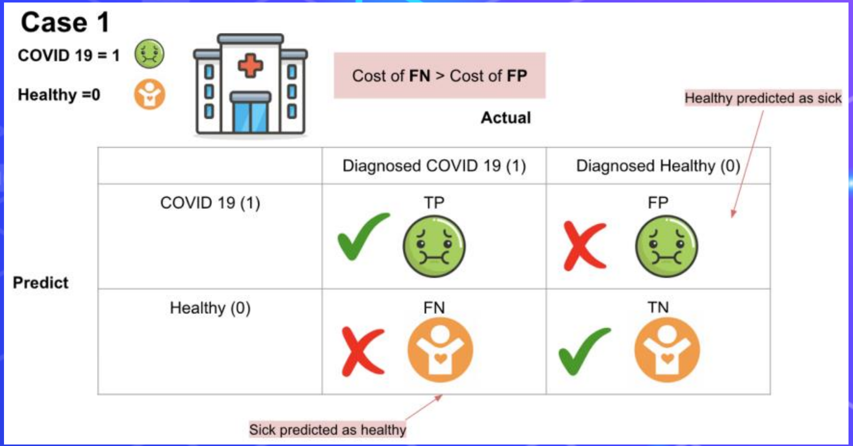
```python
from sklearn.metrics import accuracy_score


accuracy_score(y_pred, y_true)

"""
0.9090909090909091
"""
```

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.



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.

Accuracy:

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

Recall:

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

Precision:

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

F_1 score:

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

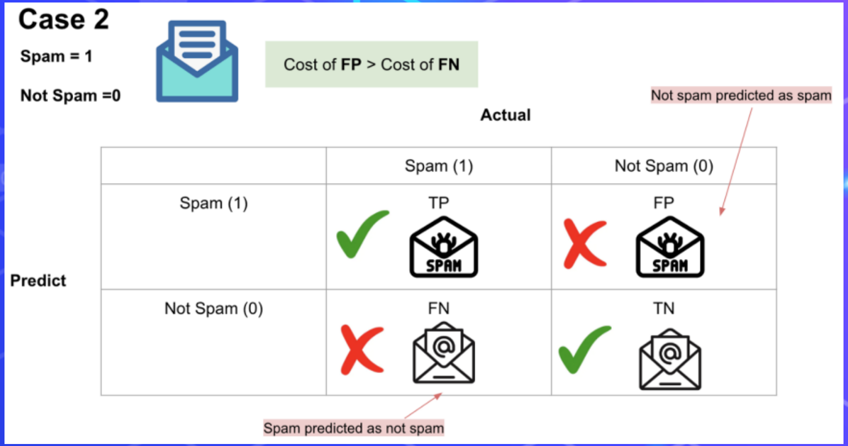
$$\textbf{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \text{ or } \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

Evaluating Model Performance (Recall or Sensitivity)

```python
from sklearn.metrics import recall_score


recall_score(y_pred, y_true)

"""
0.9090909090909091
"""
```

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.

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.

Accuracy:

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

Recall:

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

Precision:

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

F_1 score:

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

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

Evaluating Model Performance (Precision)

```python
from sklearn.metrics import precision_score


precision_score(y_pred, y_true)

"""
0.9090909090909091
"""
```

Evaluating Model Performance

Case 1

COVID 19/ Healthy

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

Recall

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

Case 2

Spam/Not Spam

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

Precision

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

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.

Accuracy:

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

Recall:

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

Precision:

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

F_1 score:

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

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

Evaluating Model Performance (F-1 Score)

```python
from sklearn.metrics import f1_score


f1_score(y_pred, y_true)

"""
0.9090909090909091
"""
```

Evaluating Model Performance (Classification Report)

```python
from sklearn.metrics import classification_report


classification_report(y_true, y_pred)

"""
                precision    recall  f1-score   support
     class 0         0.50      1.00      0.67         1
     class 1         0.00      0.00      0.00         1
     class 2         1.00      0.67      0.80         3
"""
```

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.

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

Evaluating Model Performance (F-beta Score)

```python
from sklearn.metrics import fbeta_score


fbeta_score(y_true, y_pred, beta=0.5)

"""
0.9090909090909091
"""
```

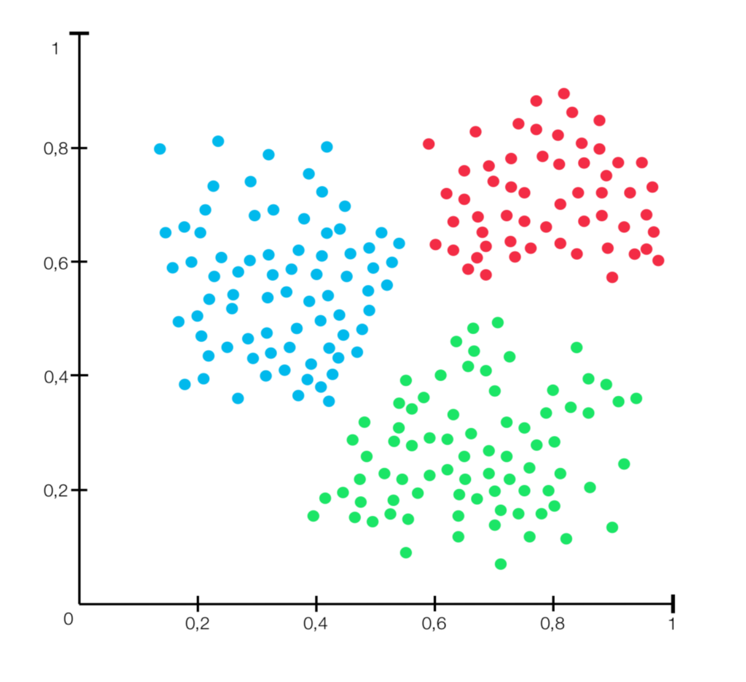
Agenda

- What is Machine Learning
- Supervised Learning
  - Regression
  - Classification
- **Unsupervised Learning**
  - **<u>Clustering</u>**
  - Association rule mining
  - Dimension Reduction
- Model Selection & Evaluation
  - Cross Validation
  - Hyperparameter Tuning
- Reinforcement Learning

Clustering

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups.

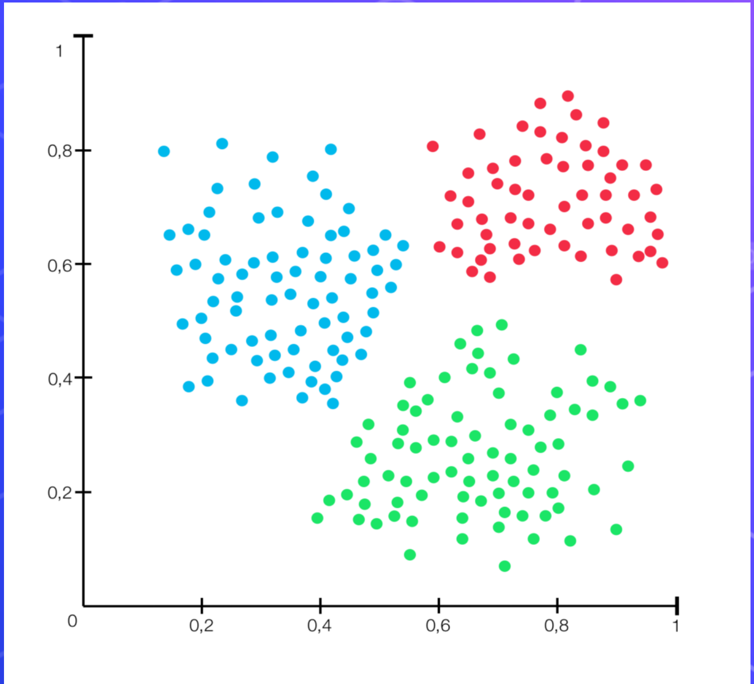
For example,
it can be used to cluster the university students to many segments depending on features like scores, departments, etc.



- University students clustering
- Mall customers segmentation
- Recommendation systems
- ...

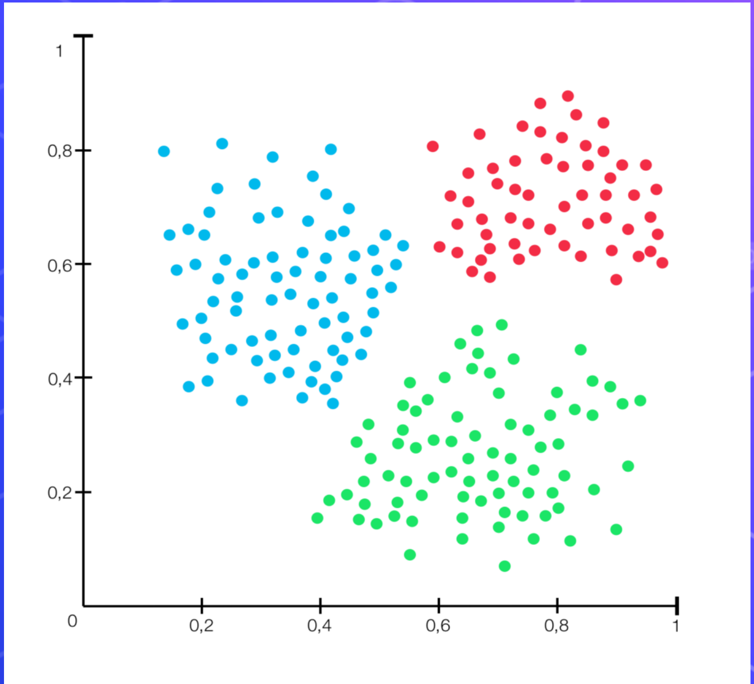
Clustering

- KMeans
- Hierarchical Clustering
- Density Based Clustering – DBSCAN
- Gaussian Mixture Model – GMM
- Evaluating Model Performance

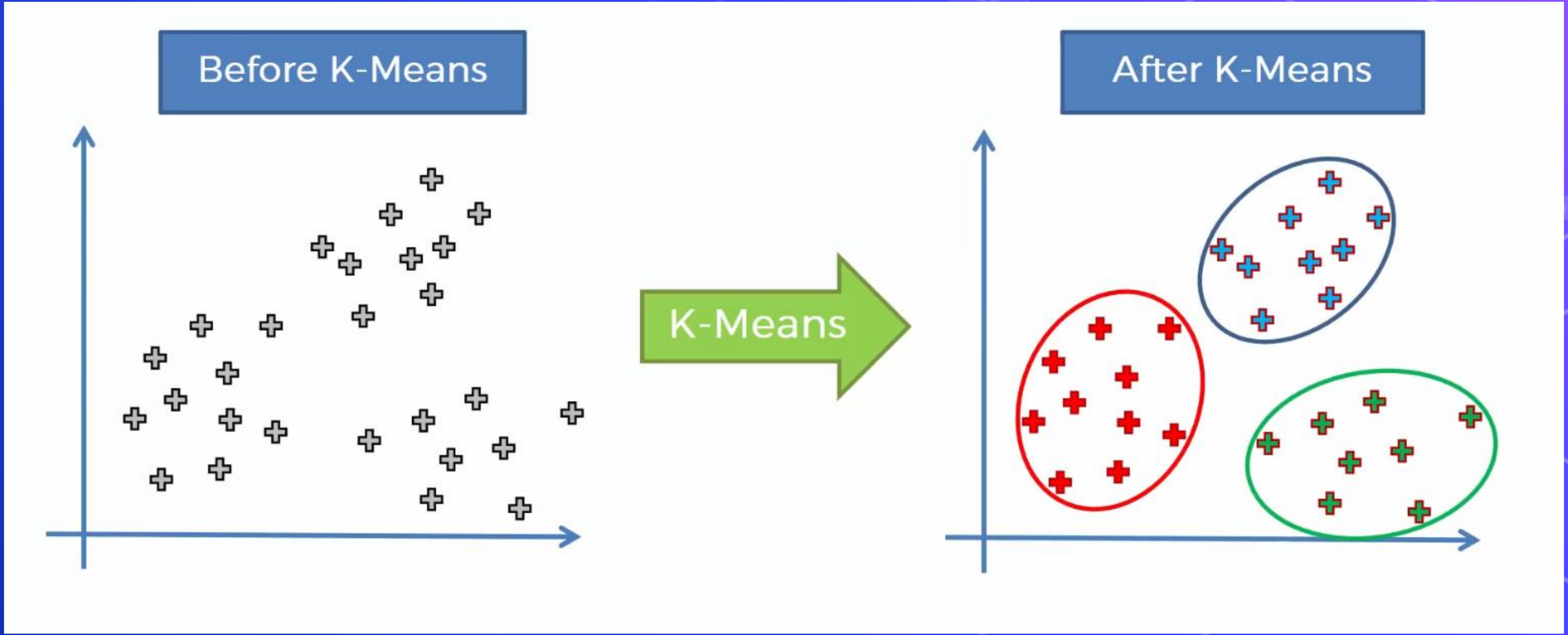


Clustering

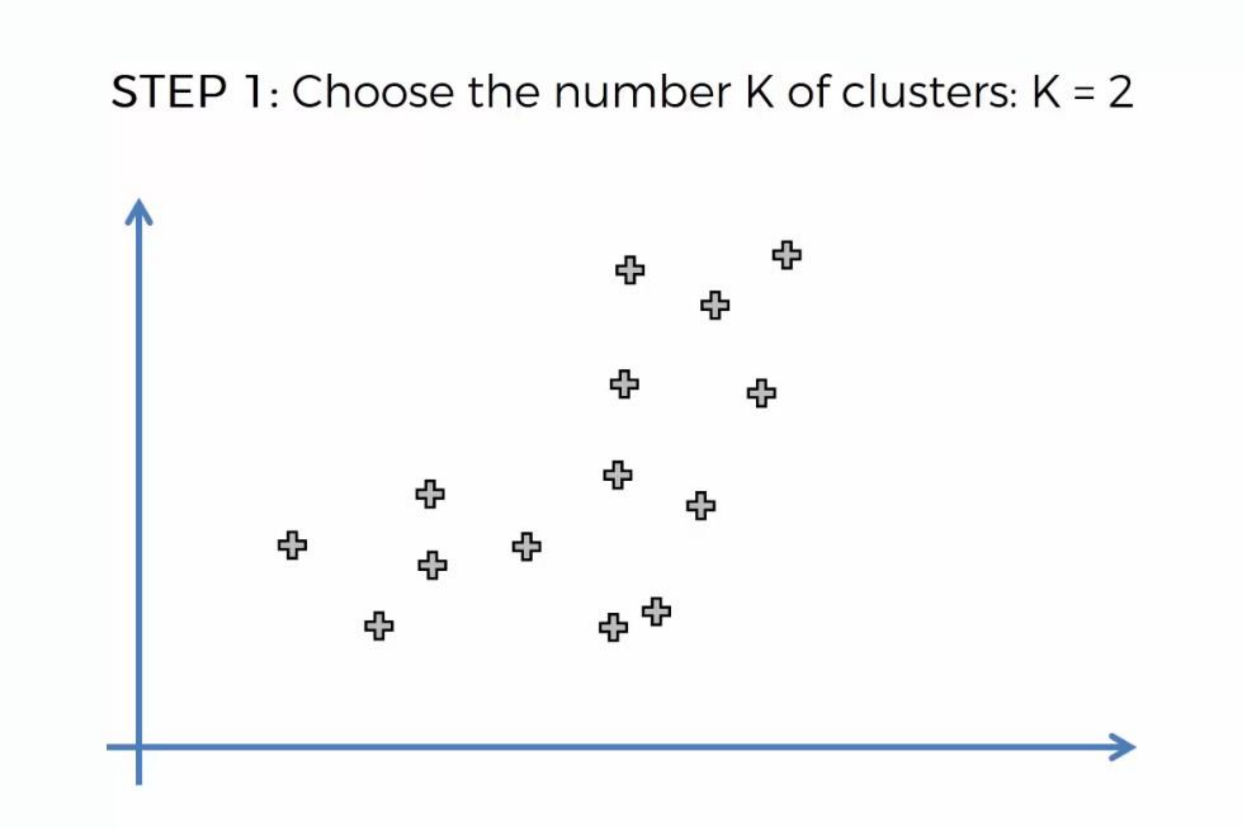
- KMeans
- Hierarchical Clustering
- Density Based Clustering – DBSCAN
- Gaussian Mixture Model – GMM
- Evaluating Model Performance



