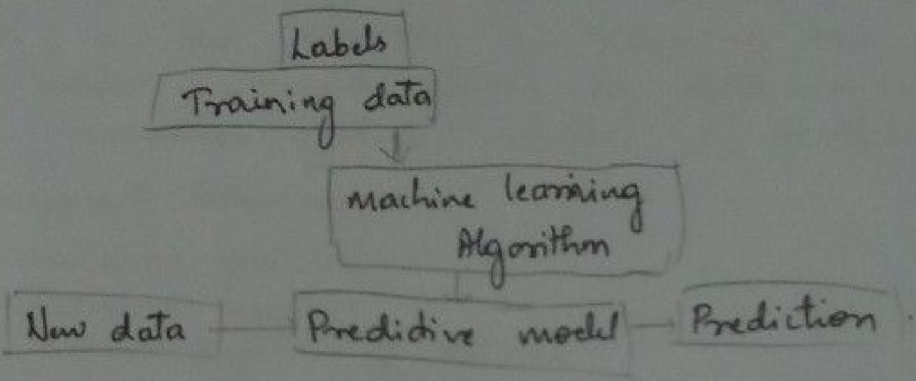
Nikitha M S
4NI17IS051
ISE 8 'A'
IS0321- IML

1 Machine learning is a field of study that gives computers the ability to study or learn without being explicitly programmed.

Types of machine learning.

1. Supervised learning.
   - Labeled data
   - direct feedback
   - Predict outcome.



→ classification is a subcategory of supervised learning which predict the categorical class labels of new instance, based on past observations, where the class labels are unordered values that can be group membership of an instance eg : spam email classification.

→ Regression which is also a type of supervised learning which is used for prediction of contineous outcomes called as regression analysis. Given the numbers or explanatory variables & a contineous response variable & try to find relationship between those variables.
eg. Predicting stock prices based on previous data.

2. Unsupervised learning
   - No labels / target
   - No feedback
   - Finding hidden structure in data.

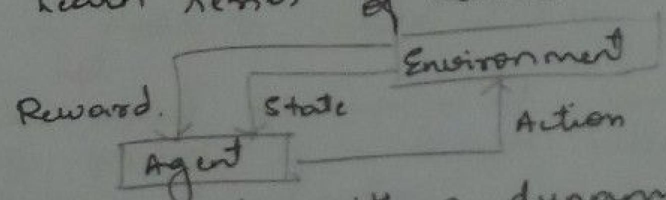
clustering :

1 of 5

clustering is technique that allows us to organize a information into meaningful subgroups without having any prior knowledge of group membership. Each cluster group which arises during analysis shares a certain degree of similarity between them.

> Dimentionality Reduction - Simplifies the input by mapping them into lower dimensional space. Higher the number of features it is difficult to visualise the training set. It is the process of reducing the no of random variables under consideration by obtaining a set of principal variables. feature selection & feature extraction

3. Reinforcement learning
   - Decision process
   - Reward system
   - Learn series of actions.



> Computer interacts with a dynamic environment in which it must perform a certain goal.

Eg : Game of chess.

2. Sample : It is seperate rows in a feature matrix. In IRIS dataset there are 150 Iris flowers measurement from 3 different flowers species, setosa, Virginica & Versicolor. Each flower sample represent 1 row in our data set.

Features : Each flower sample which is one row in data - set and the flower measurements in centimeter stored as columns which is called as features of data set, features include sepal length, sepal width, Petal length & petal width.

We represent data set in Matrix format - X where each sample as seperate row and each feature as seperate column. This has 150 samples & 4 features which can be written as 150 × 4 matrix $X \in R^{150 \times 4}$.

$$\begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & x_4^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(150} & x_2^{(150)} & x_3^{(150)} & x_4^{(150)} \end{bmatrix}$$

Each row in feature matrix represents one flower instan

$x^{(i)} = [x_1^{(i)} \quad x_2^{(i)} \quad x_3^{(i)} \quad x_4^{(i)}]$ → vector (4 Dimentional row-vector).

Each feature dimension is 150-dimensional column vector

$$x_j = \begin{bmatrix} x_j^{(1)} \\ x_j^{(2)} \\ \vdots \\ x_j^{(150} \end{bmatrix}$$

3. Perceptron algorithm.

→ classification algorithm

→ Algorithm that would automatically learn the optimal weight co-efficients that are then multiplied with the I/P features in order to make decicion of whether a neuron fines or not

→ In the content of supervised learning & classification it is used to predict if a sample belongs to one class or the other (Binary classification (+ve class) → 1 & (-ve class) → -1)

→ Activation function - $\phi(z)$ - takes a linear combination of certain input values ~~X~~ X & a corresponding weight vector W, where z is the so-called net input

$$z = w_1 x_1 + \dots + w_m x_m$$

→ Perceptron can be used ~~on~~ if the data can be linearly seperable & learning rate is sufficiently small.

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

m - no of column/features

- If the activation of a particular sample $x^{(i)}$. o/p of $\phi(z)$ > defined threshold θ we predict class 1 otherwise we predict class -1

→ It is a simple unit step function

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq \theta \\ -1 & \text{otherwise} \end{cases}$$

θ - threshold.

→ For simplicity bring θ to left hand side.

$$z = w_0 x_0 + w_1 x_1 + \dots + w_m x_m = w^T x.$$

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

Nikitha M S
4NI17IS051
ISE 8 'A'
IS0321 - IML

Perceptron implementation algorithm

1. Initialize the weight to 0 or small random numbers
2. For each training sample $x^{(i)}$ perform these steps
    1. compute the output value $\hat{y}$
    2. Update the weights.

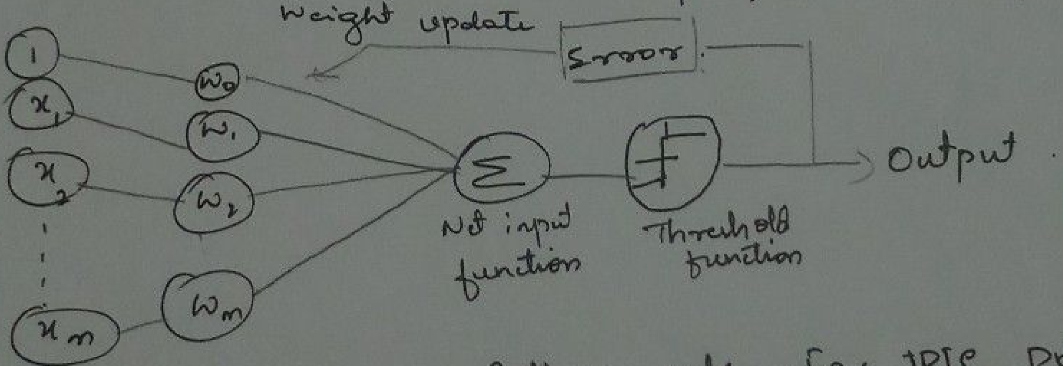
Here the output value is the class label predicted by unit step function.

Updating weights - $w_j = w_j + \Delta w_j$

The value of Δw_j is calculated by perceptron learning rule $\Delta w_j = \eta(y^{(i)} - \hat{y}^{(i)})\, x_j^{(i)}$

where η - learning rate (0 to 1)
$y^{(i)}$ - true class lable of i^{th} sample
$\hat{y}^{(i)}$ - predicted lable.

For 2D dataset $\Delta w_0 = \eta(y^{(i)} - \text{output}^{(i)})]$
$\Delta w_1 = \eta(y^{(i)} - \text{output}^{(i)})\, x_1^{(i)}$
$\Delta w_2 = \eta(y^{(i)