KMeans

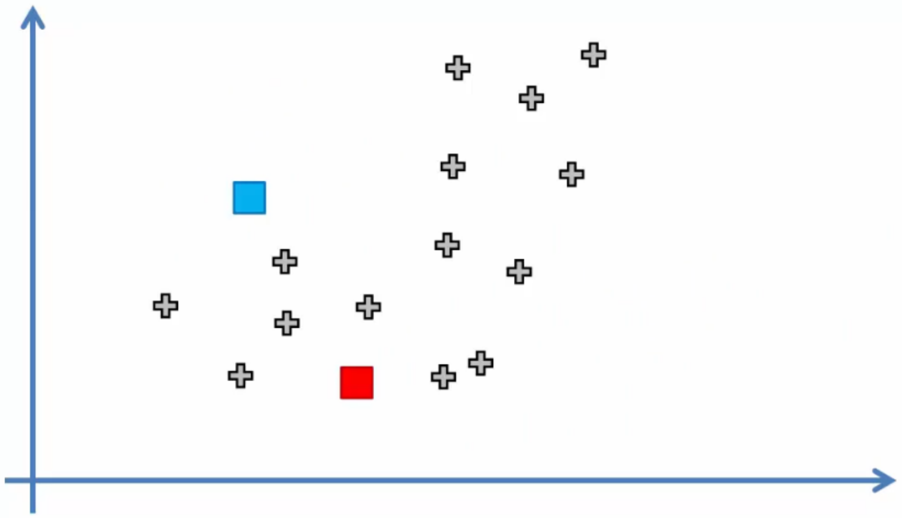


KMeans



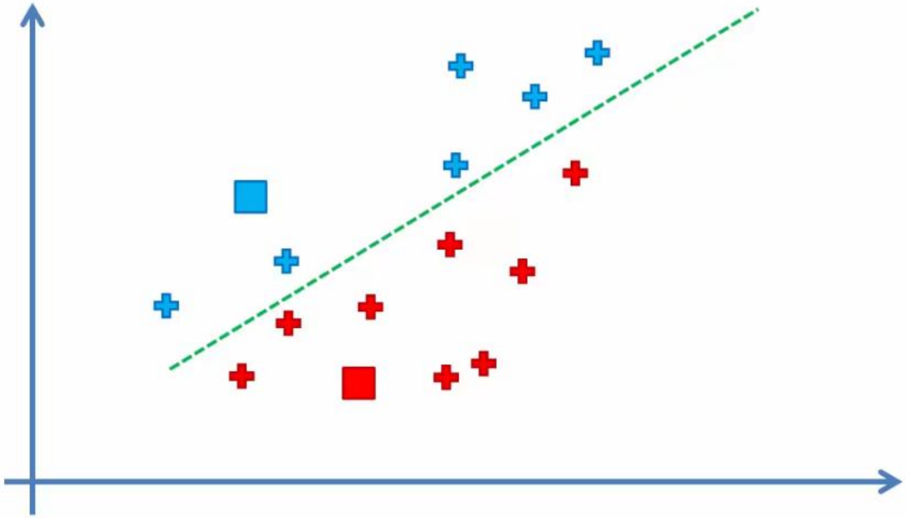
KMeans

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



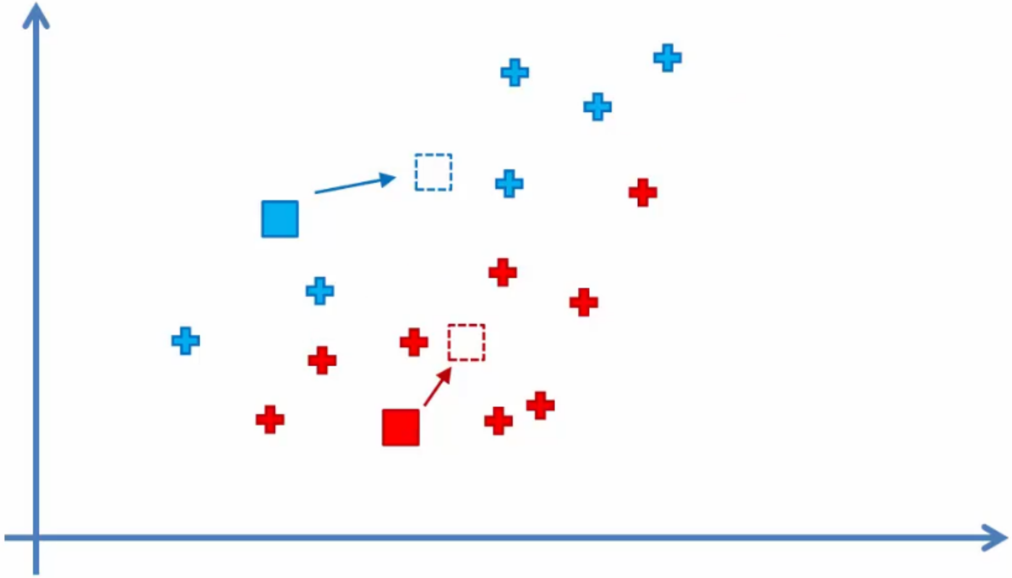
KMeans

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



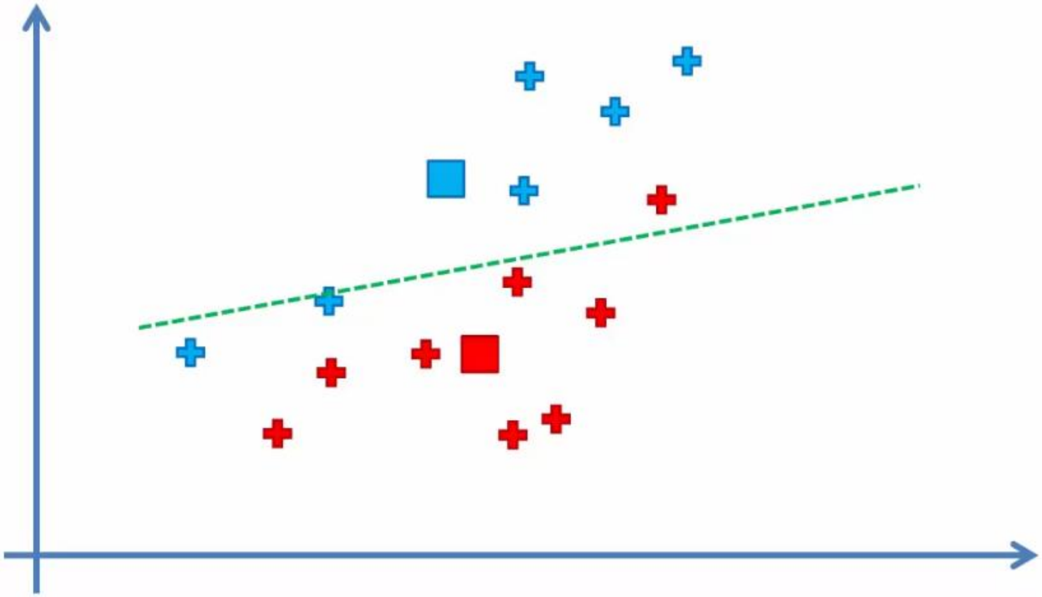
KMeans

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



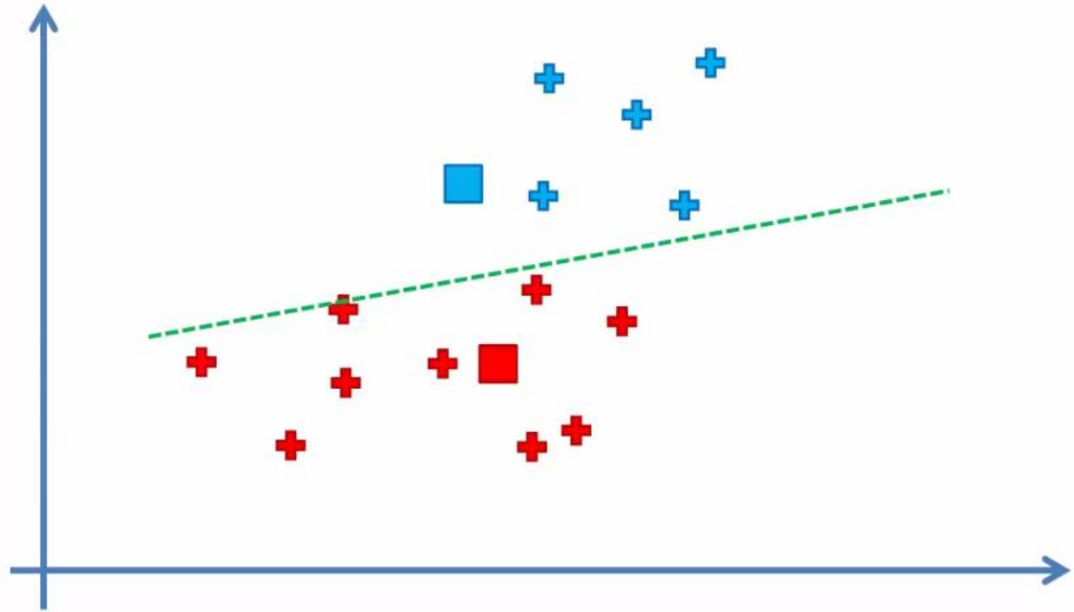
KMeans

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

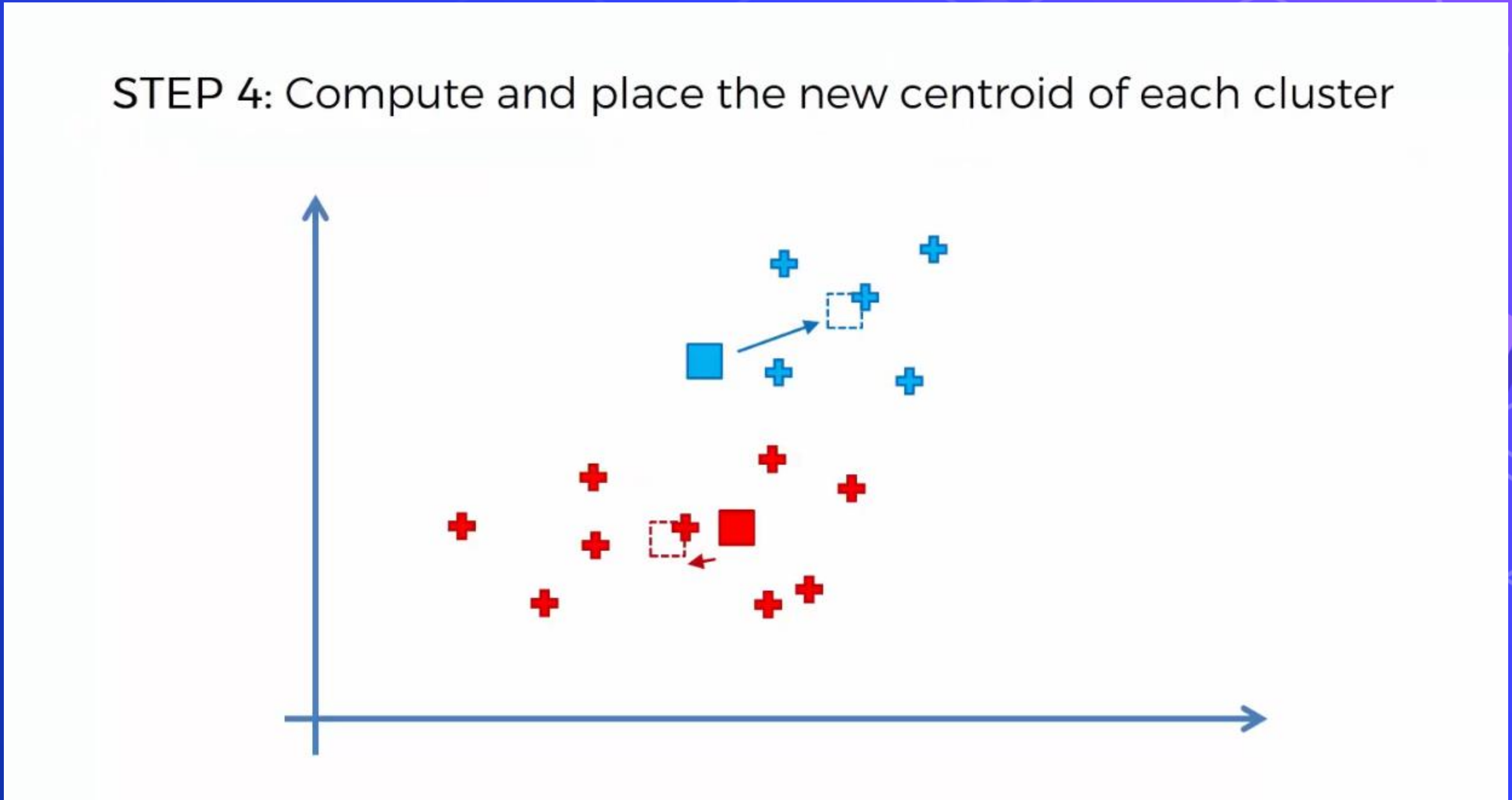


KMeans

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

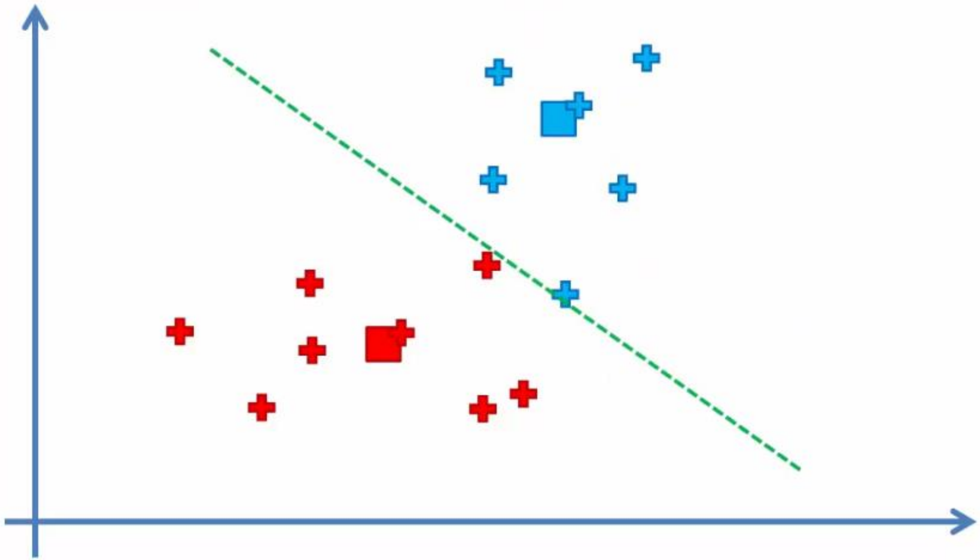


KMeans



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.

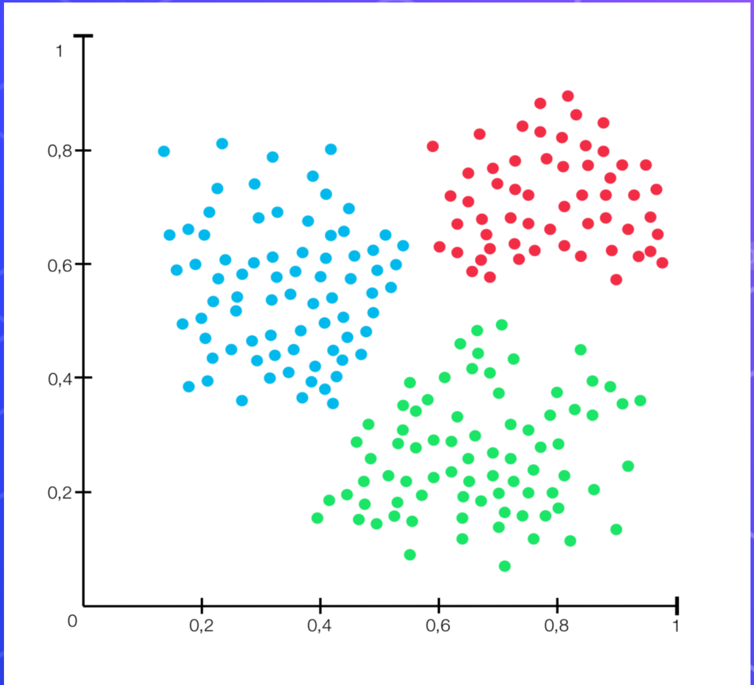


KMeans

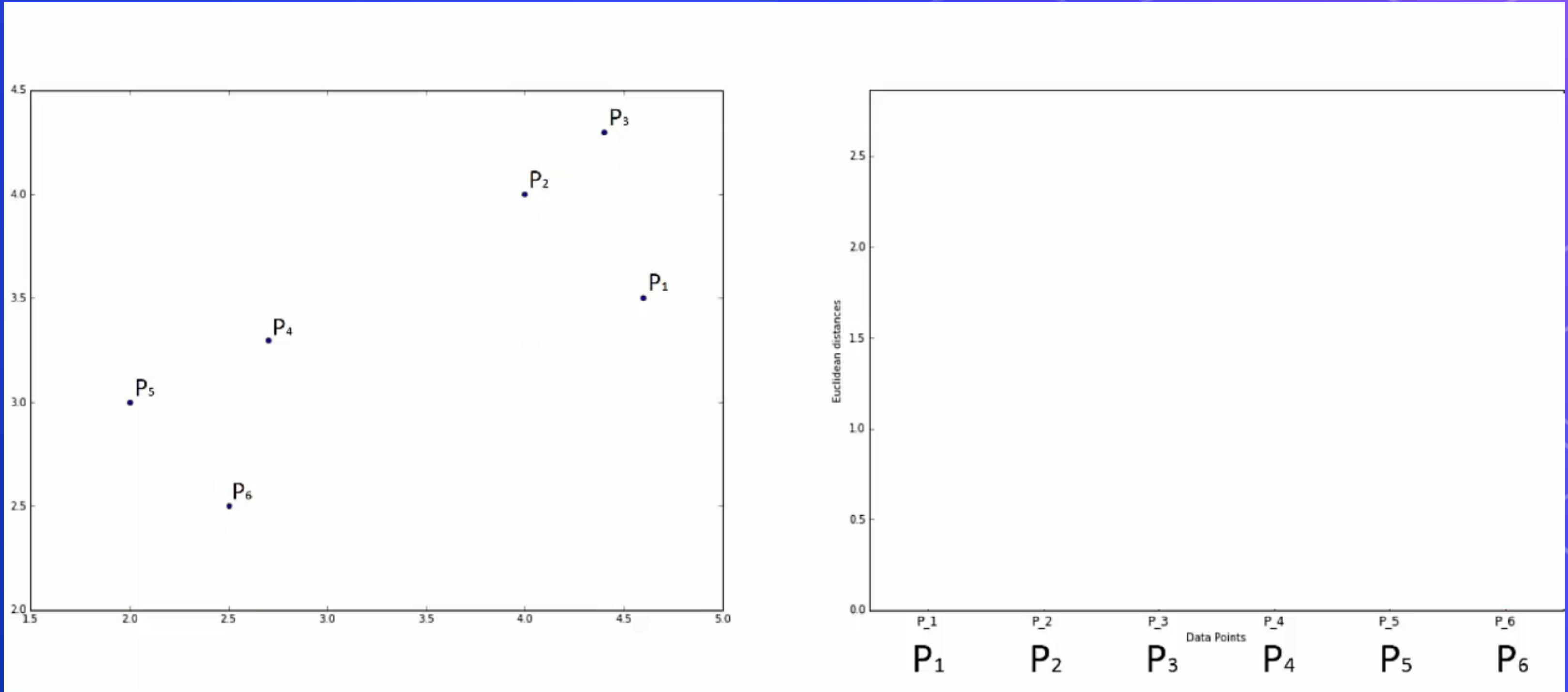
```python
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
kmeans.predict(X)
```

Clustering

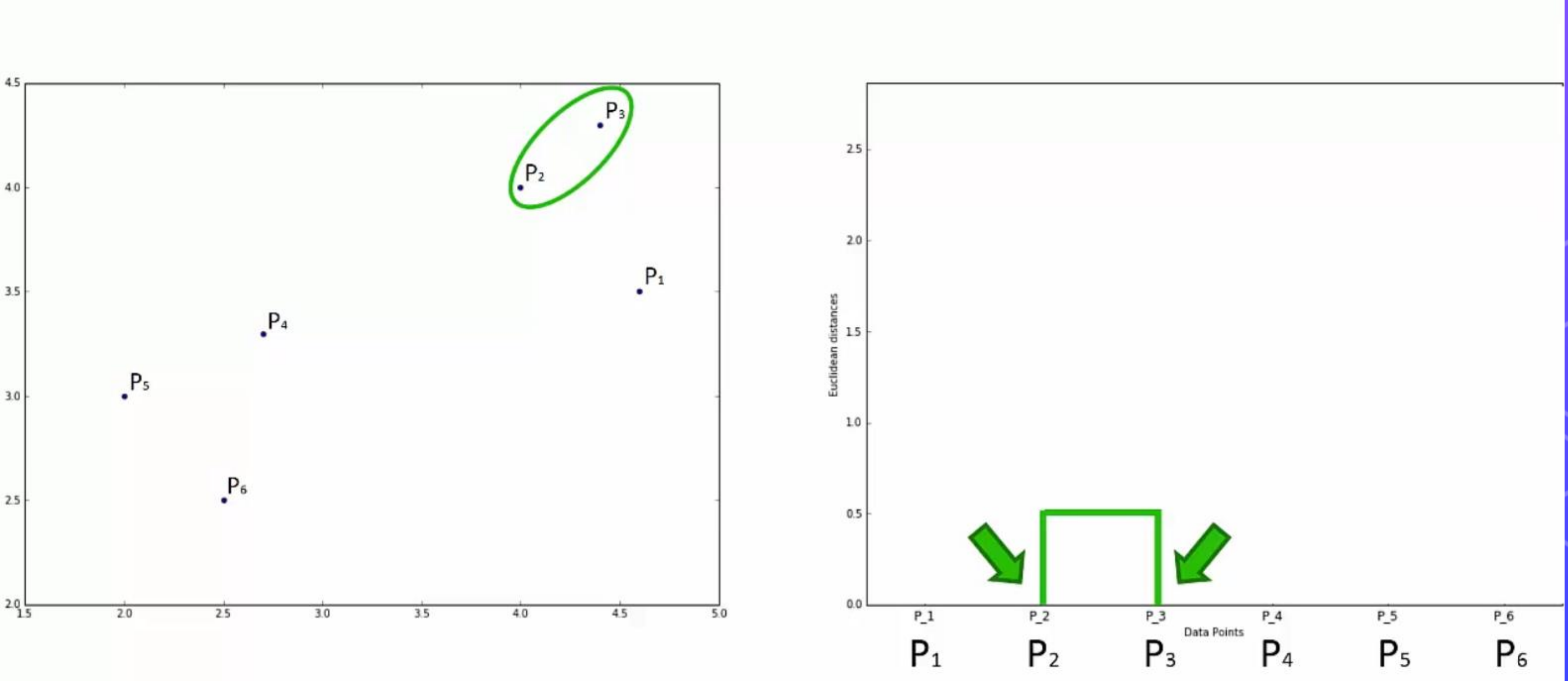
- KMeans
- **Hierarchical Clustering**
- Density Based Clustering – DBSCAN
- Gaussian Mixture Model – GMM
- Evaluating Model Performance



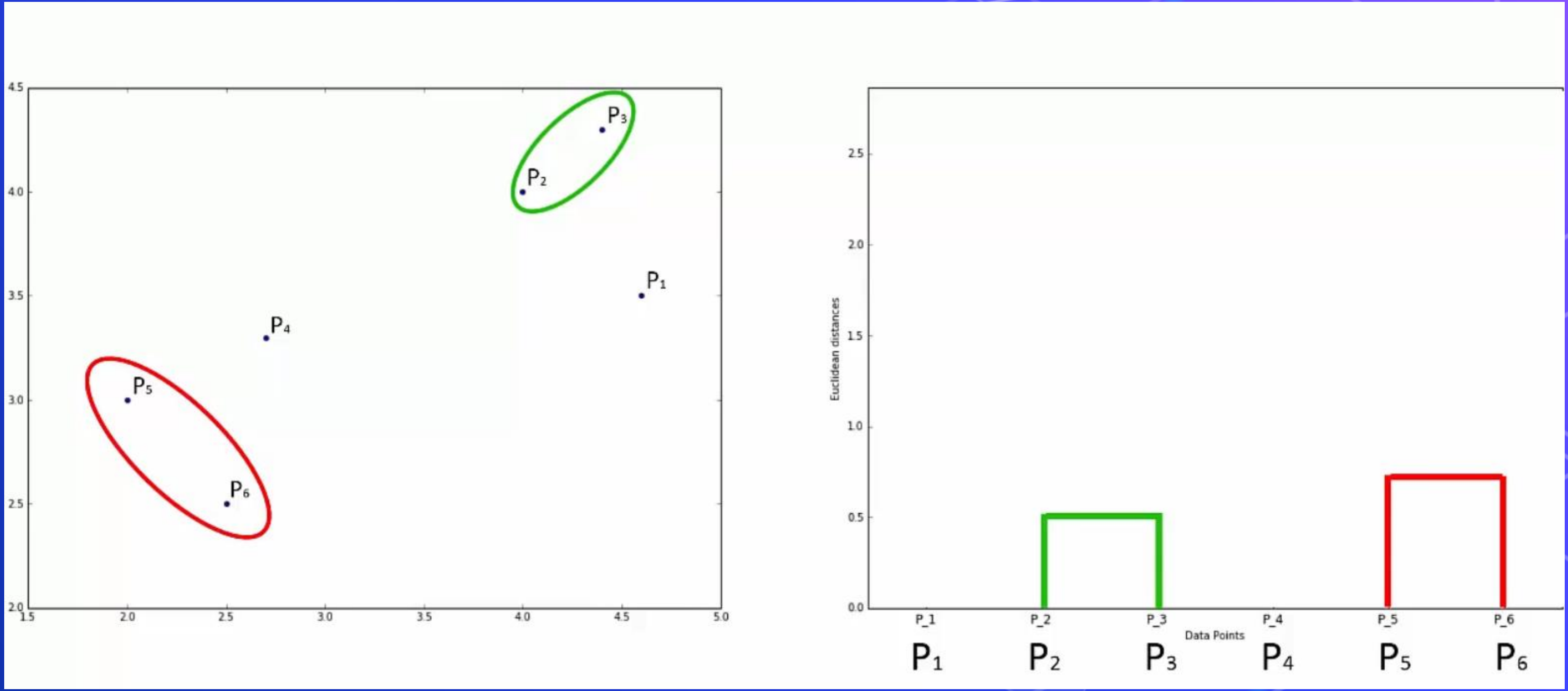
Hierarchical Clustering with dendrogram



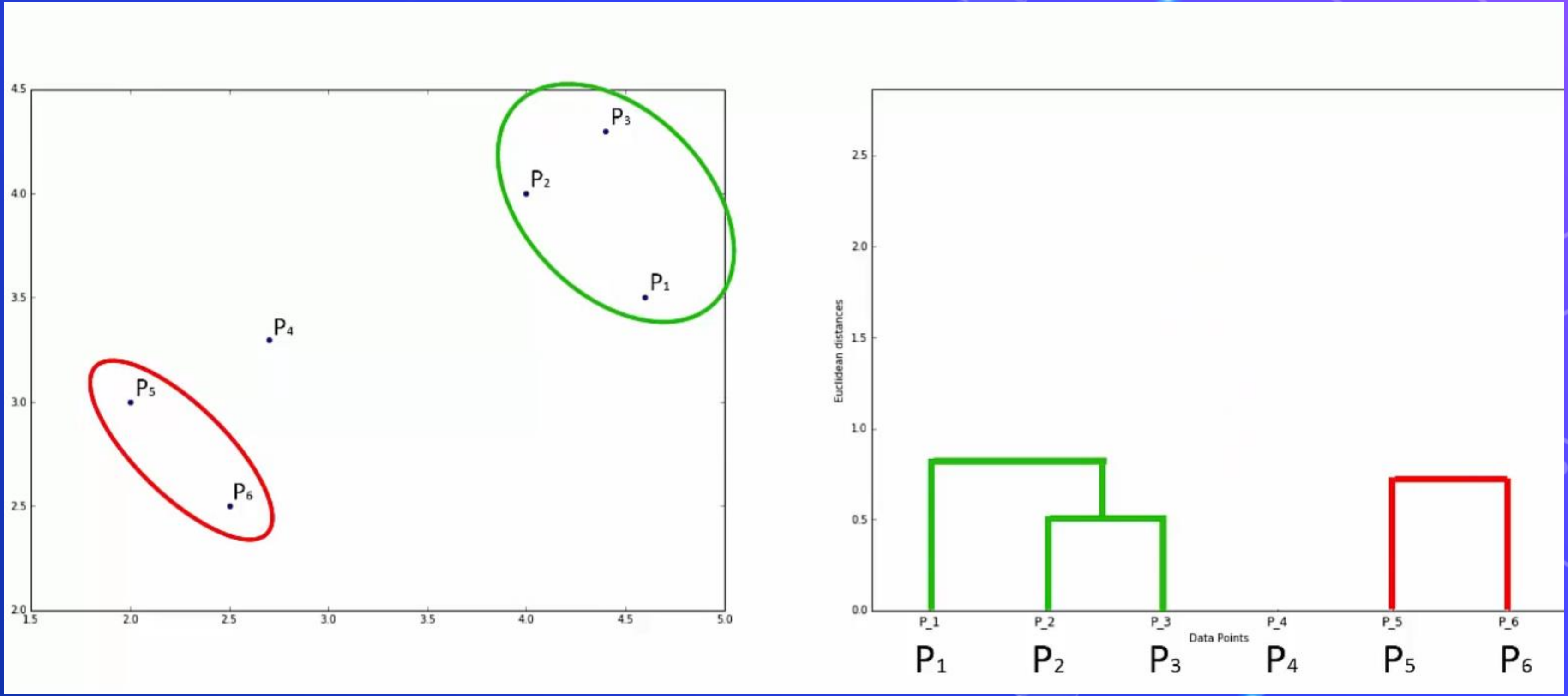
Hierarchical Clustering with dendrogram



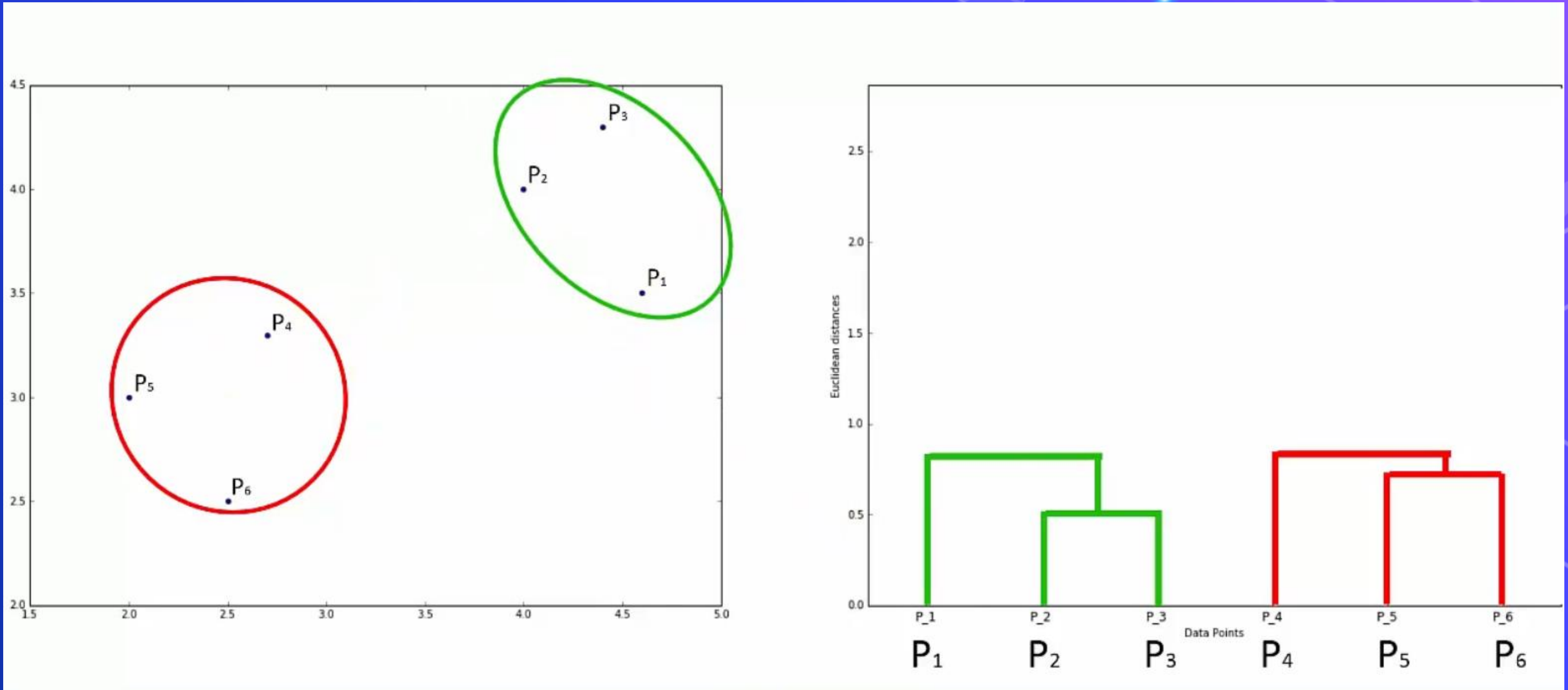
Hierarchical Clustering with dendrogram



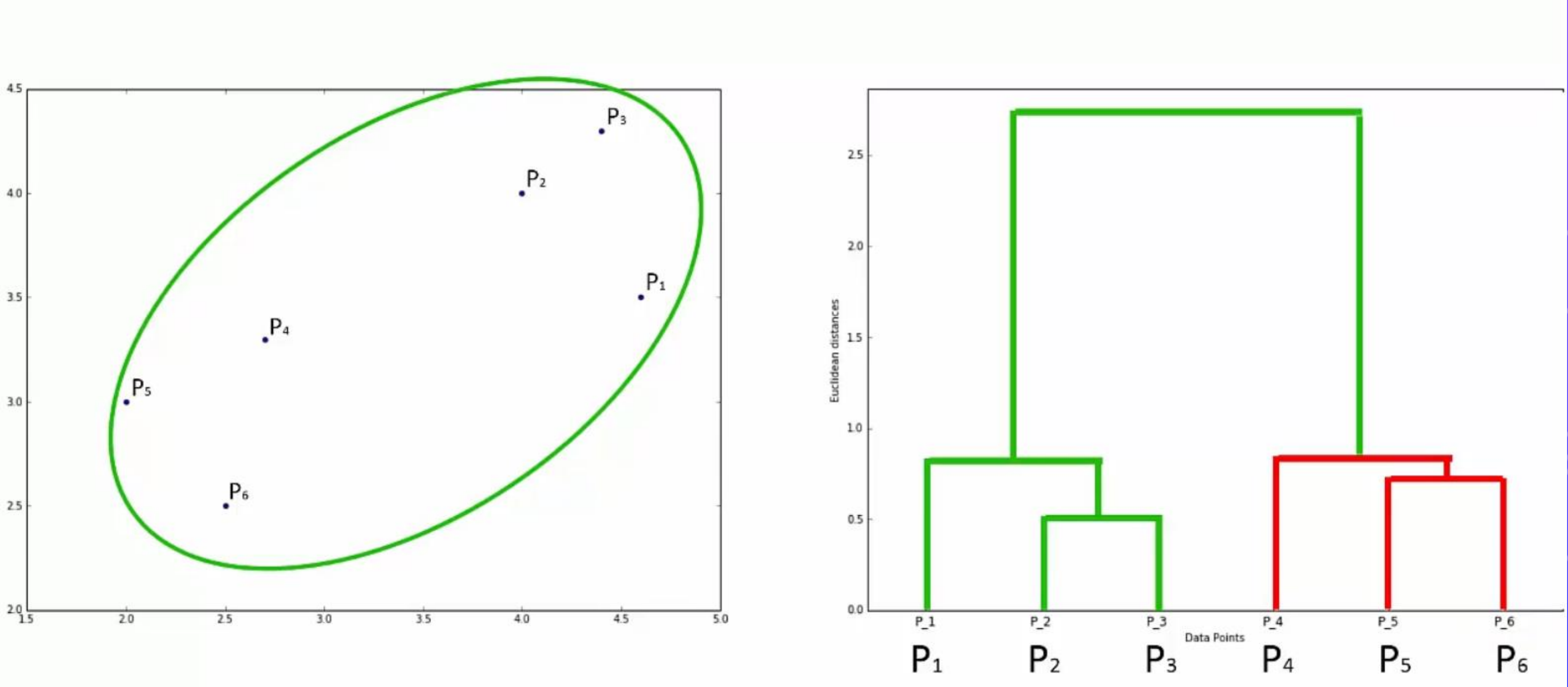
Hierarchical Clustering with dendrogram



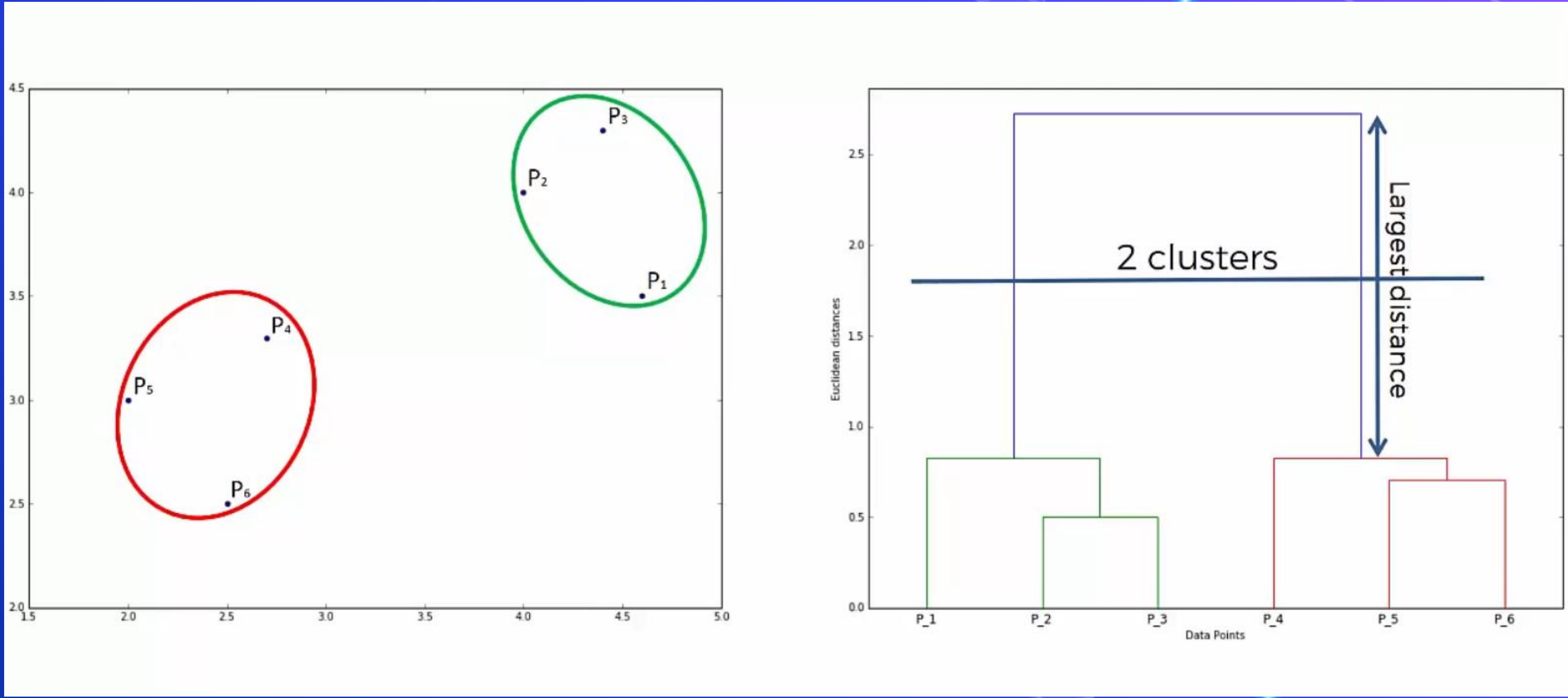
Hierarchical Clustering with dendrogram



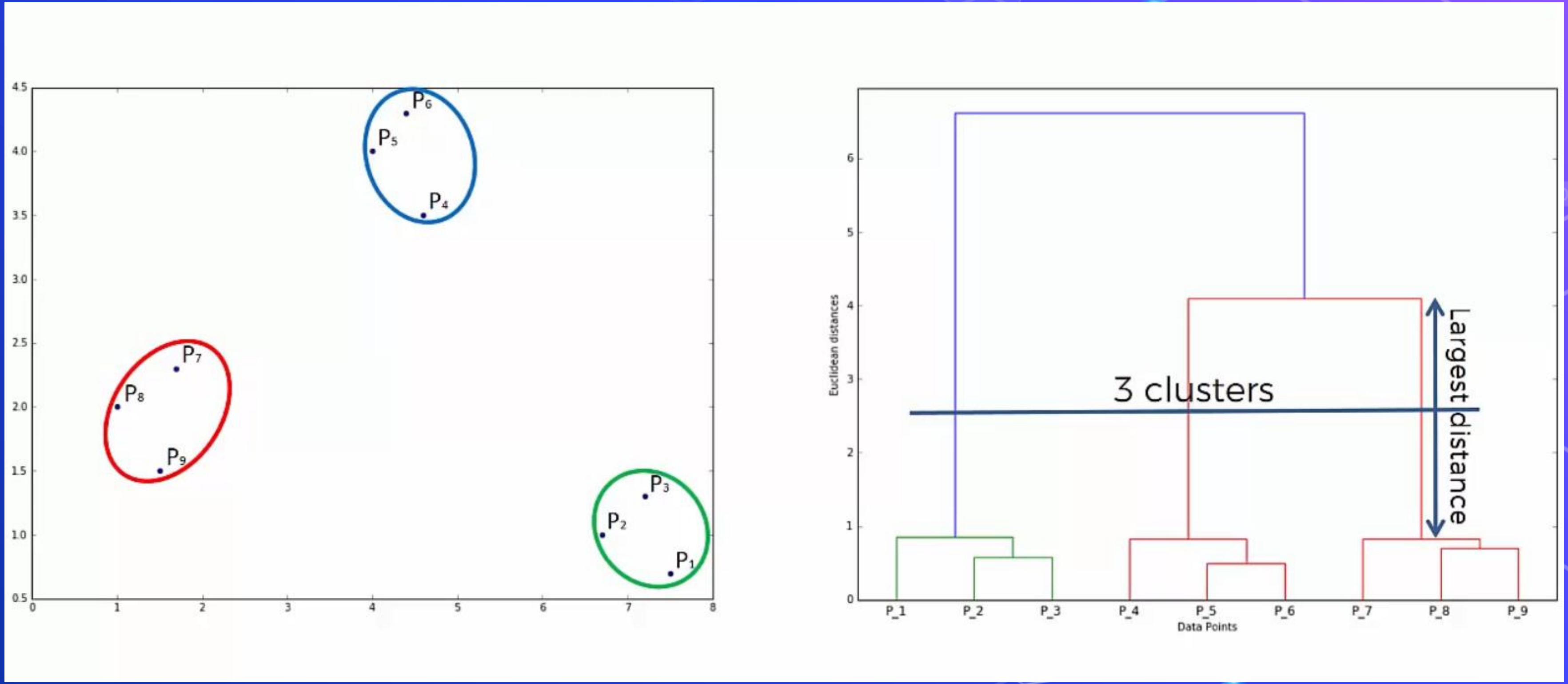
Hierarchical Clustering with dendrogram



Hierarchical Clustering with dendrogram



Hierarchical Clustering with dendrogram

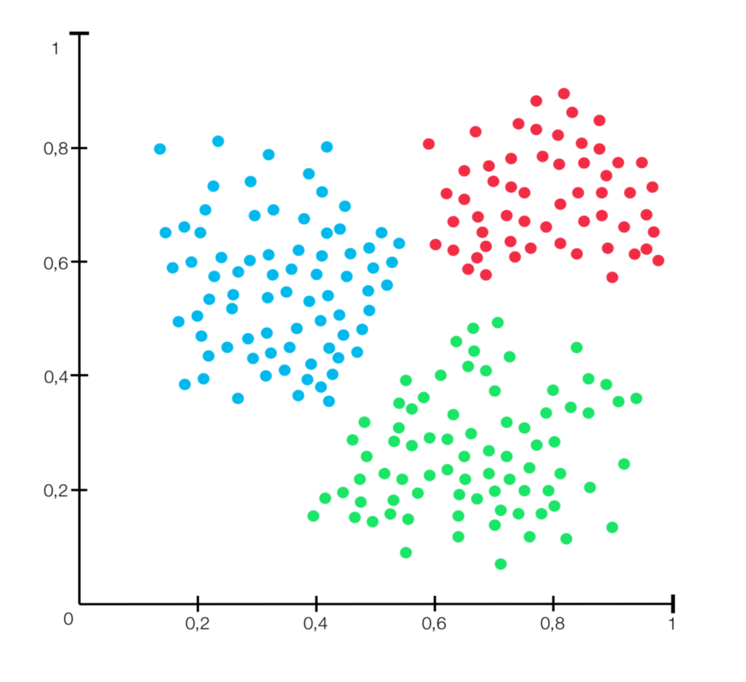


Hierarchical Clustering with dendrogram

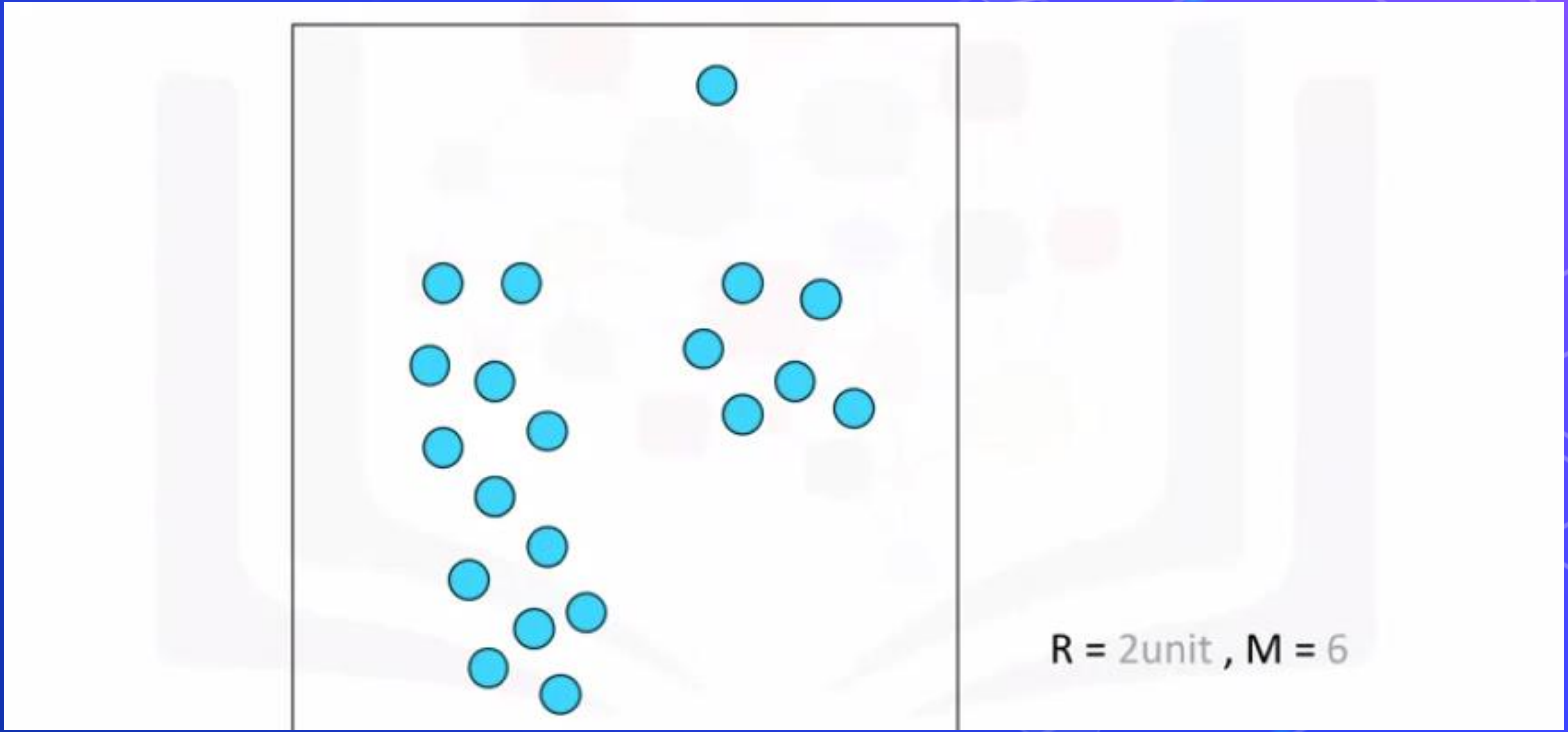
```python
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering

# visualize dendrogram
dendrogram = sch.dendrogram(sch.linkage(x, method='ward'))

# train model
model = AgglomerativeClustering(n_clusters=3)
y_labels = model.fit_predict(x)
```

Clustering

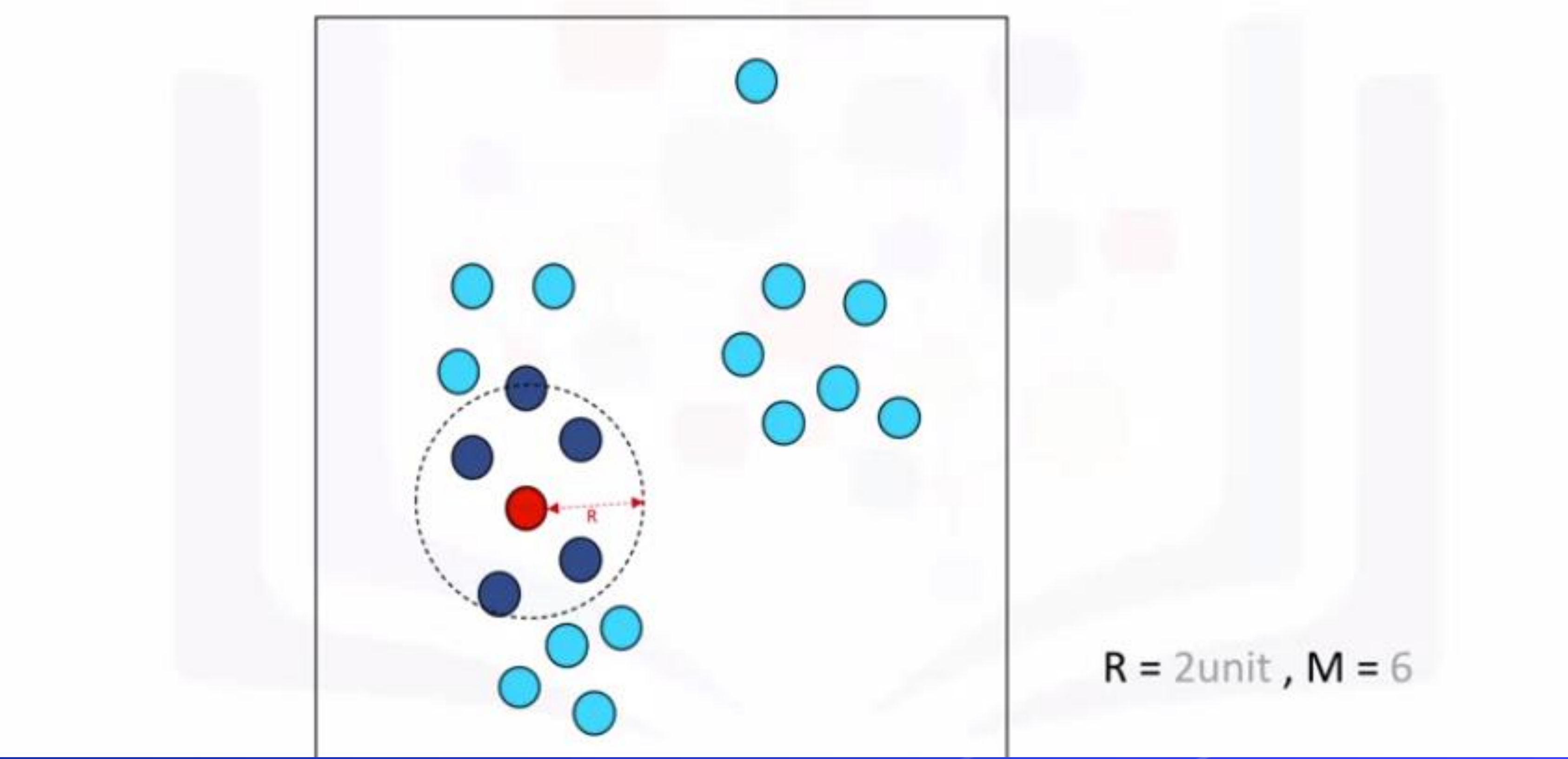
- KMeans
- Hierarchical Clustering
- <u>Density Based Clustering – DBSCAN</u>
- Gaussian Mixture Model – GMM
- Evaluating Model Performance



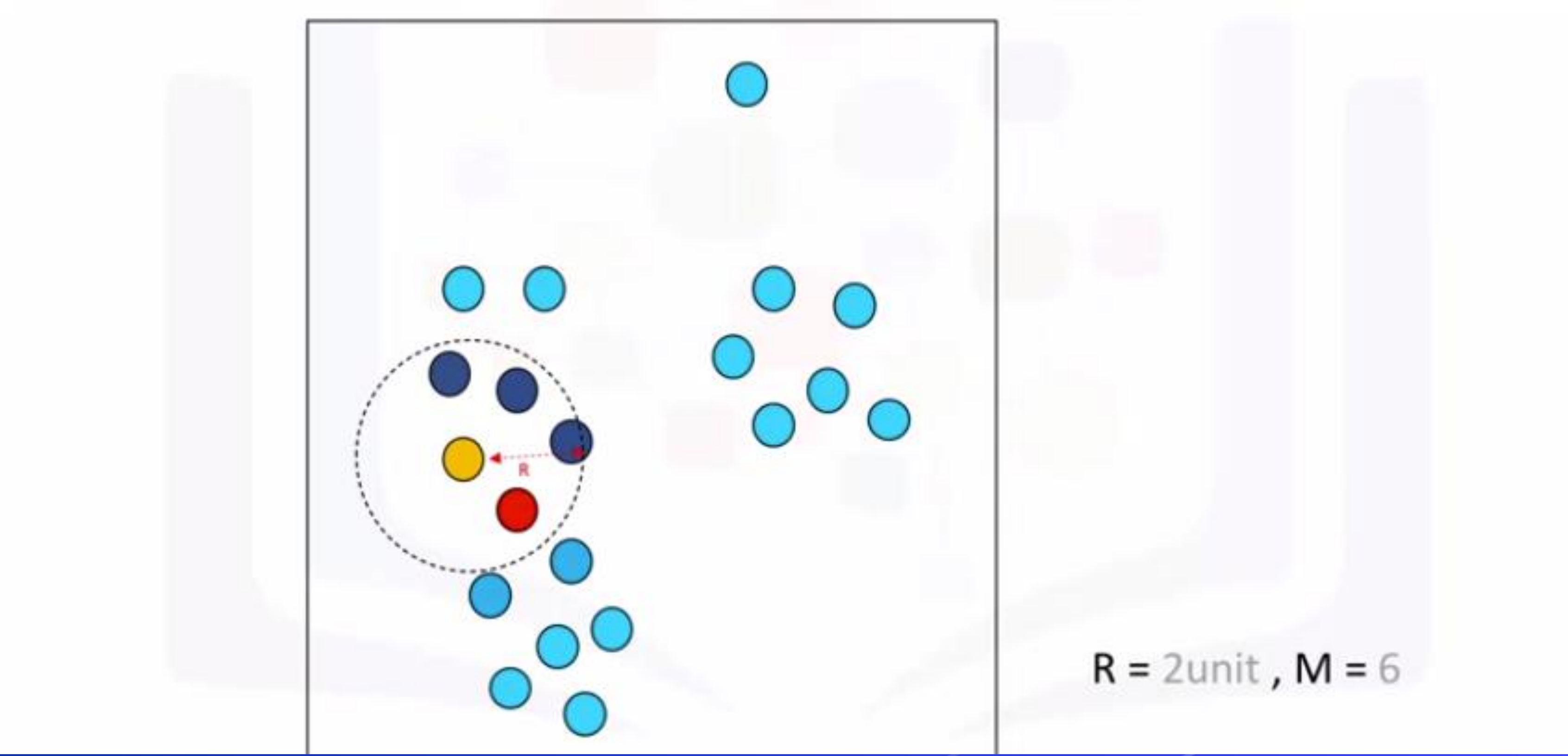
Density Based Clustering - DBSCAN



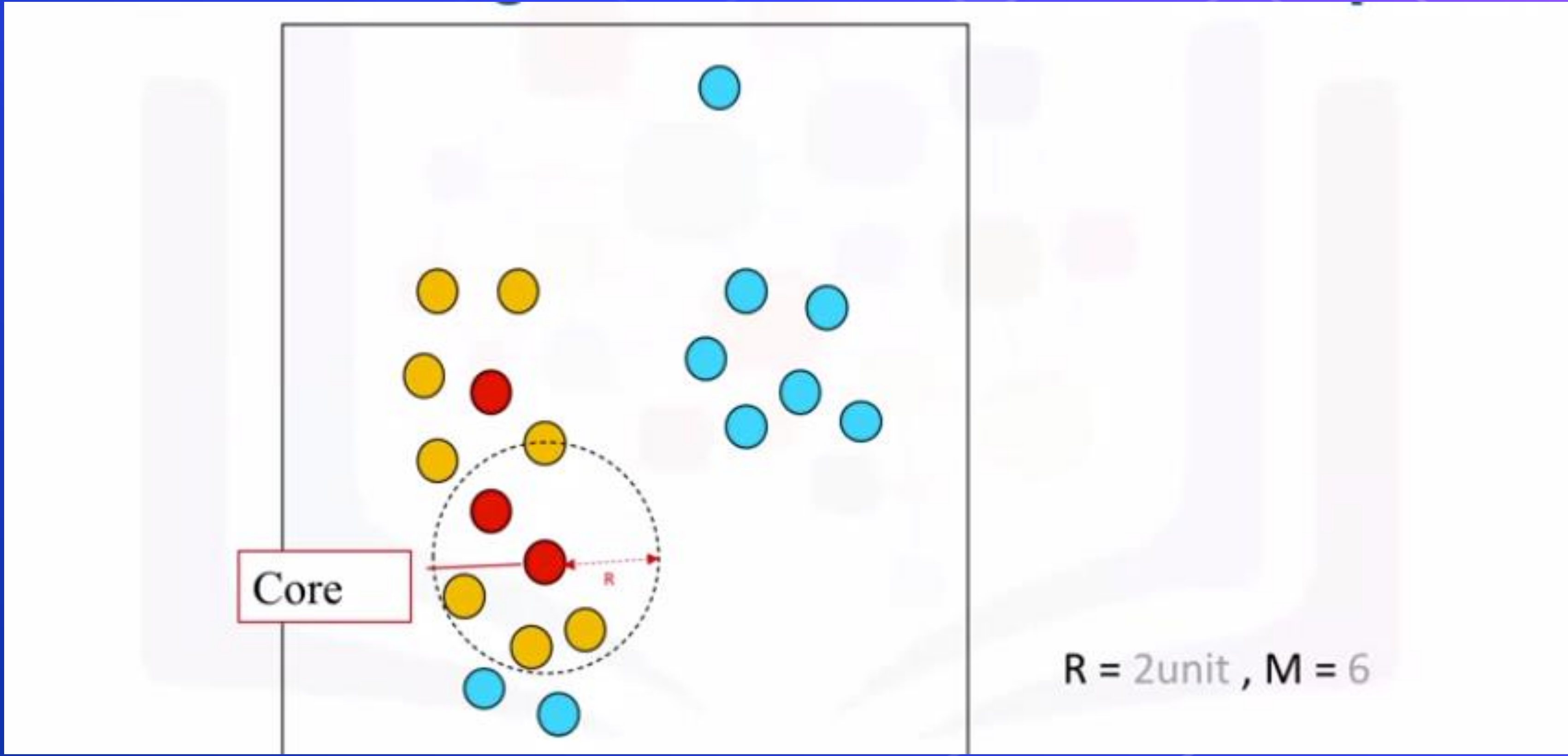
Density Based Clustering – DBSCAN (Core point)



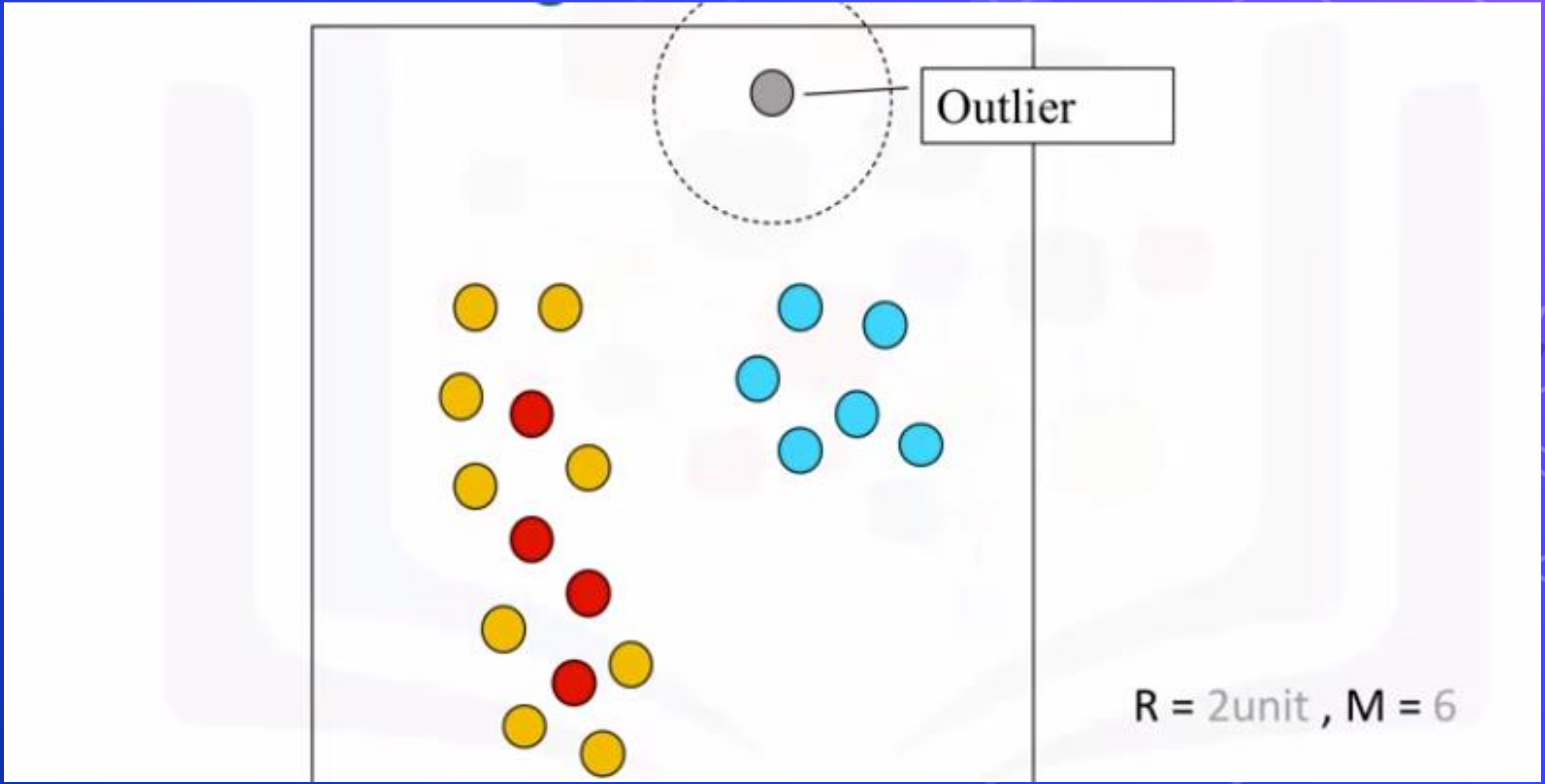
Density Based Clustering – DBSCAN (Border point)



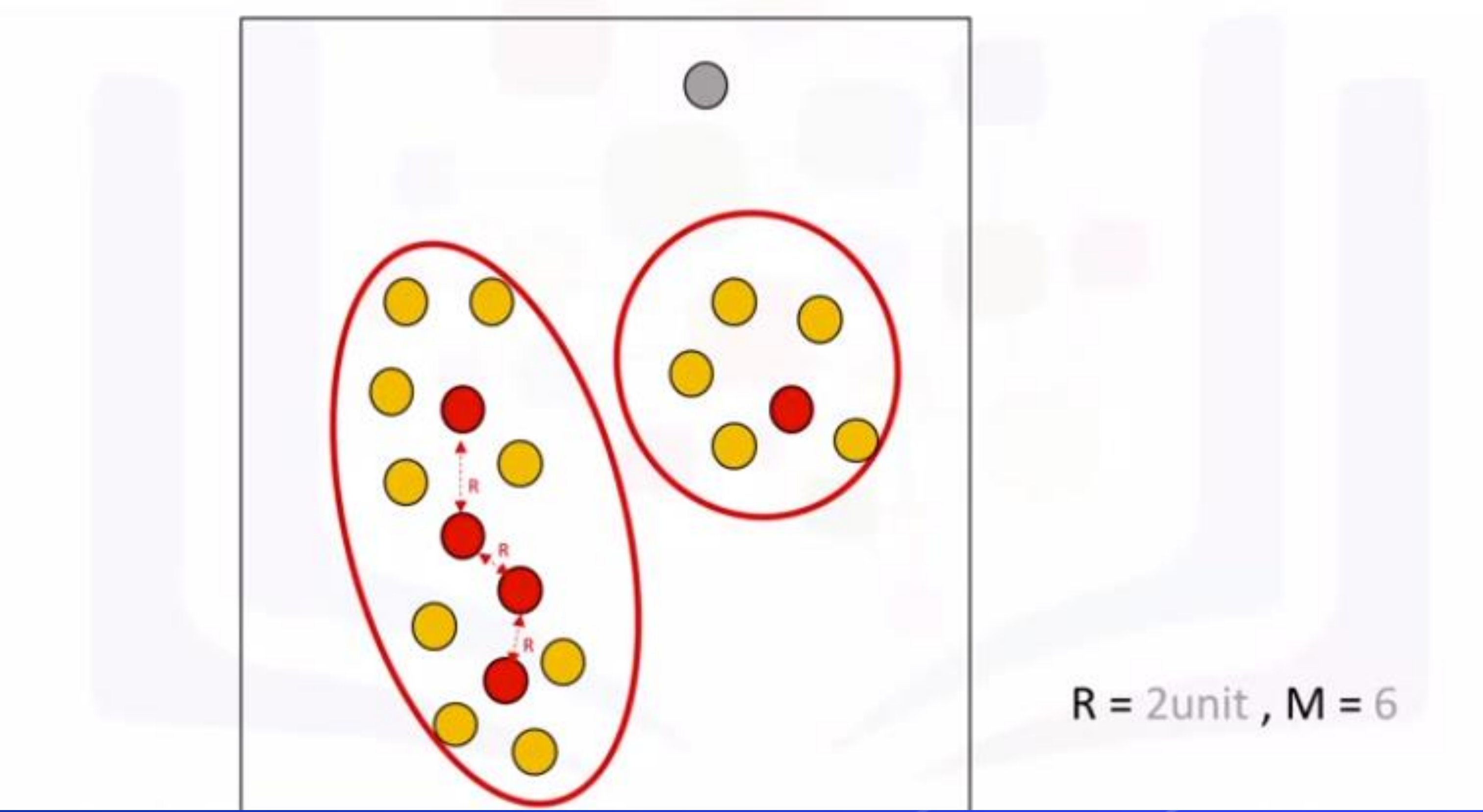
Density Based Clustering - DBSCAN



Density Based Clustering - DBSCAN



Density Based Clustering - DBSCAN

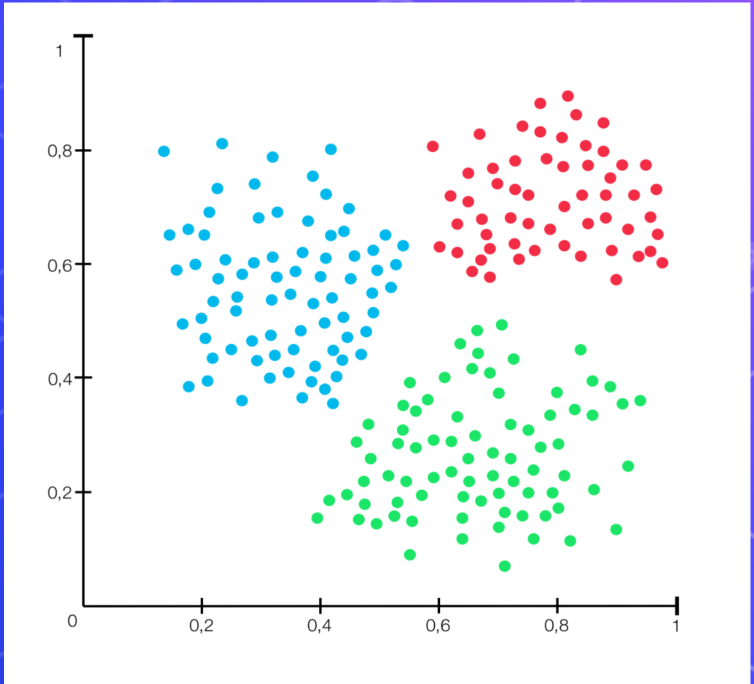


Density Based Clustering - DBSCAN

```python
from sklearn.cluster import DBSCAN

model.DBSCAN(eps=0.3, min_samples=10)
y_labels = model.fit_predict(x)
```

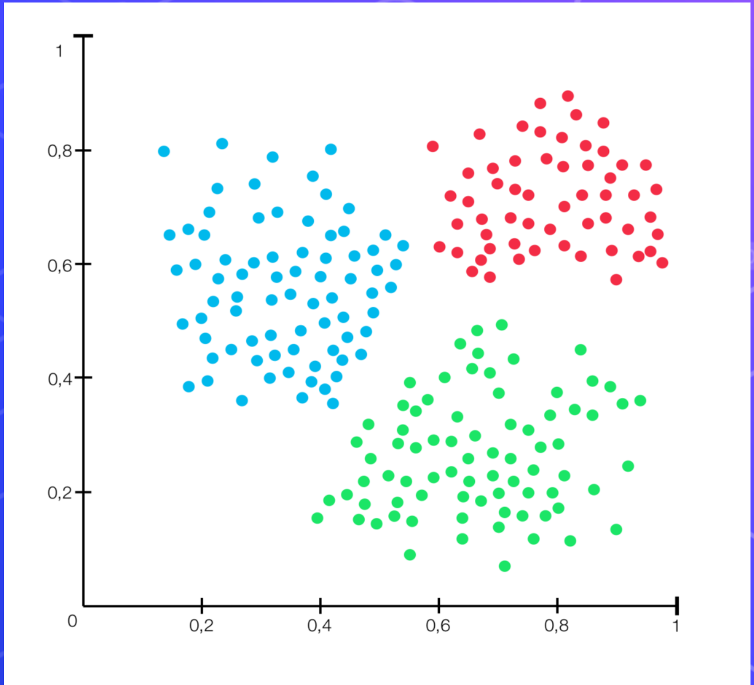
Clustering

- KMeans
- Hierarchical Clustering
- Density Based Clustering – DBSCAN
- <u>**Gaussian Mixture Model – GMM**</u>
- Evaluating Model Performance



Clustering

- KMeans
- Hierarchical Clustering
- Density Based Clustering – DBSCAN
- Gaussian Mixture Model – GMM
- <u>Evaluating Model Performance</u>



Agenda

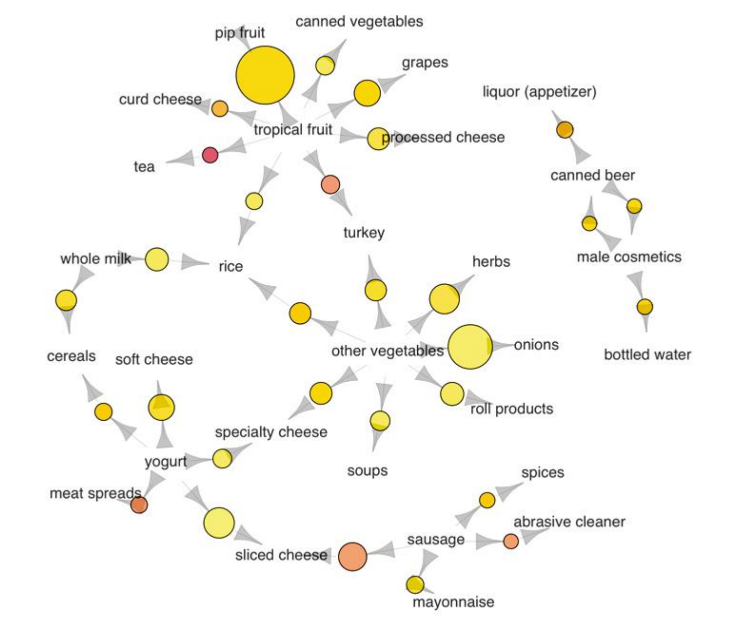
- What is Machine Learning
- Supervised Learning
  - Regression
  - Classification
- **Unsupervised Learning**
  - Clustering
  - **Association rule mining**
  - Dimension Reduction
- Model Selection & Evaluation
  - Cross Validation
  - Hyperparameter Tuning
- Reinforcement Learning

Association rule mining

Association rule mining is a procedure which aims to observe frequently occurring patterns, correlations, or associations from datasets found in various kinds of data.

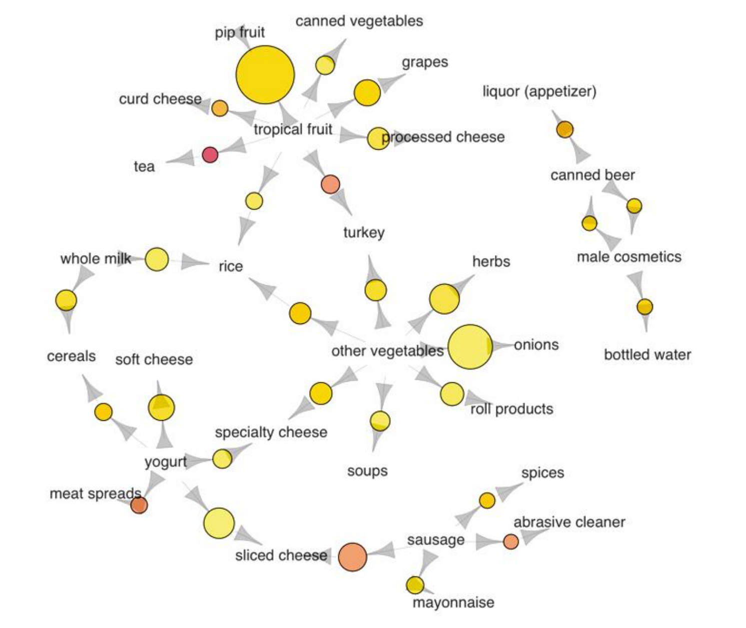
For example,
it can be used to know which products are purchased together and which are most willing to support.

- Market Basket Analysis
- Web usage mining
- Recommendation systems
- …



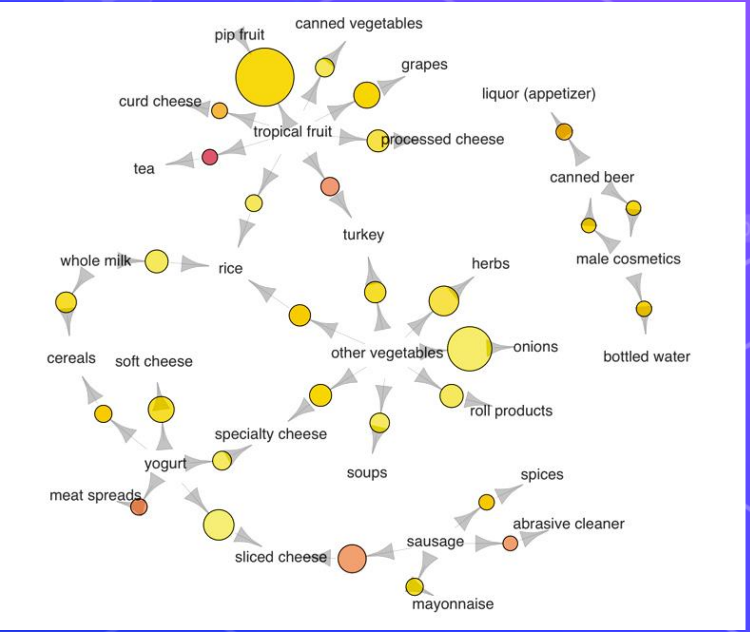
Association rule mining

- Apriori

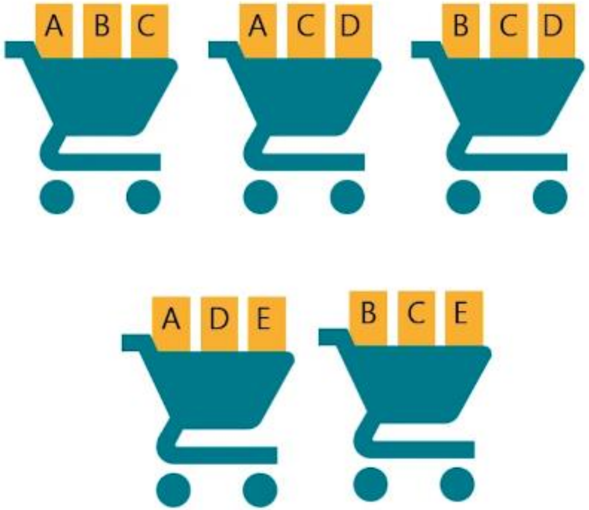


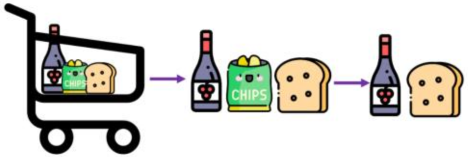
Association rule mining

- [Apriori](#)



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

Apriori

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

Apriori

Association Rule Mining - Subset Creation

- Frequent 3-Item Set = I => {Milk, Bread, Butter}
- Non-Empty subset are
  - {{Milk}, {Bread}, {Butter}, {Milk, Bread}, {Milk, Butter}, {Bread, Butter}}
- How to form Association Rule…?
  - For every non-empty subset S of I, the association rule is,
    - **S →(I-S)**
    - **If support(I) / support(S) >= min_confidence**

Apriori

Association Rule Mining - Subset Creation

- Non-Empty subset are
  - {{Milk}, {Bread}, {Butter}, {Milk, Bread}, {Milk, Butter}, {Bread, Butter}}
  - Min_Support = 30% and Min_Confidence = 60%

- Rule 1: {Milk} → {Bread, Butter} {S=50%, C=66.67%}
  - Support = 6/12 = 50%
  - Confidence = Support (Milk, Bread, Butter)/Support(Milk) = $\frac{6/12}{9/12}$ = 6/9 = 66.67% > 60%
  - Valid

- Rule 2: {Bread} → {Milk, Butter} {S=50%, C=60%}
  - Support = 6/12 = 50%
  - Confidence = Support (Milk, Bread, Butter)/Support(Bread) = 6/10 = 60% >= 60%
  - Valid

Apriori

Association Rule Mining - Subset Creation

- Non-Empty subset are
  - {{Milk}, {Bread}, {Butter}, {Milk, Bread}, {Milk, Butter}, {Bread, Butter}}
  - Min_Support = 30% and Min_Confidence = 60%

- Rule 3: {Butter} → {Milk, Bread} {S=50%, C=60%}
  - Support = 6/12 = 50%
  - Confidence = Support (Milk, Bread, Butter)/Support(Butter) = 6/10 = 60%>=60
  - Valid

- Rule 4: {Milk, Bread} → {Butter} {S=50%, C=85.7%}
  - Support = 6/12 = 50%
  - Confidence = Support (Milk, Bread, Butter)/Support(Milk, Bread) = 6/7 = 85.7% > 60%
  - Valid

Apriori

Association Rule Mining - Subset Creation

- Non-Empty subset are
  - {{Milk}, {Bread}, {Butter}, {Milk, Bread}, {Milk, Butter}, {Bread, Butter}}
  - Min_Support = 30% and Min_Confidence = 60%

- Rule 5: {Milk, Butter} → {Bread} {S=50%, C=85.7%}
  - Support = 6/12 = 50%
  - Confidence = Support (Milk, Bread, Butter)/Support(Milk, Butter) = 6/7 = 85.7% >= 60%
  - Valid

- Rule 6: {Bread, Butter} → {Milk} {S=50%, C=66.67%}
  - Support = 6/12 = 50%
  - Confidence = Support (Milk, Bread, Butter)/Support(Bread, Butter) = 6/9 = 66.67%>=60
  - Valid

Apriori

```python
from apyori import apriori

records = []
for i in range(0, rows):
    records.append([str(df.values[i,j]) for j in range(0, columns)])

association_rules = apriori(records)
association_results = list(association_rules)
```

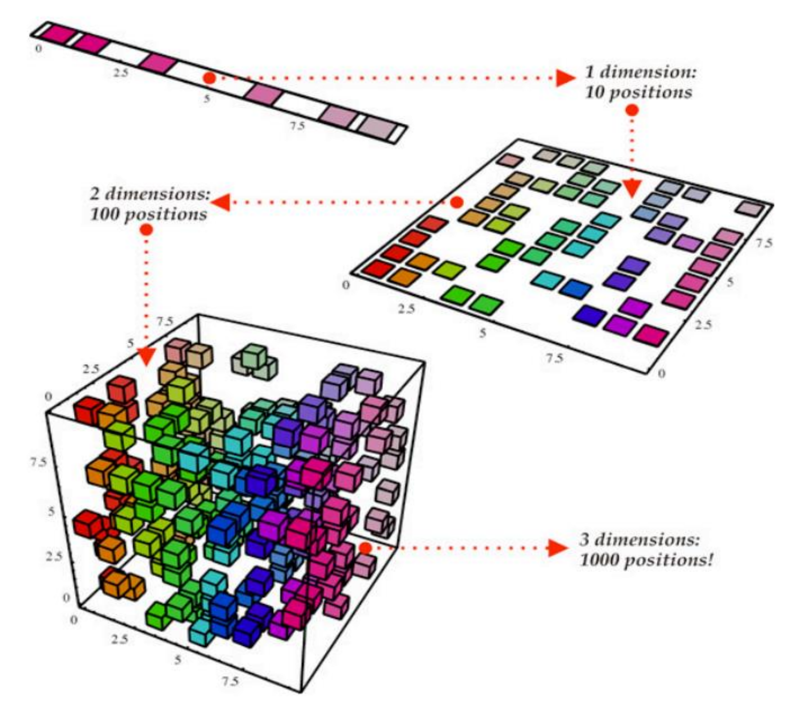
Agenda

- What is Machine Learning
- Supervised Learning
  - Regression
  - Classification
- **Unsupervised Learning**
  - Clustering
  - Association rule mining
  - **Dimension Reduction**
- Model Selection & Evaluation
  - Cross Validation
  - Hyperparameter Tuning
- Reinforcement Learning

Dimension Reduction

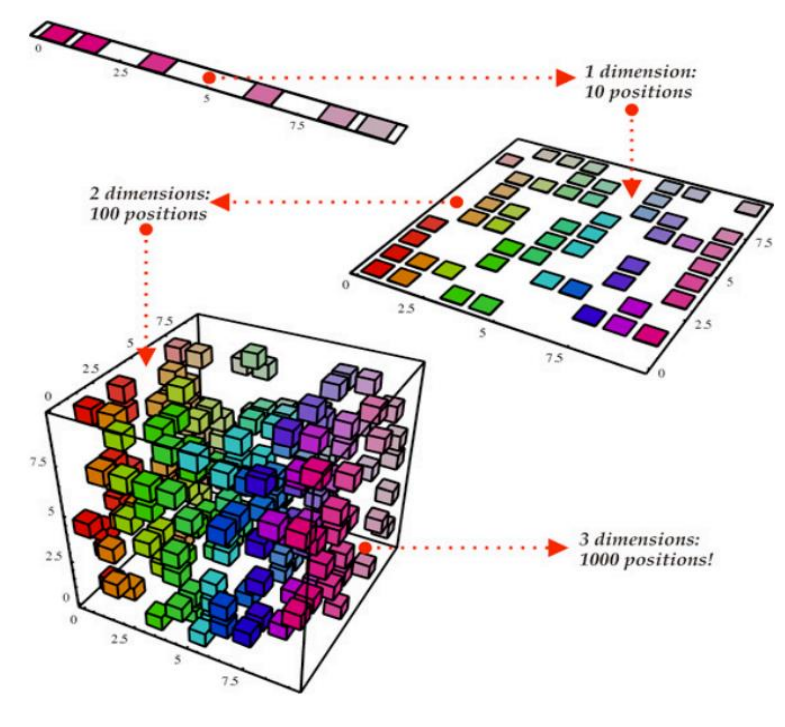
is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data

For example,
it can be used as data compression to reduce size of data for easy processing.



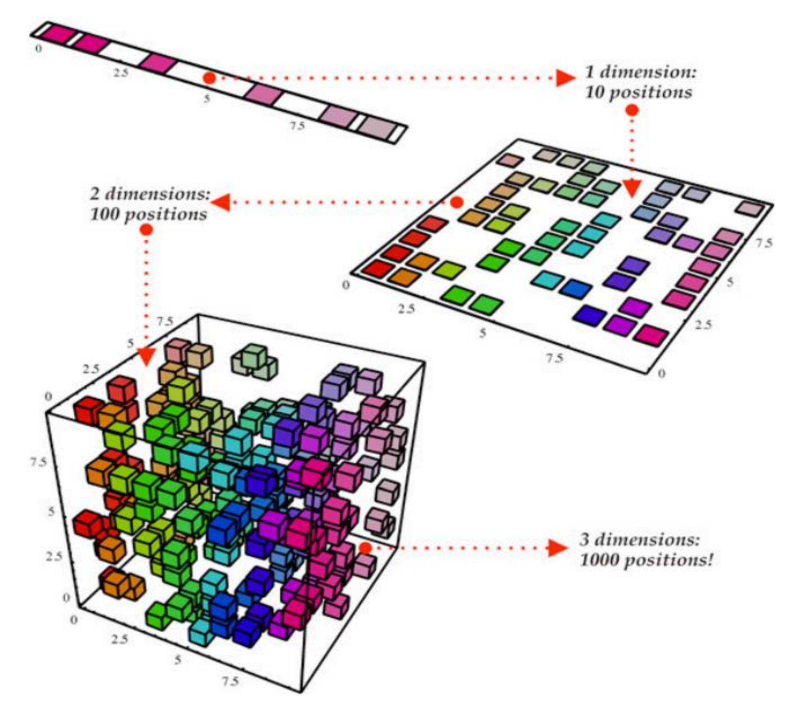
Dimension Reduction

- PCA
- Kernel PCA



Dimension Reduction

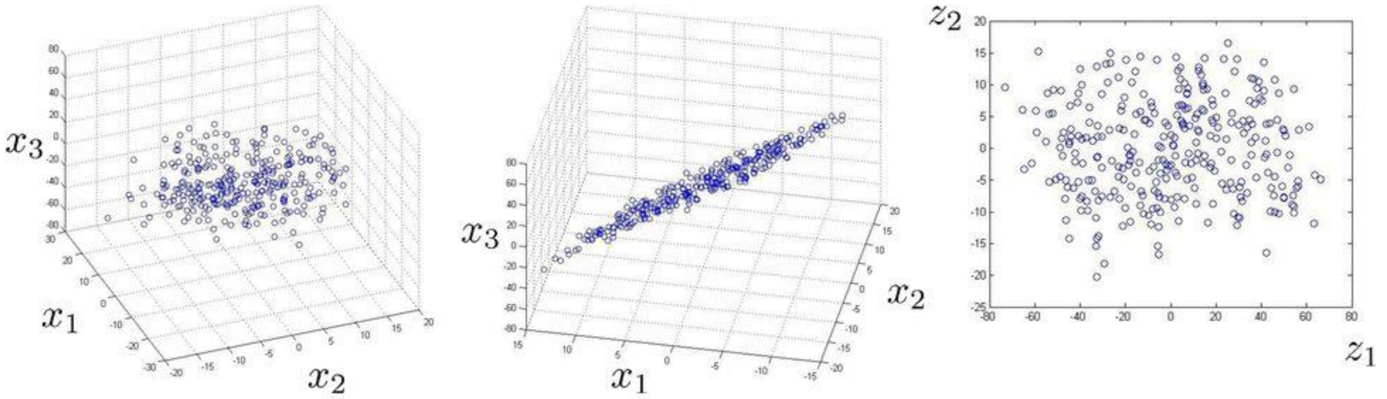
- PCA
- Kernel PCA



PCA

Data Compression

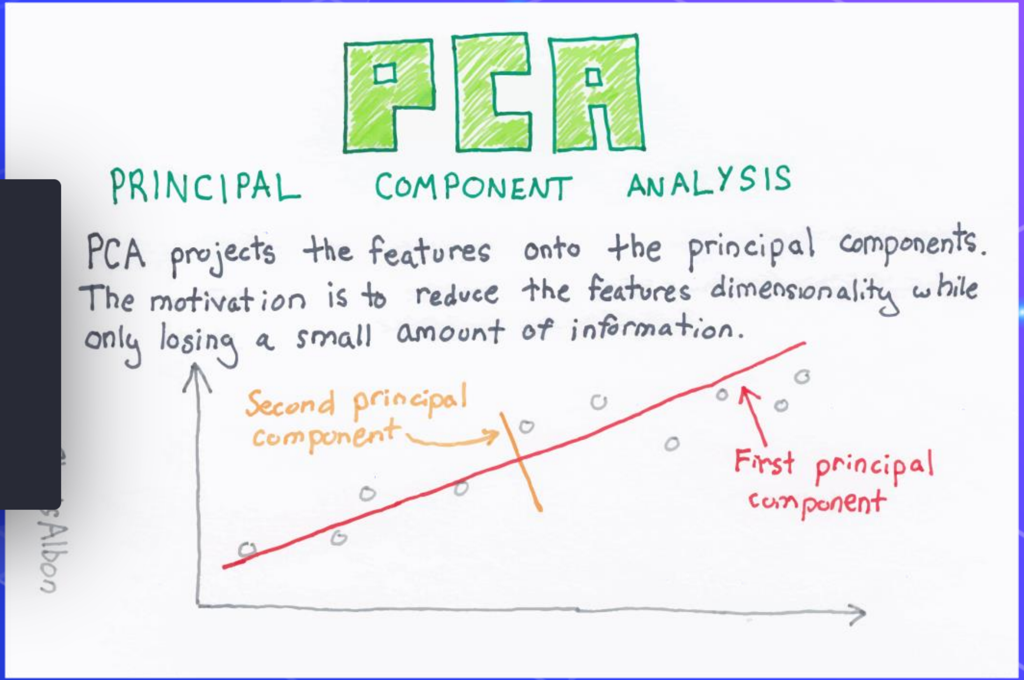
Reduce data from 3D to 2D



Andrew Ng

PCA

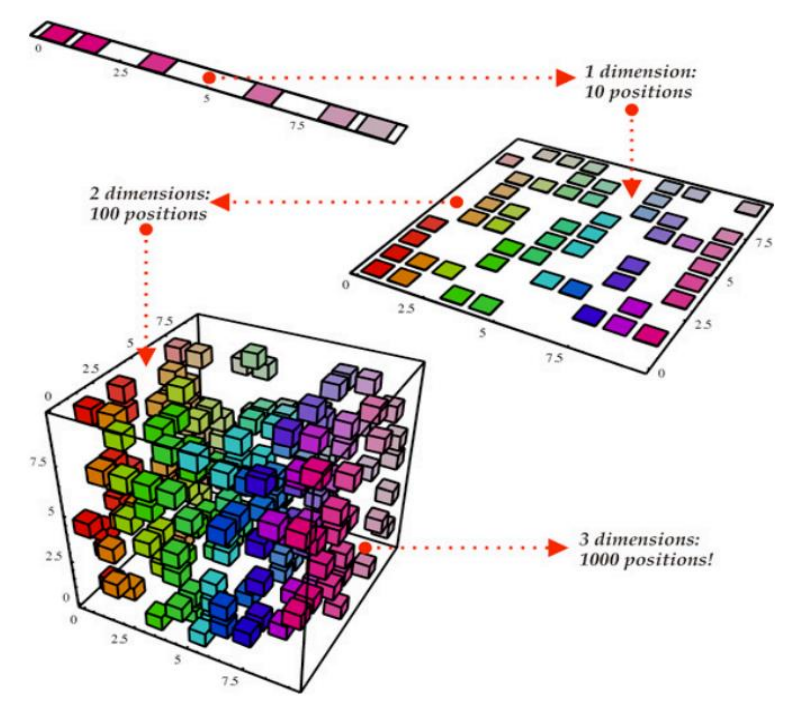
```python
from sklearn.decomposition import PCA

X = # feature vector

pca = PCA(0.9)
X = pca.fit_transform(X)
pca.explained_variance_ratio_
```



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

Dimension Reduction

- PCA
- Kernel PCA



Agenda

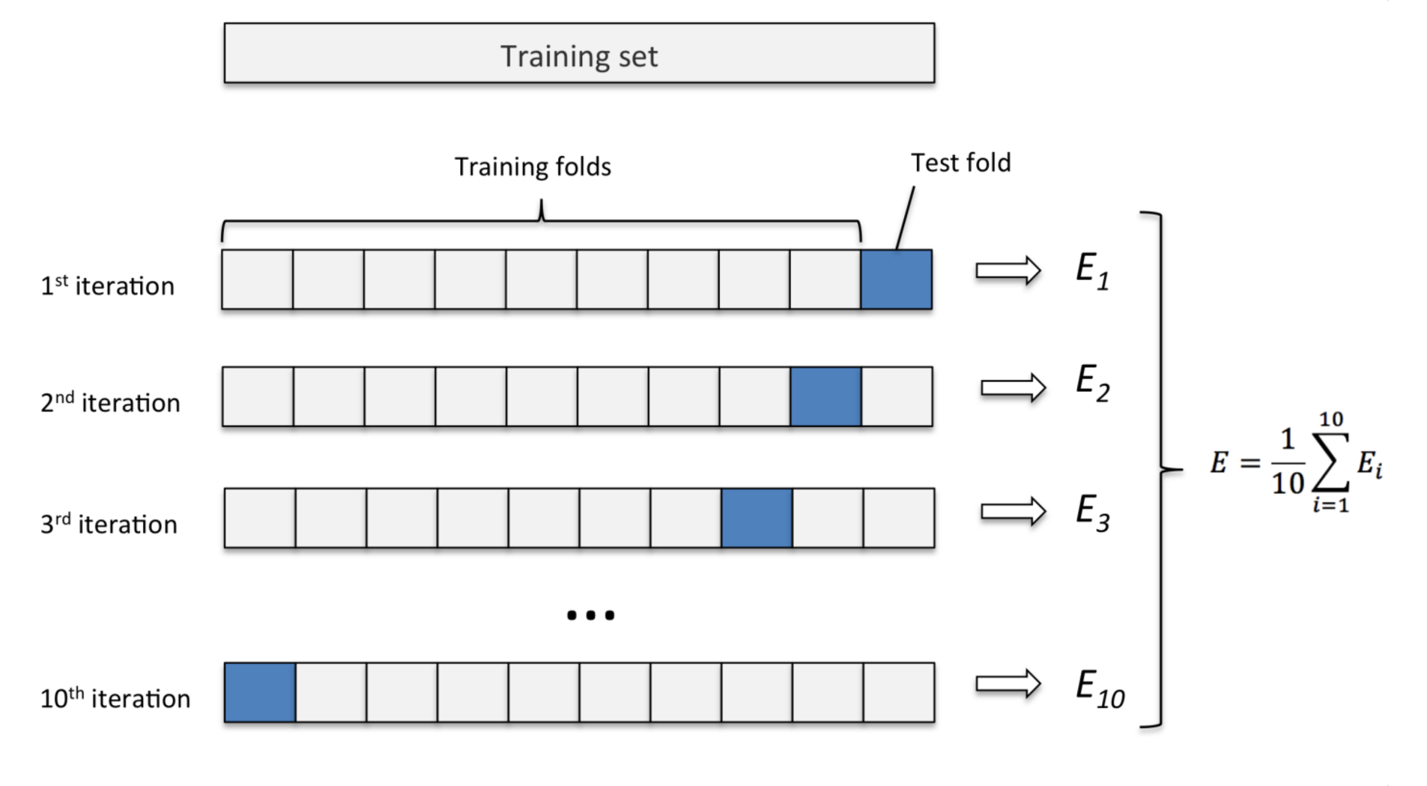
- What is Machine Learning
- Supervised Learning
  - Regression
  - Classification
- Unsupervised Learning
  - Clustering
  - Association rule mining
  - Dimension Reduction
- **Model Selection & Evaluation**
  - **Cross Validation**
  - Hyperparameter Tuning
- Reinforcement Learning

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

```python
from sklearn.svm import SVC
from sklearn.model_selection import cross_validate

svc = SCV()
cv_results_svc = cross_validate(svc, X, Y, cv=10, return_train_score=True)

cv_results_svc['test_score'].mean()
"""
0.8036085674097743
"""
```

Agenda

- What is Machine Learning
- Supervised Learning
  - Regression
  - Classification
- Unsupervised Learning
  - Clustering
  - Association rule mining
  - Dimension Reduction
- **Model Selection & Evaluation**
  - Cross Validation
  - **Hyperparameter Tuning**
- Reinforcement Learning

Hyperparameter Tuning

- Grid Search
- Randomized Search

```python
from sklearn.model_selection import GridSearchCV


params = [
        {'C':[1, 10, 100], 'kernel':['linear', 'sigmoid', 'poly']},
        {'C':[1, 10, 100], 'kernel':['rbf'], 'gamma':[0.5, 0.6, 0.7, 0.1, 0.01, 0.01]}
         ]

grid_search = GridSearchCV(estimator=model,
                           param_grid=params,
                           scoring='accuracy',
                           cv=10)
grid_search.fit(x_train, y_train)


print(grid_search.best_params_)
print(grid_search.best_params_)
```

Hyperparameter Tuning

- Grid Search
- <u>Randomized Search</u>

```python
from sklearn.model_selection import RandomizedSearchCV


params = [
        {'C':[1, 10, 100], 'kernel':['linear', 'sigmoid', 'poly']},
        {'C':[1, 10, 100], 'kernel':['rbf'], 'gamma':[0.5, 0.6, 0.7, 0.1, 0.01, 0.01]}
         ]

randomized_search = RandomizedSearchCV(estimator=model,
                                       param_grid=params,
                                       scoring='accuracy',
                                       cv=10)
randomized_search.fit(x_train, y_train)


print(randomized_search.best_params_)
print(randomized_search.best_params_)
```

Agenda

- What is Machine Learning
- Supervised Learning
  - Regression
  - Classification
- Unsupervised Learning
  - Clustering
  - Association rule mining
  - Dimension Reduction
- Model Selection & Evaluation
  - Cross Validation
  - Hyperparameter Tuning
- <u>**Reinforcement Learning**</u>

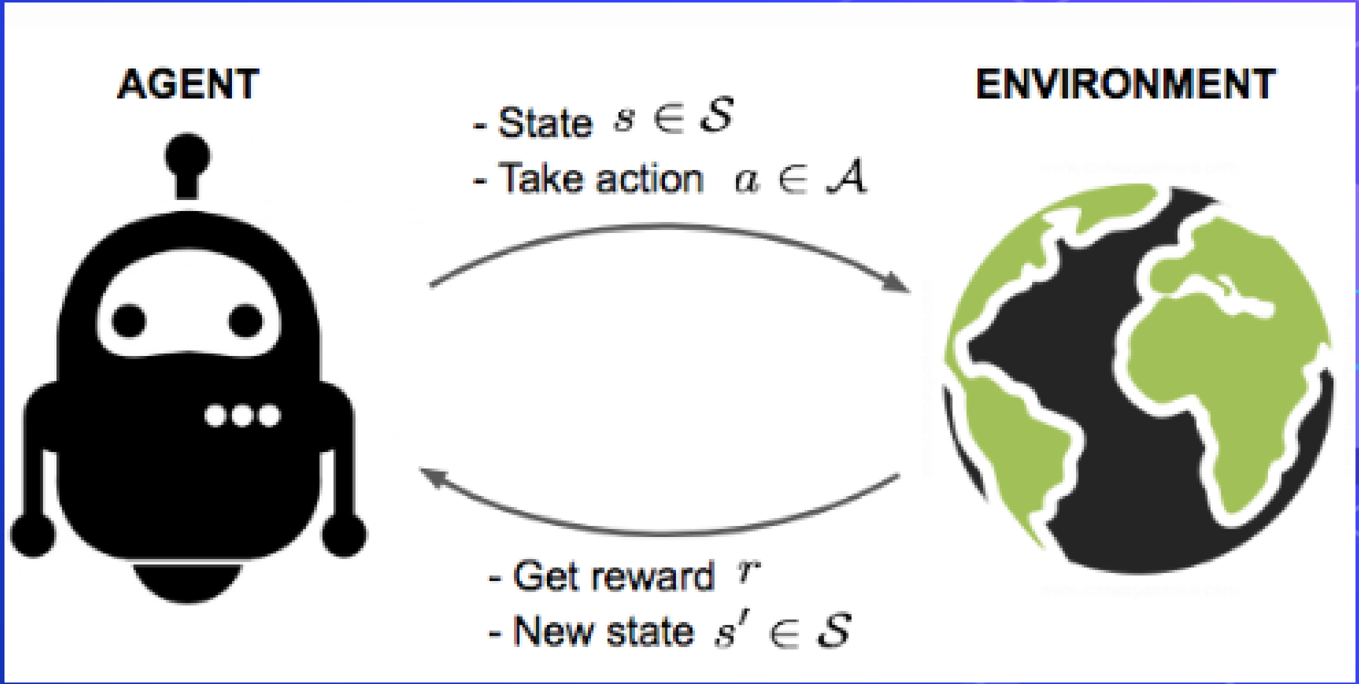
Reinforcement Learning

- 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

- <u>What is Reinforcement Learning.</u>
- 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 >_

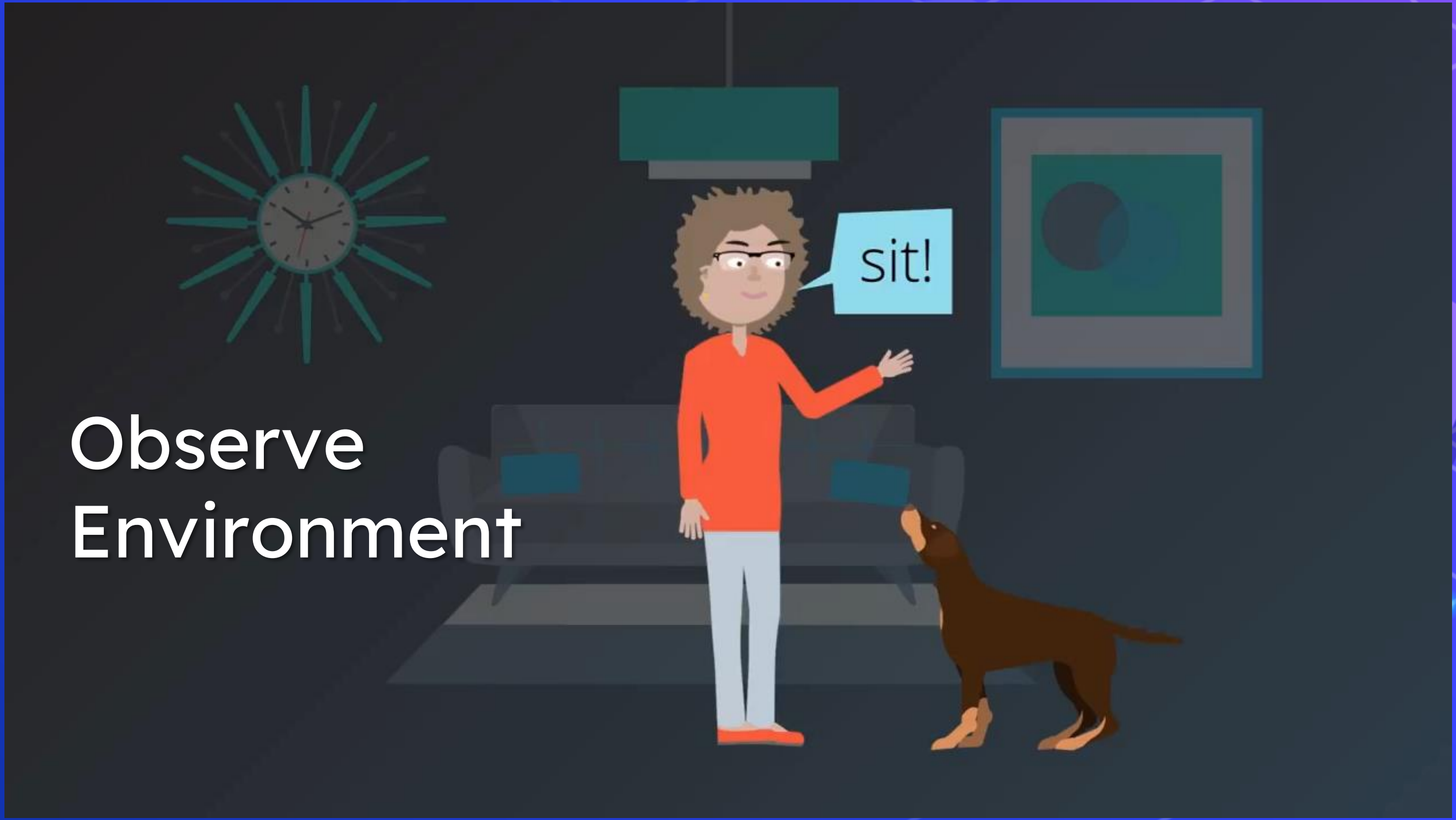
What is RL ?


AGENT
ENVIRONMENT
- State $s \in \mathcal{S}$
- Take action $a \in \mathcal{A}$
- Get reward r
- New state $s' \in \mathcal{S}$

What is RL ?

3azooz (Agent)

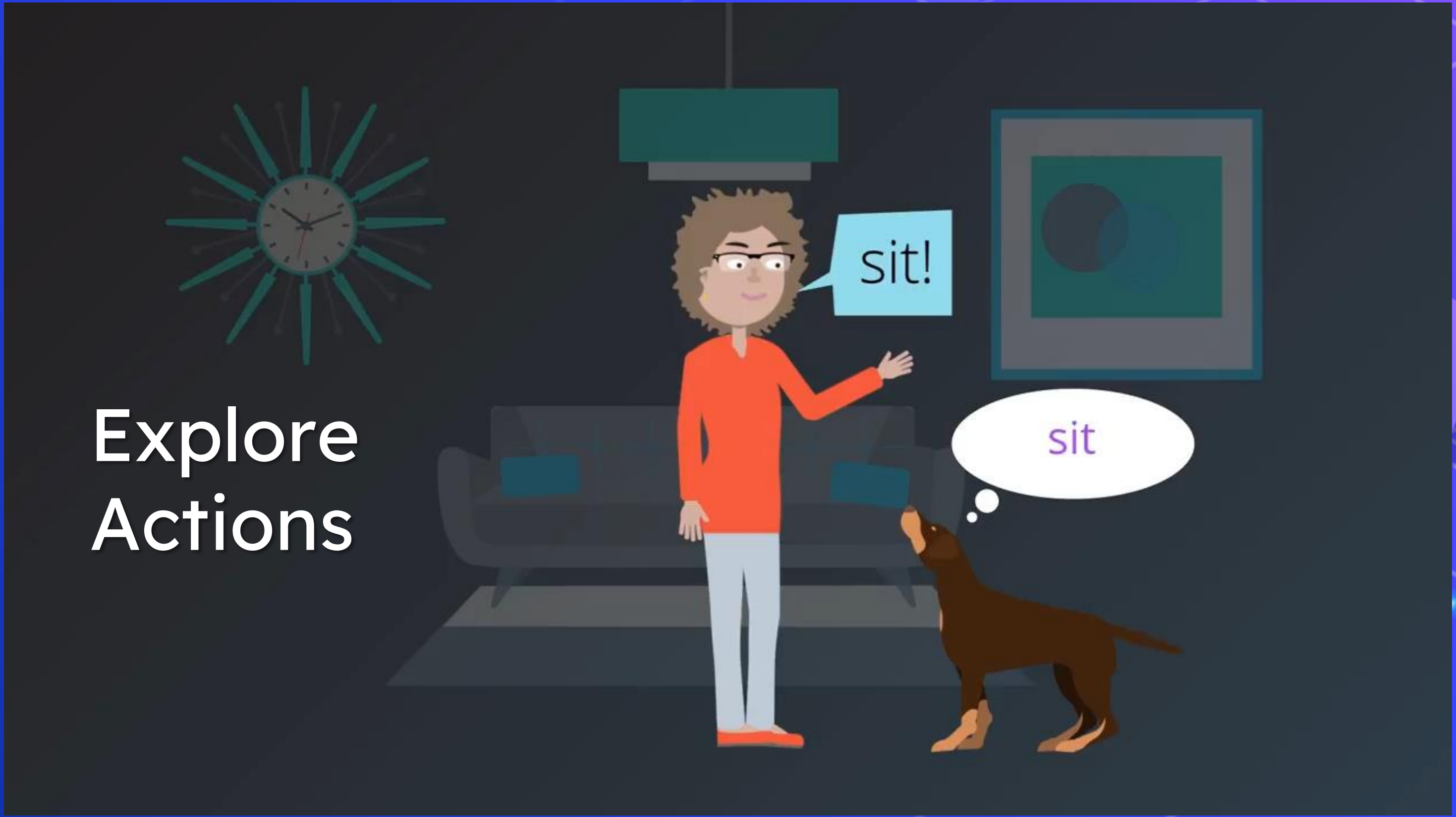
What is RL ?



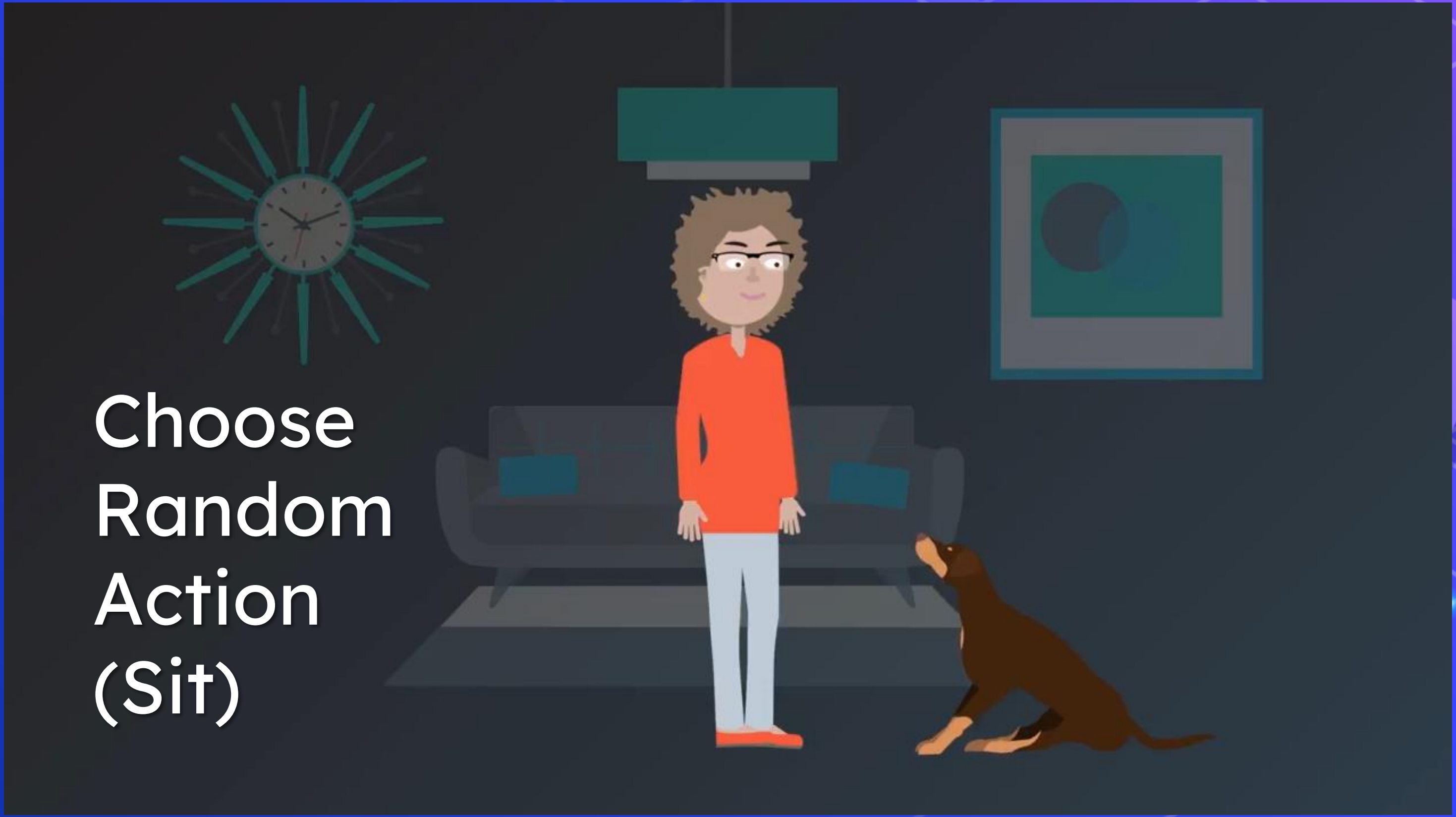
What is RL ?

What is RL ?

What is RL ?



What is RL ?

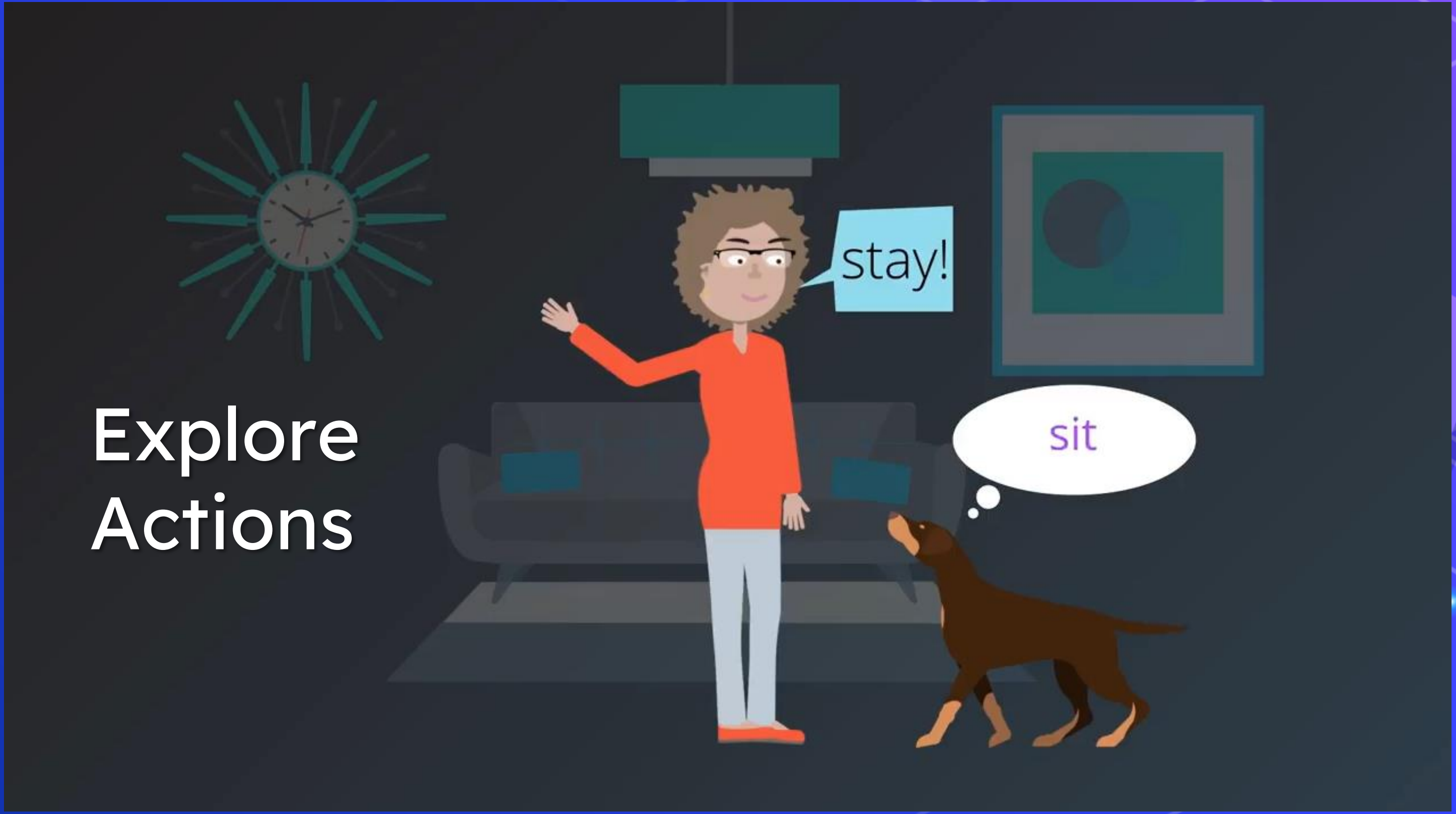


What is RL ?

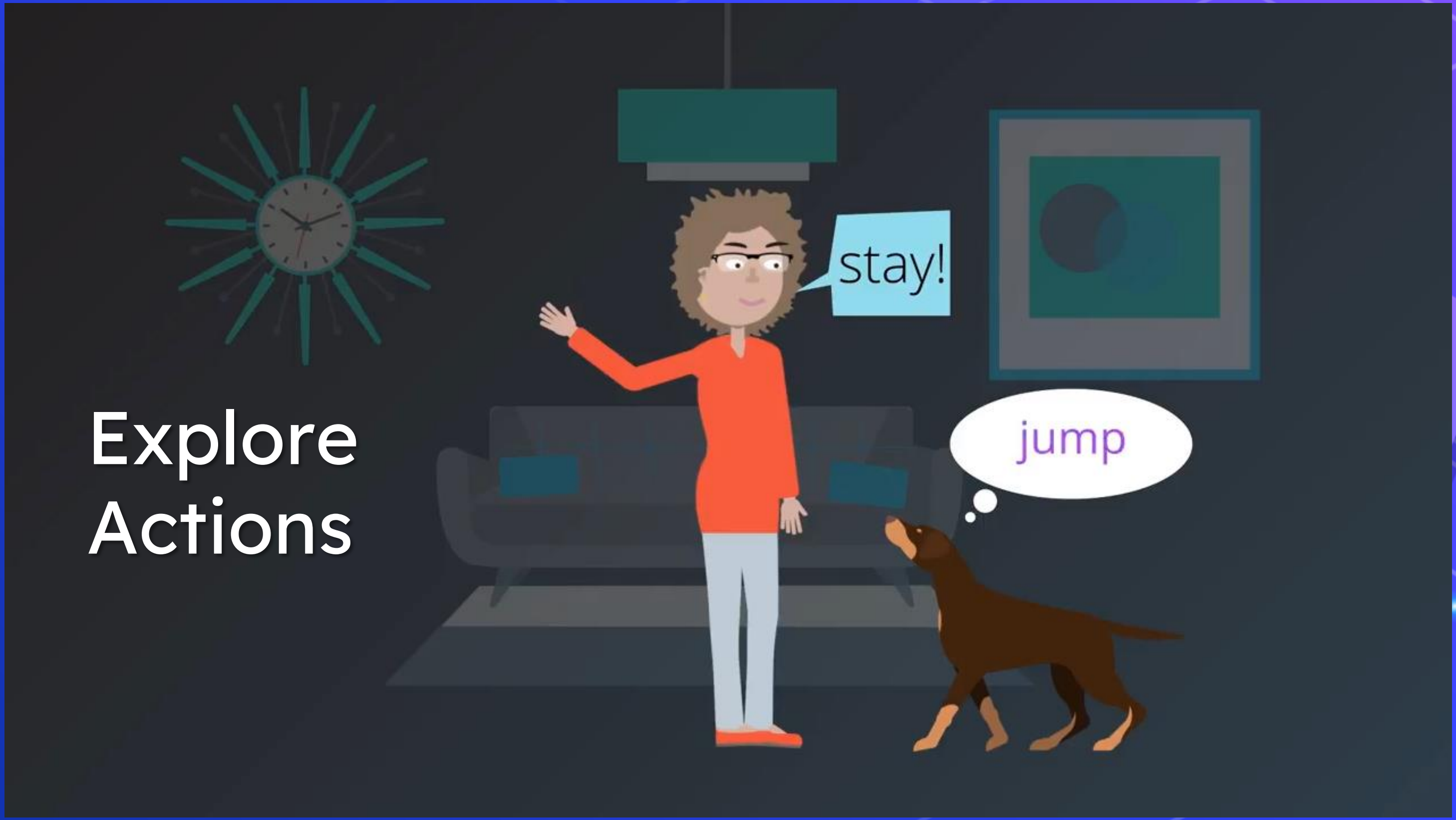
What is RL ?

What is RL ?

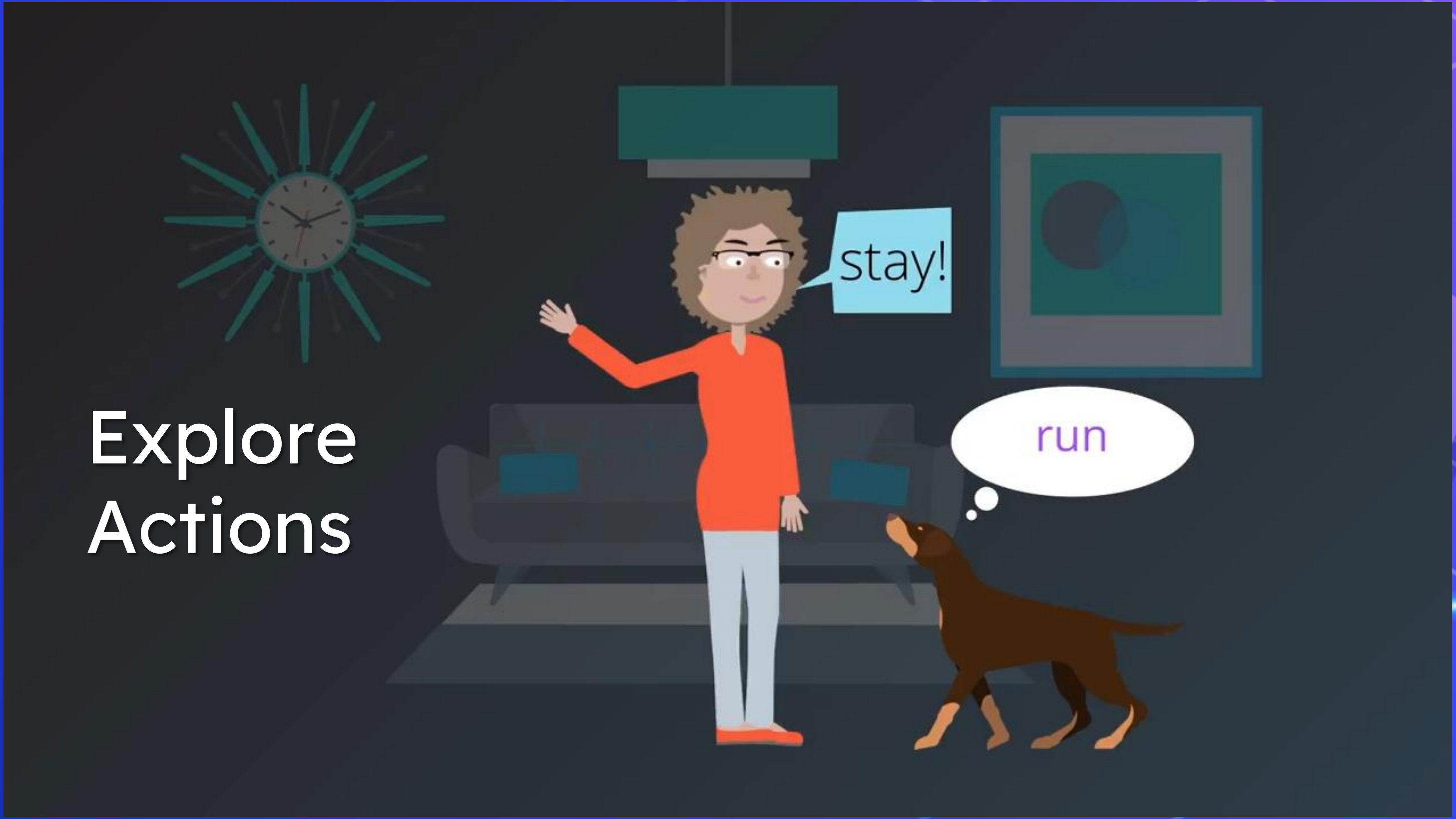
What is RL ?



What is RL ?



What is RL ?



What is RL ?

What is RL ?

What is RL ?

What is RL ?

What is RL ?

Again and again for a lot of times (Episodes)

Reinforcement Learning

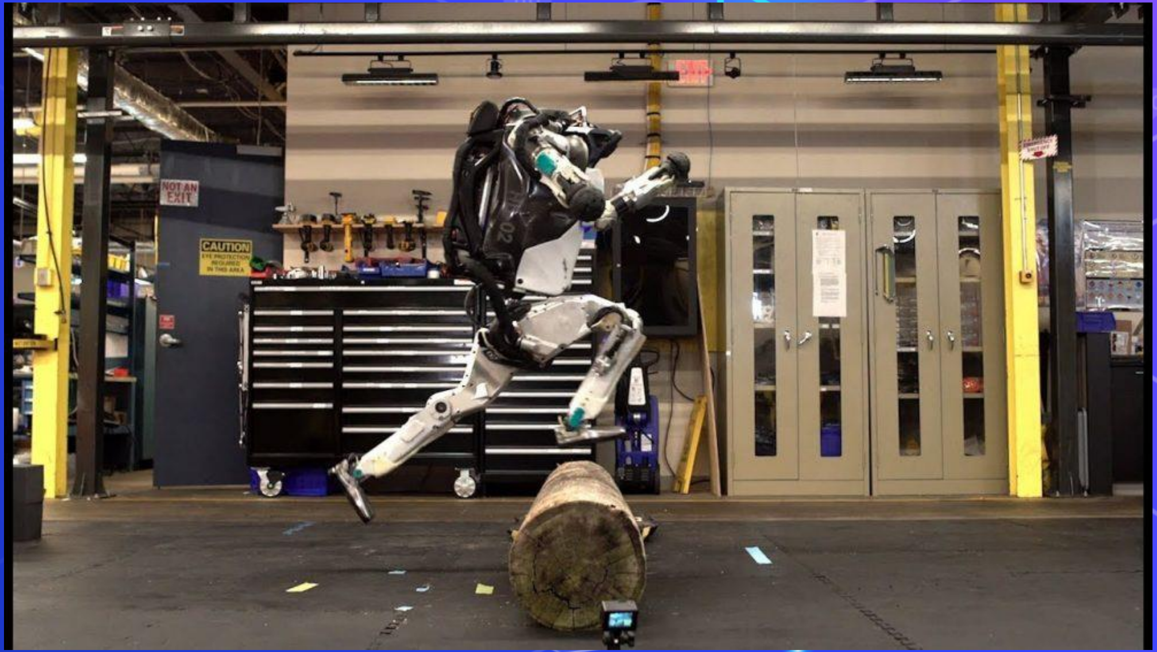
- What is Reinforcement Learning.
- <u>Applications of Reinforcement Learning.</u>
- Why RL is Freaky.
- Markov Decision Process (MDP).
- MDP Examples.
- MDP Solution.
- Q-Learning Algorithm.
- Open AI Gym environment.
- Coding time >_

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.
- ...

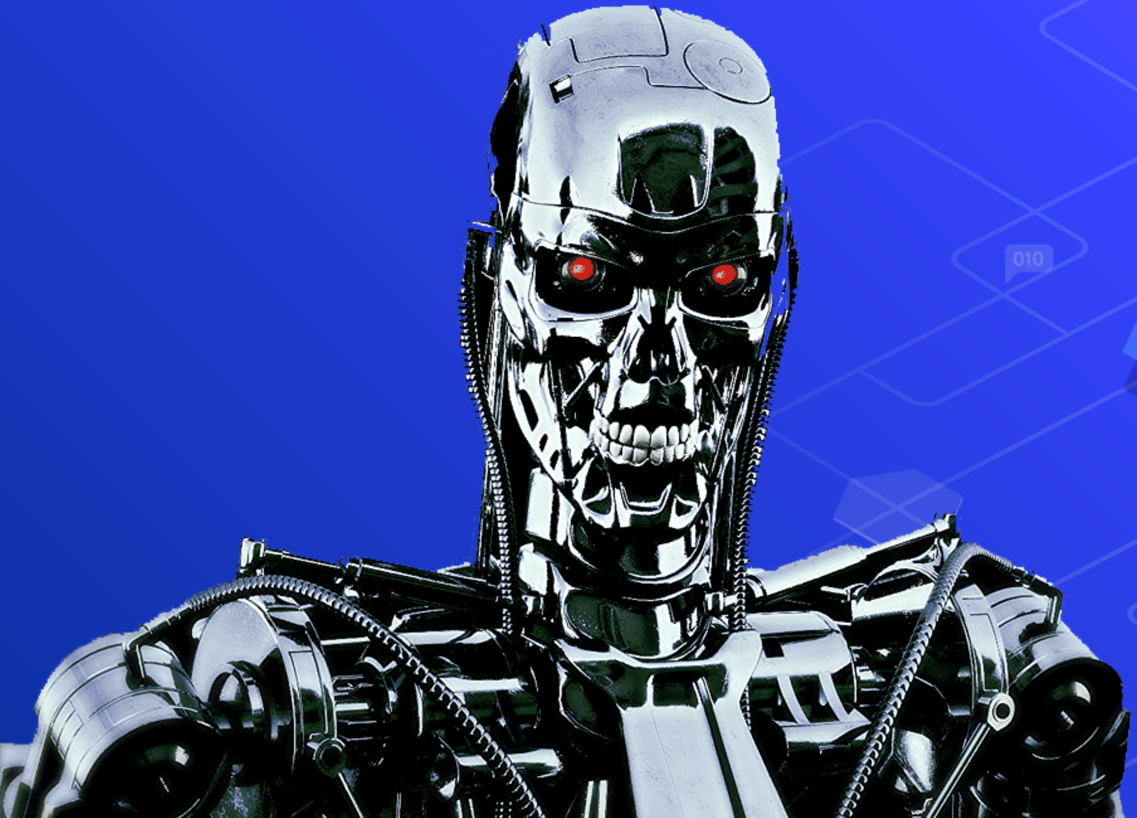
Reinforcement Learning Apps

- 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 ?

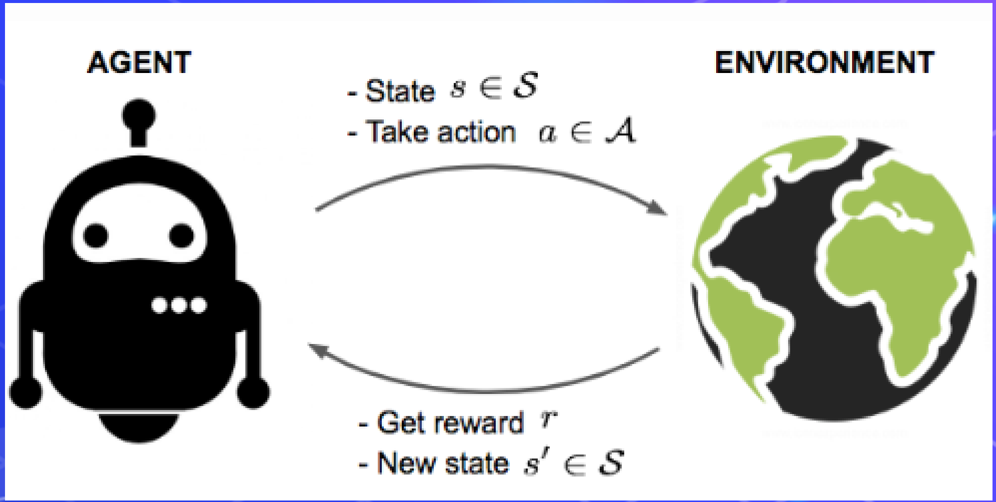
Why RL is Creepy ?

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).*

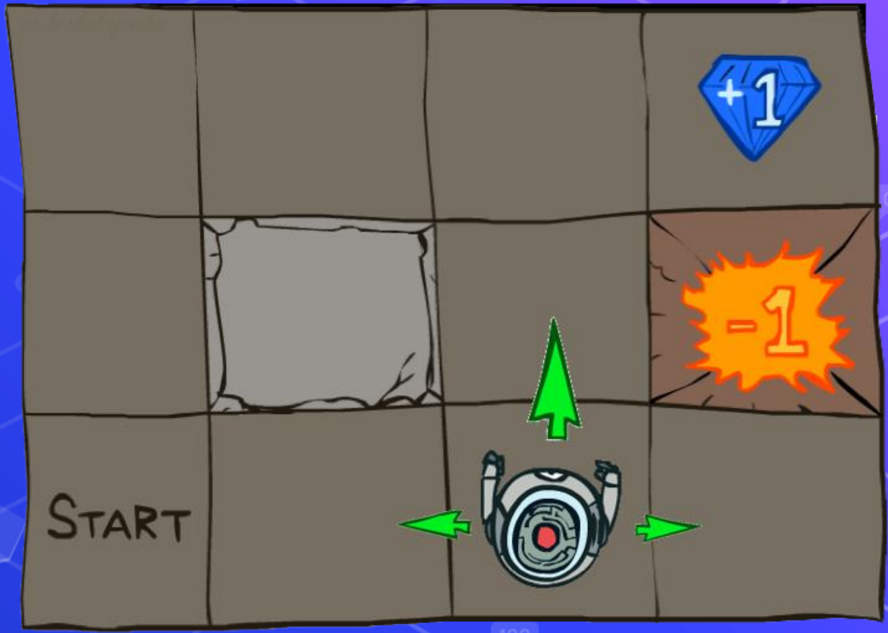


Reinforcement Learning

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

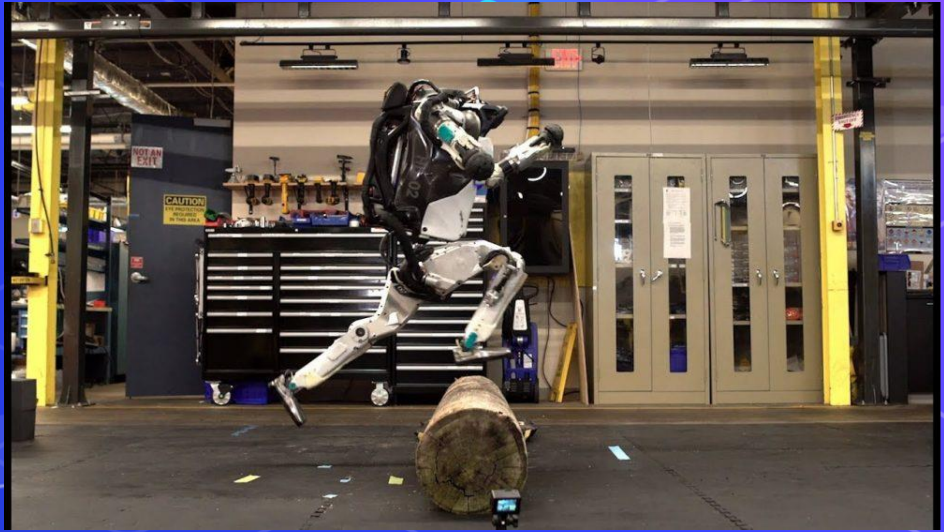
Markov Decision process (MDP) - Maze

- 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.

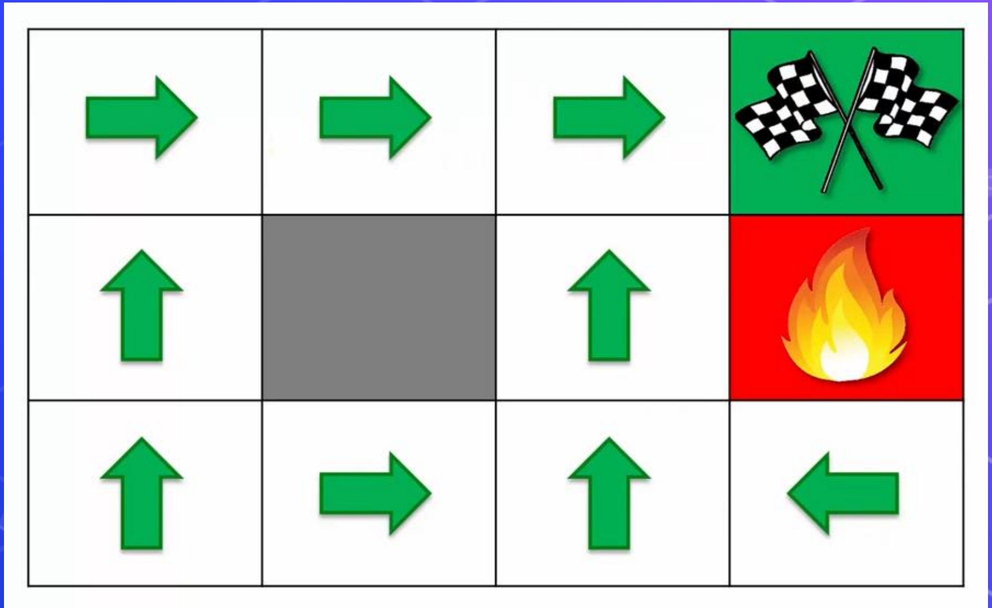
MDP Solution.

- <u>Policy.</u>
- 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.
- <u>Exploration vs Exploitation.</u>
- Reward Function.
- Discounted Reward Factor.
- Q-Values.

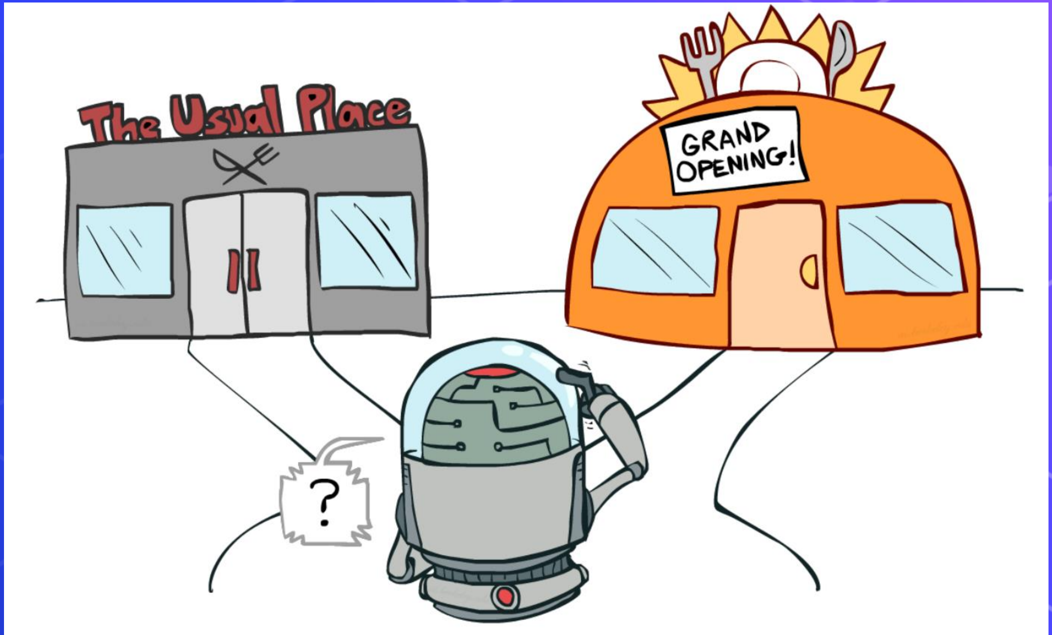
Exploration vs Exploitation

Exploration

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

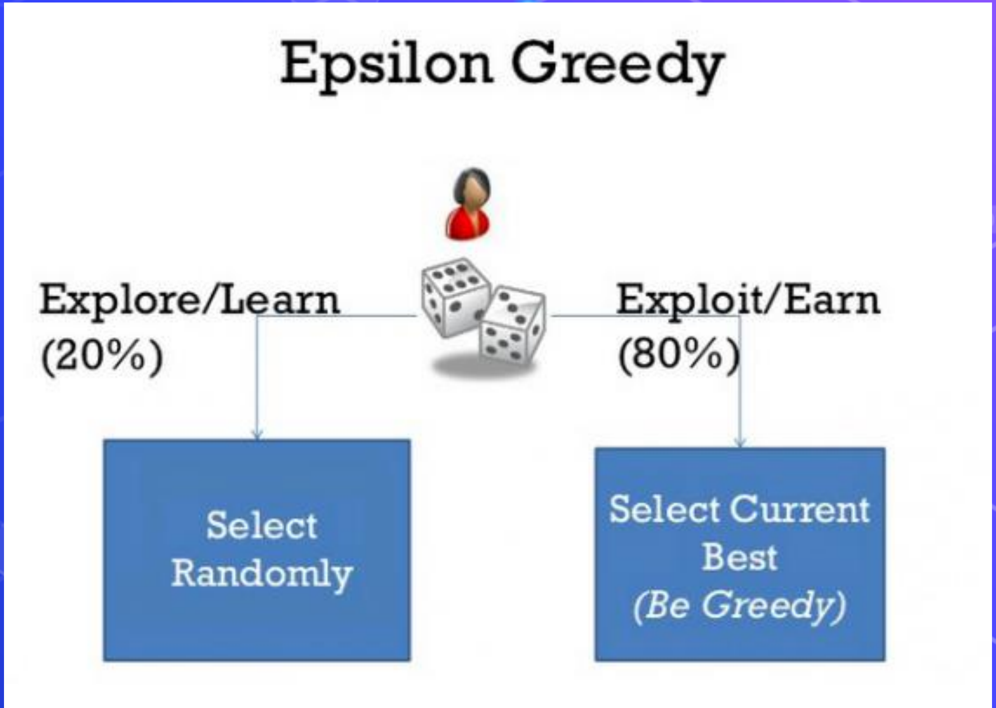
Exploitation

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



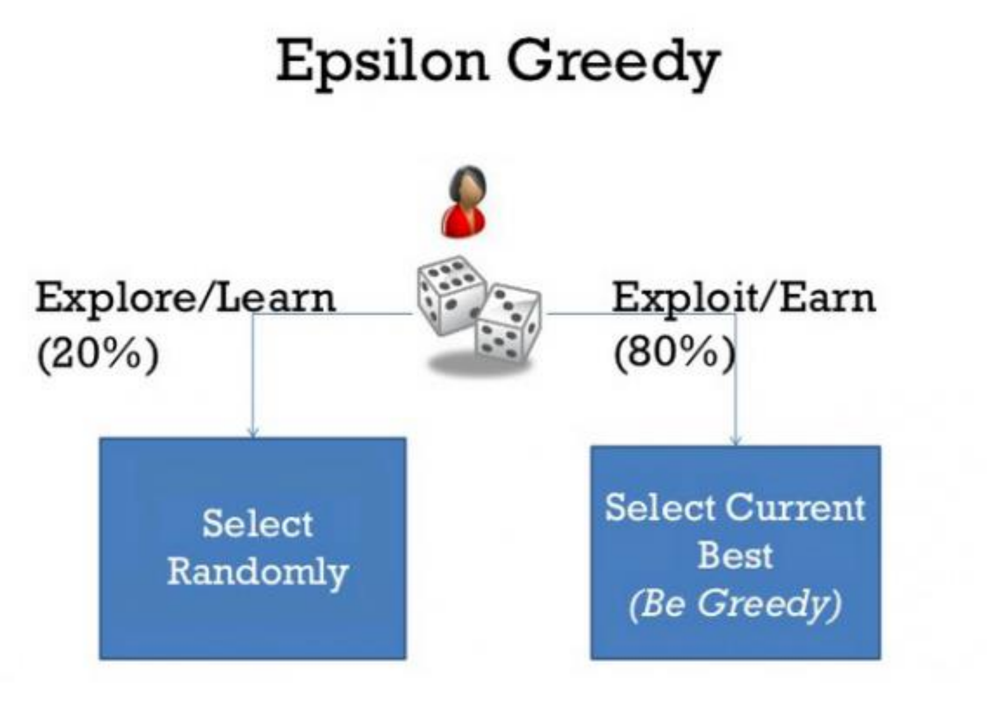
Exploration vs Exploitation (Epsilon-Greedy algorithm)

- Epsilon = 0.2 (20%) having random action.
- (1-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.

```
if random_num > epsilon:
    # choose action via exploitation
else:
    # choose action via exploration
```

MDP Solution

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

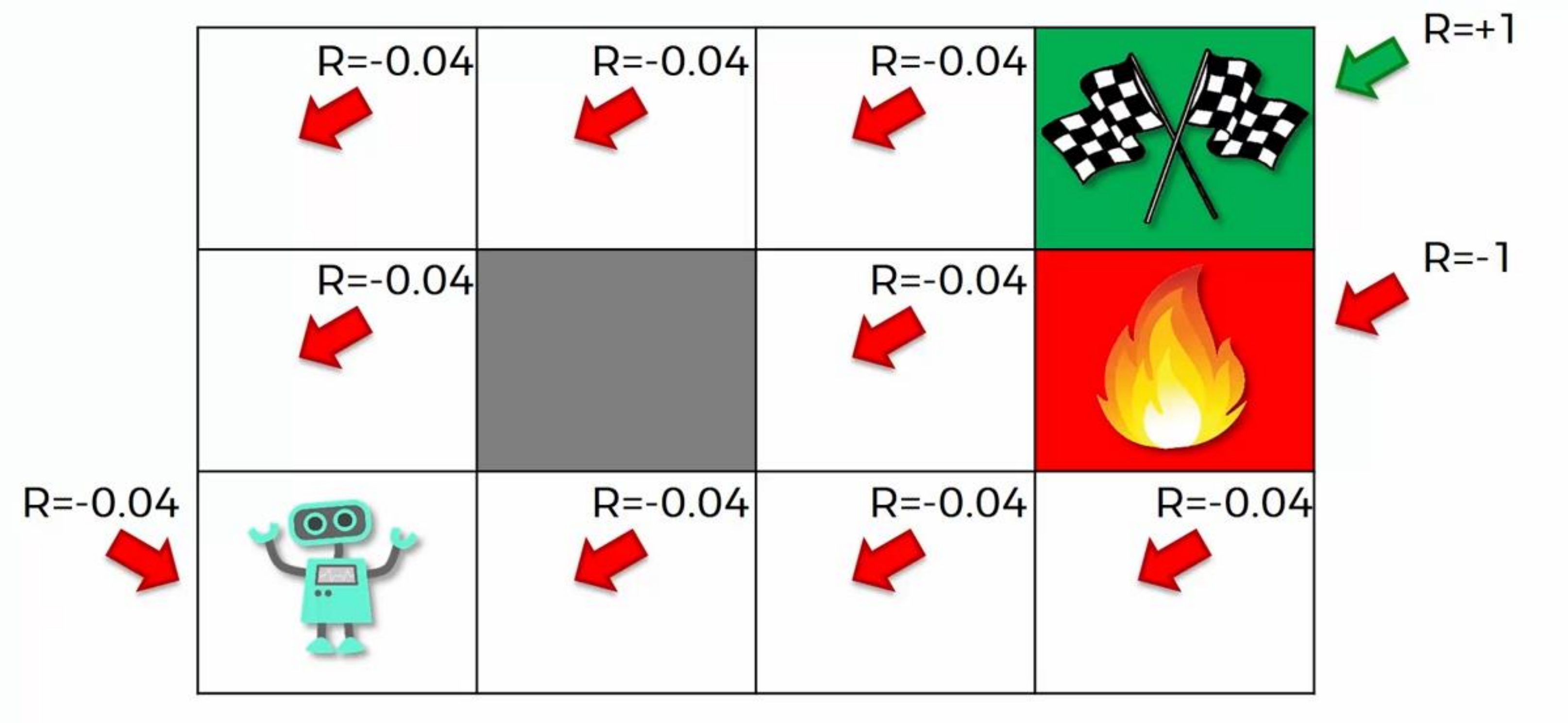
Reward Function

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.

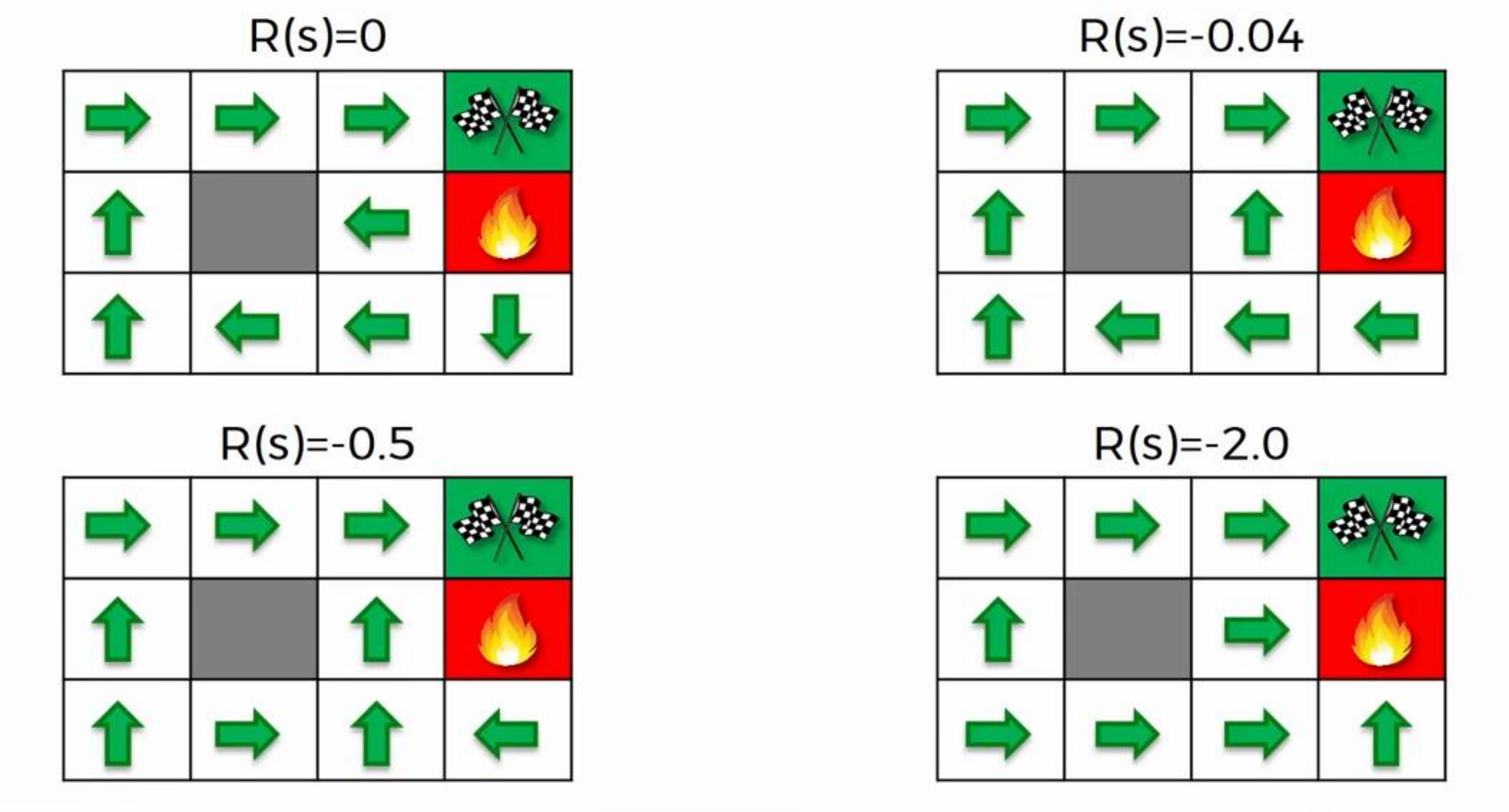
The expected return of the rewards is what's driving the agent to make the decisions it makes.

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

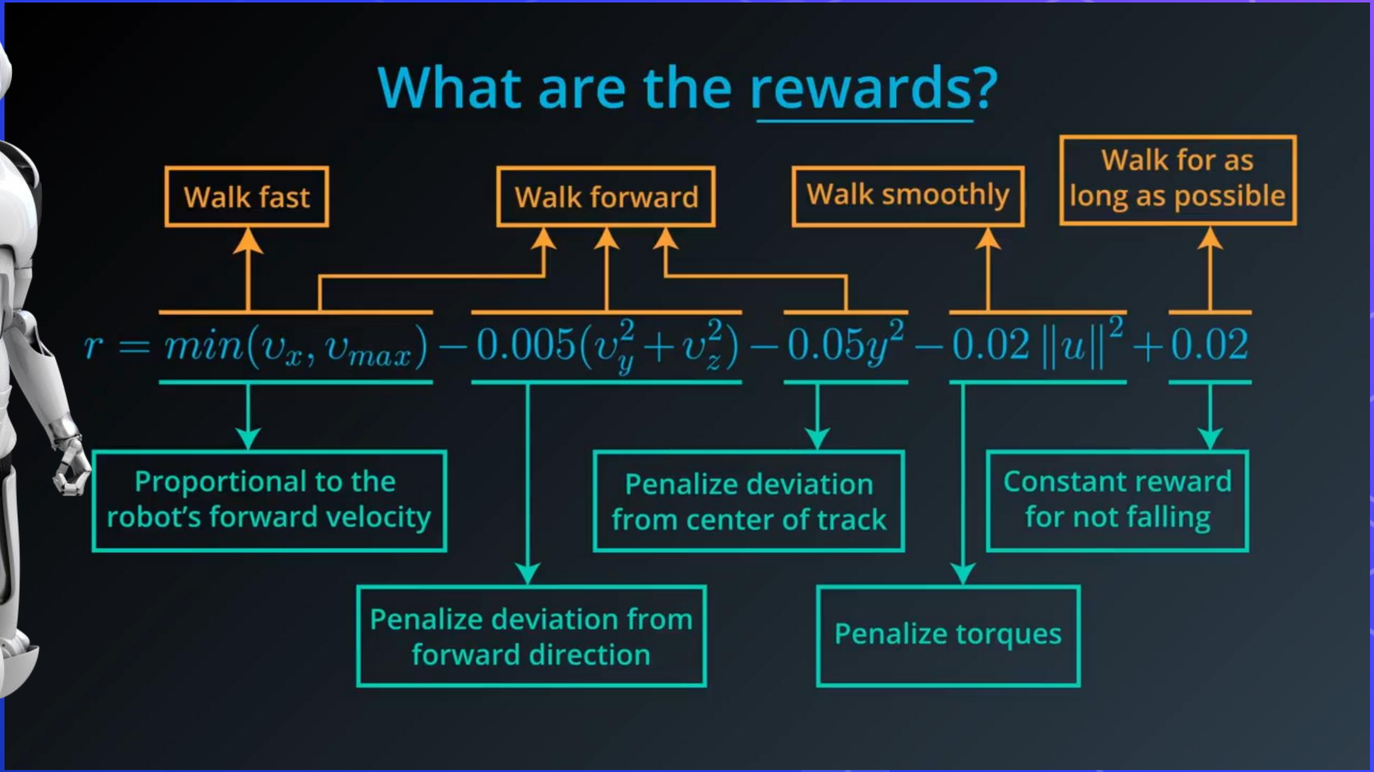
Reward Function



Reward Function



Reward Function



MDP Solution

- Policy.
- Exploration vs Exploitation.
- Reward Function.
- <u>**Discounted Reward Factor.**</u>
- 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).

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

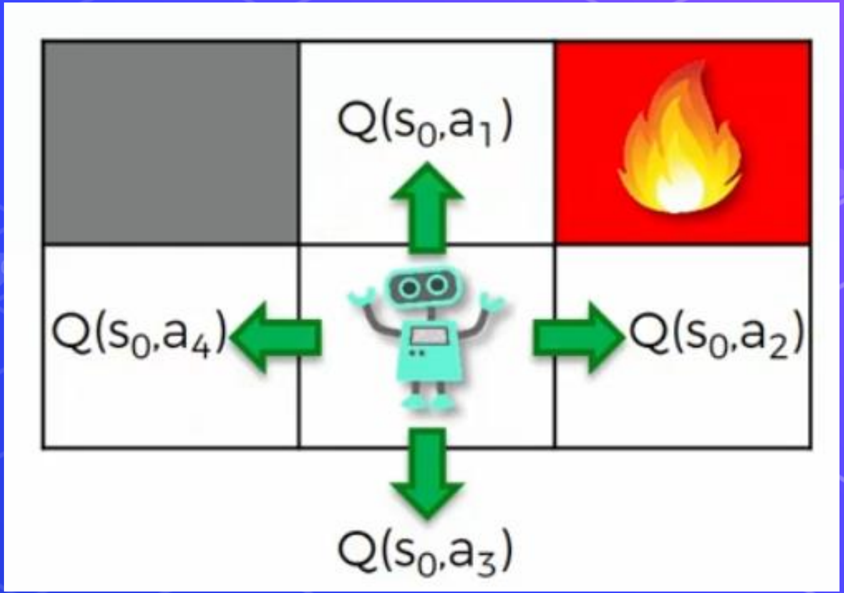
MDP Solution

- 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.

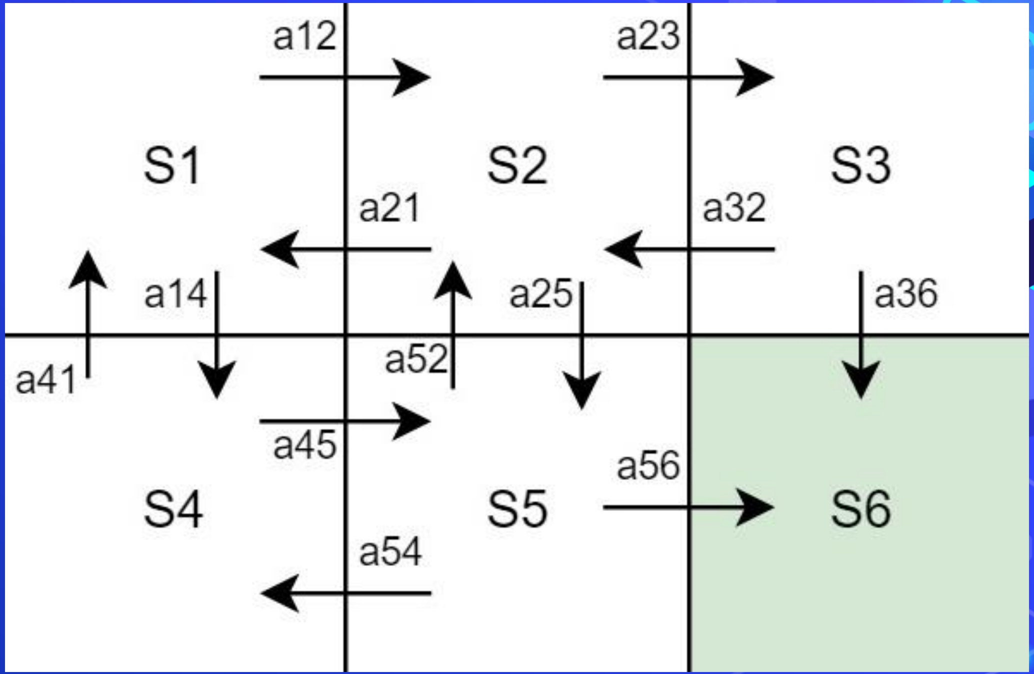
$$q^{new}(s,a) = (1-\alpha)\underbrace{q(s,a)}_{\text{old value}} + \alpha\overbrace{\left(R_{t+1} + \gamma \max_{a'} q(s',a')\right)}^{\text{learned value}}$$

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 epsilon 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

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

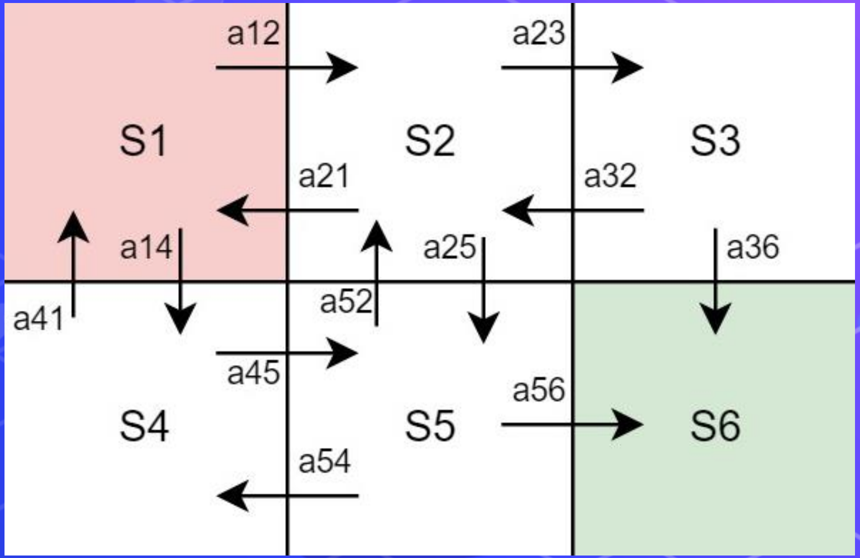
Q(S, A)	Value
Q(S1, A12)	0
Q(S1, A14)	0
Q(S2, A21)	0
Q(S2, A23)	0
Q(S2, A25)	0
Q(S3, A32)	0
Q(S3, A36)	0
Q(S4, A41)	0
Q(S4, A45)	0
Q(S5, A54)	0
Q(S5, A52)	0
Q(S5, A56)	0

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.

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

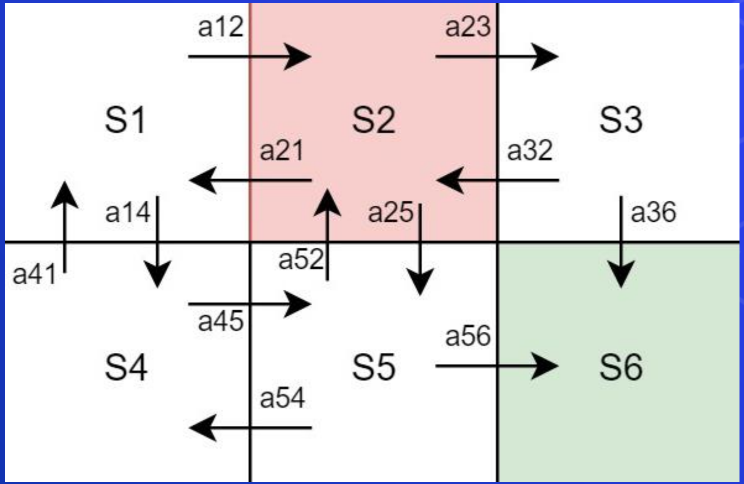



```
q_new(S1, a12) = r + 0.5 * max[q(S2, a21), q(S2, a23), q(S2, a25)]
= 0 + 0.5 * 0 = 0
```

Q Learning Algorithm (Grid World)

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

5- New State became S2.



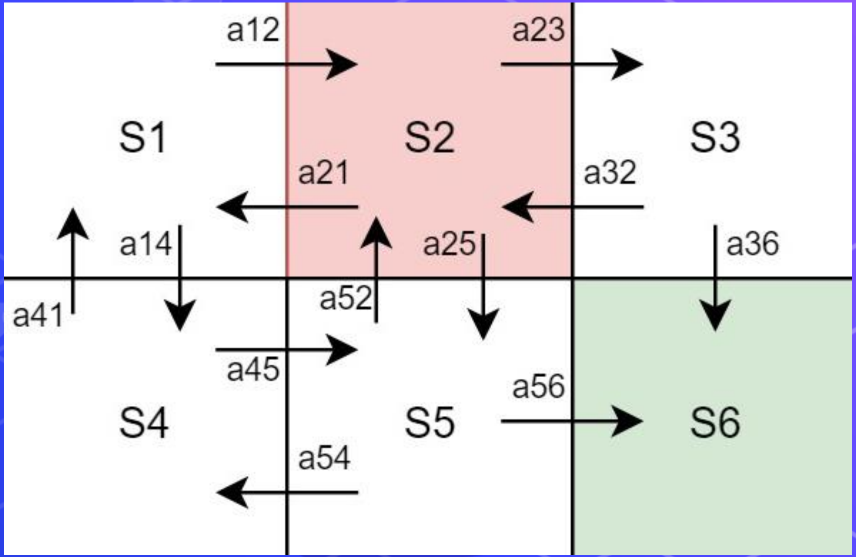
Q(S, A)	Value
Q(S1, A12)	0
Q(S1, A14)	0
Q(S2, A21)	0
Q(S2, A23)	0
Q(S2, A25)	0
Q(S3, A32)	0
Q(S3, A36)	0
Q(S4, A41)	0
Q(S4, A45)	0
Q(S5, A54)	0
Q(S5, A52)	0
Q(S5, A56)	0

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.

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

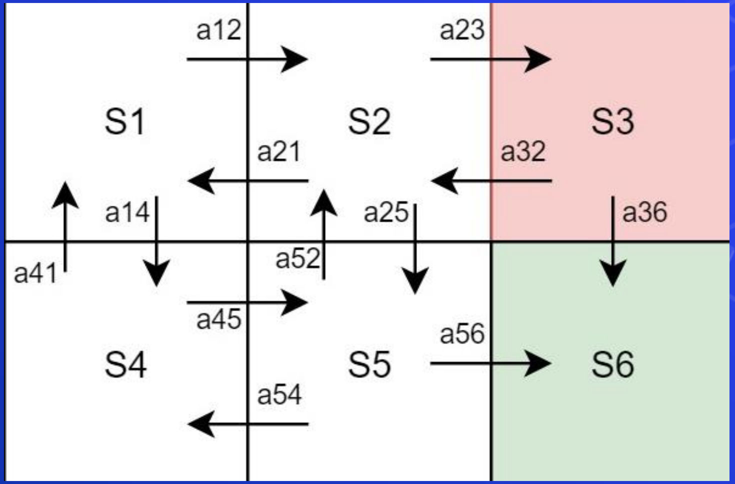



```
q_new(S2, a23) = r + 0.5 * max[q(S3, a32), q(S3, a36)]
= 0 + 0.5 * 0 = 0
```

Q Learning Algorithm (Grid World)

8- Update the q table with new Q(S2, A23) value = 0.

9- New State became S3.



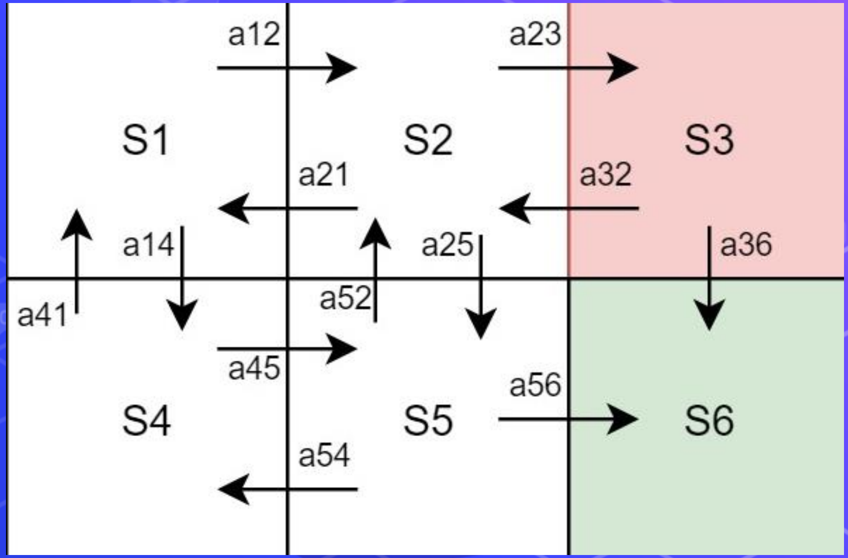
Q(S, A)	Value
Q(S1, A12)	0
Q(S1, A14)	0
Q(S2, A21)	0
Q(S2, A23)	0
Q(S2, A25)	0
Q(S3, A32)	0
Q(S3, A36)	0
Q(S4, A41)	0
Q(S4, A45)	0
Q(S5, A54)	0
Q(S5, A52)	0
Q(S5, A56)	0

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.

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

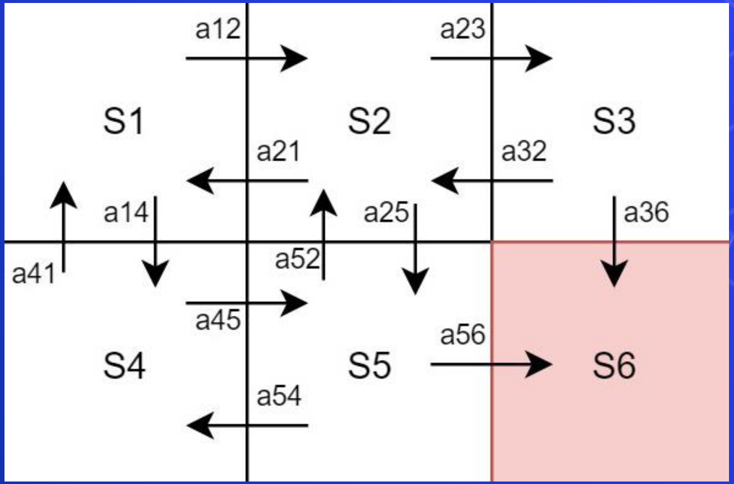


q_new(S3, a36) = r = 100, because there is no actions in S' -> S6 (Goal)

Q Learning Algorithm (Grid World)

12- Update the q table with new Q(S3, A36) value = 100.

13- New State became S6 (Goal or Terminal State) and the episode ends.



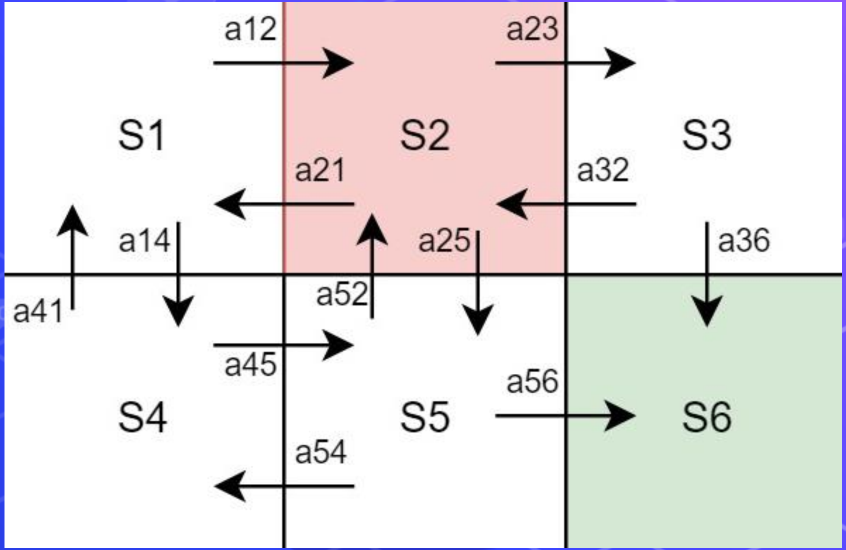
Q(S, A)	Value
Q(S1, A12)	0
Q(S1, A14)	0
Q(S2, A21)	0
Q(S2, A23)	0
Q(S2, A25)	0
Q(S3, A32)	0
Q(S3, A36)	100
Q(S4, A41)	0
Q(S4, A45)	0
Q(S5, A54)	0
Q(S5, A52)	0
Q(S5, A56)	0

Q Learning Algorithm (Grid World)

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.

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

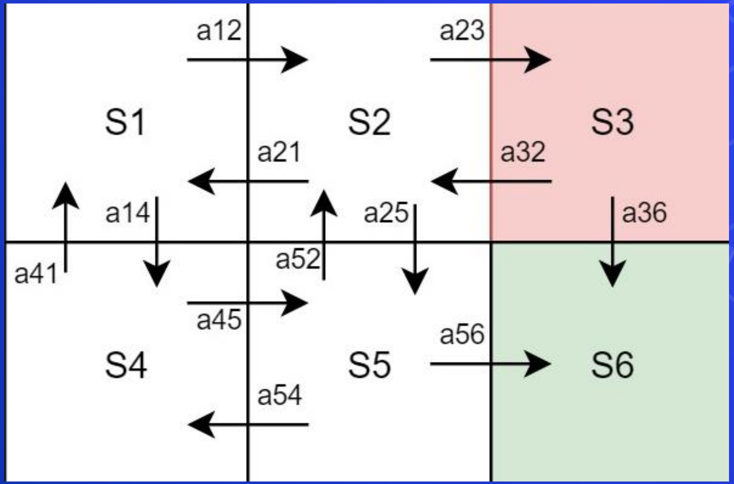



```
q_new(S2, a23) = r + 0.5 * max[q(S3, a32), q(S3, a36)]
= 0 + 0.5 * 100 = 50
```

Q Learning Algorithm (Grid World)

12- Update the q table with new Q(S2, A23) value = 50.

13- New State became S3, and So on for N episodes.



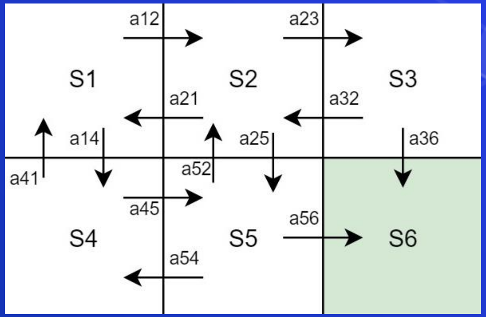
Q(S, A)	Value
Q(S1, A12)	0
Q(S1, A14)	0
Q(S2, A21)	0
Q(S2, A23)	50
Q(S2, A25)	0
Q(S3, A32)	0
Q(S3, A36)	100
Q(S4, A41)	0
Q(S4, A45)	0
Q(S5, A54)	0
Q(S5, A52)	0
Q(S5, A56)	0

Q Learning Algorithm (Grid World)

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.

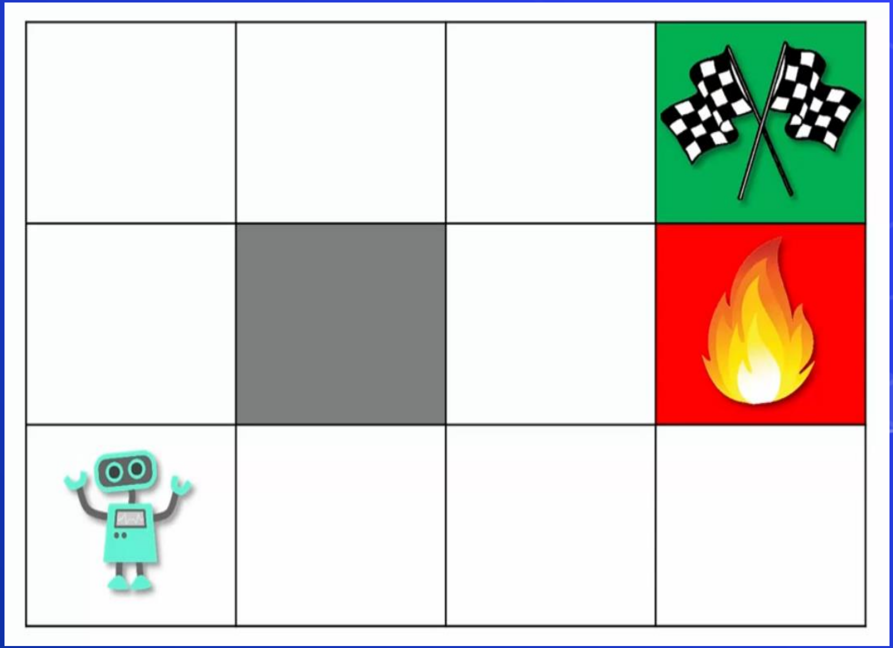
- At State 1 choose Action A12 or A14 = 25.
- At State 2 choose Action A23 = 50.
- At State 3 choose Action A36 = 100.
- At State 4 choose Action A45 = 50.
- At State 5 choose Action A56 = 100.

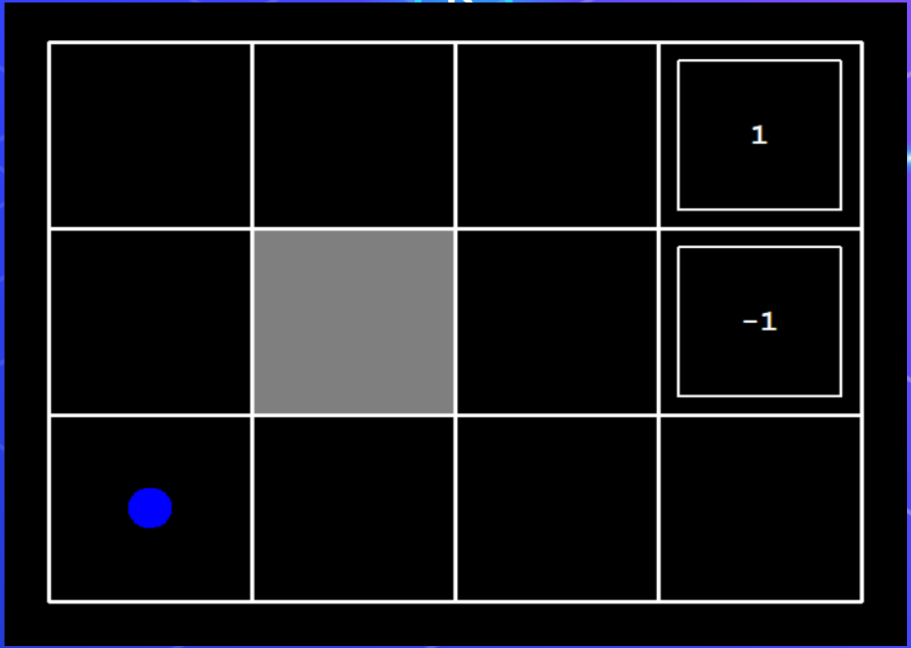


Q(S, A)	Value
Q(S1, A12)	25
Q(S1, A14)	25
Q(S2, A21)	12.5
Q(S2, A23)	50
Q(S2, A25)	25
Q(S3, A32)	25
Q(S3, A36)	100
Q(S4, A41)	12.5
Q(S4, A45)	50
Q(S5, A54)	25
Q(S5, A52)	25
Q(S5, A56)	100

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)

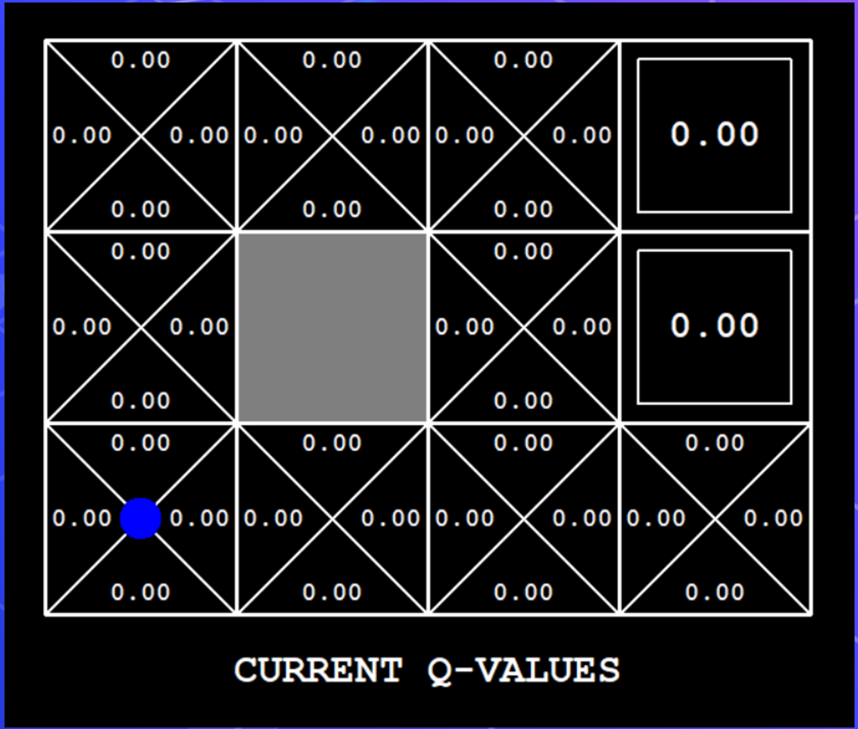
1. Initialize Q(s, a) by setting all of the Q-Values for each state equal to zero, epsilon = 1, gamma = 0.9, alpha = 0.7, the flag reward is +1 & fire penalty is -1 and every step has penalty 0, number of episodes = 400.

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

3. Based on the epsilon-greedy policy, choose an action, At first epsilon = 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 epsilon 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)

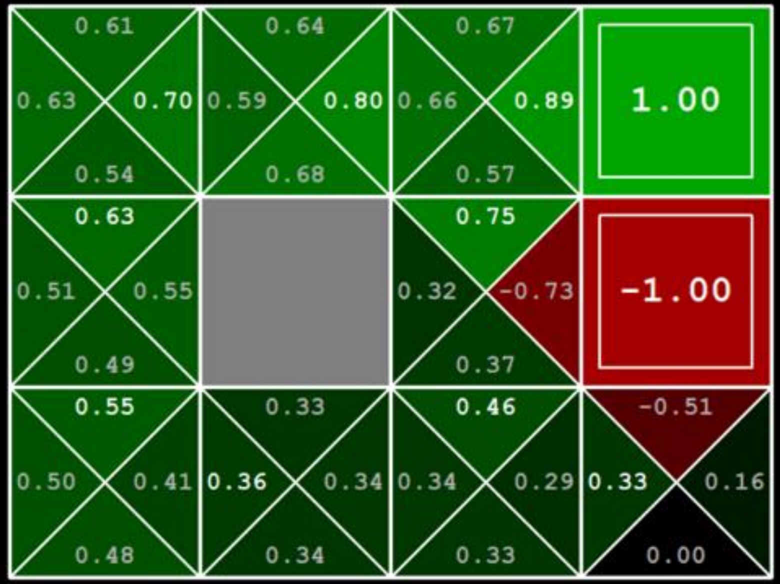
6. The episode ends when the agent reaches the flag or the fire, lets say the agent is in state [0, 2] it choses action right then it reaches the flag and gains the reward 1, so it knows that the action (right) led to the flag, it's a good action so updates the q-value of the action with the equation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

```
q_new([0, 2], 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

```
q_new([0, 1], Right) = (1-0.7) * 0 + 0.7 (0 + 0.9 * 0.7)
= 0.441
```

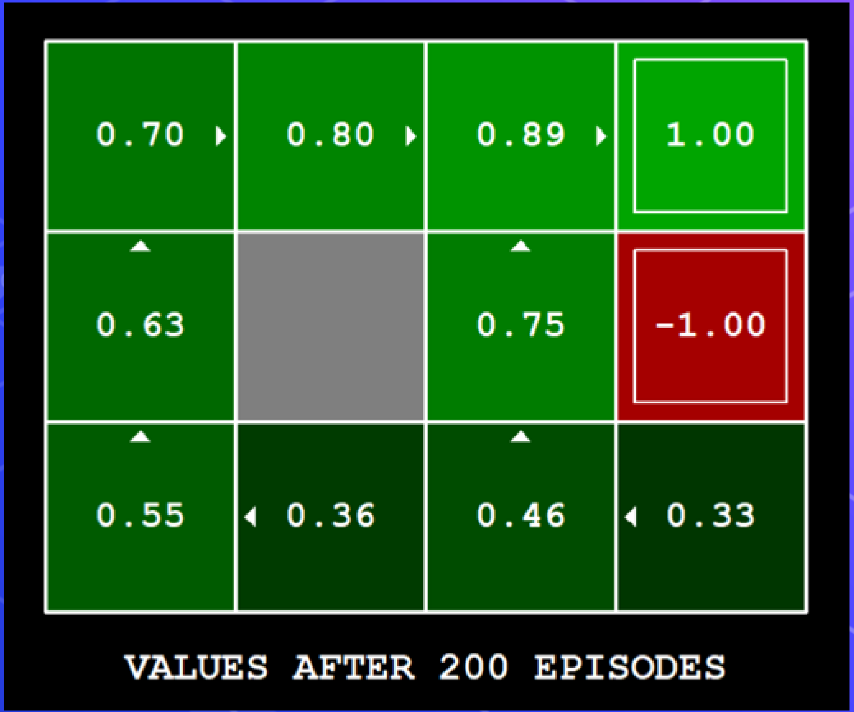


Q-VALUES AFTER 200 EPISODES

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 plays the game with its own to have the maximum reward.



Q Learning Algorithm (Another Grid World)

To run the code.

```
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.
- <u>**Open AI Gym environment.**</u>
- Coding time >_

Open AI Gym

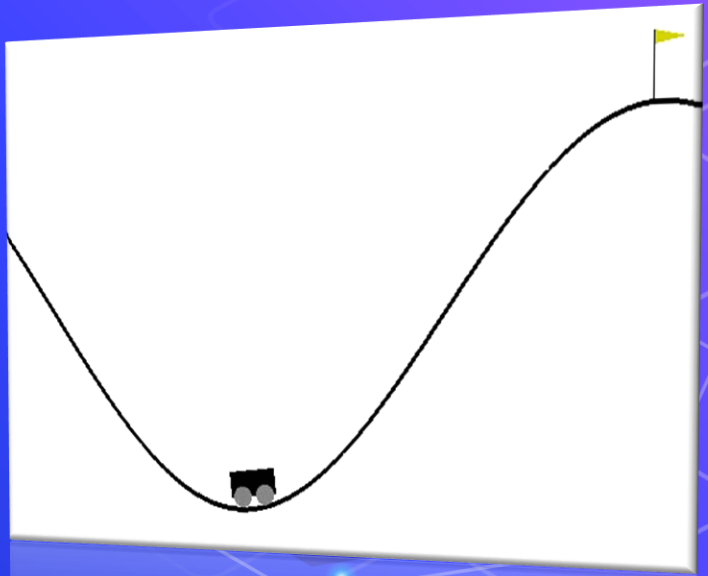
Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball.

https://gym.openai.com/

```
pip install gym
```

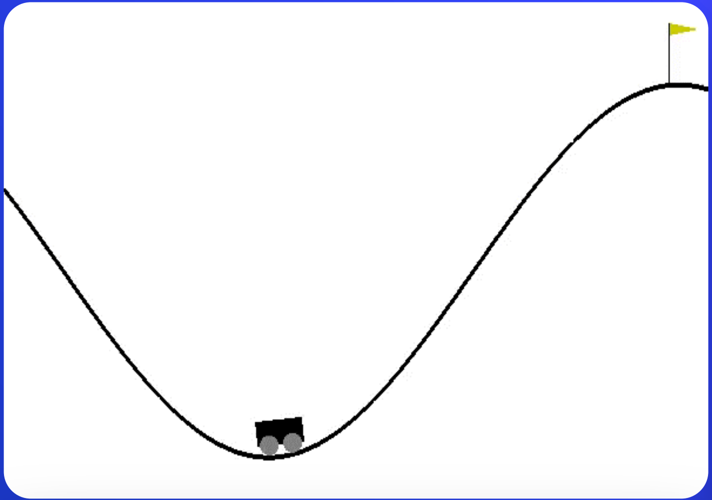
Open AI Gym – Mountain Car

```
import gym

env = gym.make('MountainCar-v0')

for i_episode in range(400):
    observation = env.reset() # get random state S0
    action = env.action_space.sample() # take random action
    observation, reward, done, info = env.step(action)
    env.render()
    if done:
        break

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 >_



```python
import gym
import numpy as np

env = gym.make("MountainCar-v0")

lr = 0.5
DISCOUNT = 0.95
EPISODES = 80
START_EPS_DECAY = 1
END_EPS_DECAY = EPISODES//2
epsilon = 1
eps_decay = epsilon/(END_EPS_DECAY - START_EPS_DECAY)

q_table = np.zeros(([10,10] + [env.action_space.n]))

```

```python
episode = 0
while episode < EPISODES:

    done = False
    discrete_state = calc_discrete_state(env.reset())

    while not done:

        # Exploit or explore
        if np.random.random() > epsilon:
            # Exploit from q-table
            action = np.argmax(q_table[discrete_state])
        else:
            # Explore - t
            action = np.random.randint(0, env.action_space.n)


        new_state, reward, done, _ = env.step(action)

        # Update q-table
        max_future_q = np.max(q_table[new_state_disc])
        current_q = q_table[discrete_state + (action,)]
        reward_fun = reward + DISCOUNT * max_future_q
        new_q = (1 - lr) * current_q + lr * reward_fun
        q_table[discrete_state + (action,)] = new_q

        discrete_state = calc_discrete_state(new_state_disc)

        env.render()

    if END_EPS_DECAY >= episode >= START_EPS_DECAY:
        epsilon -= epsilon_change

    episode += 1
```

Questions ?!

Thanks!

>_ Live long and prosper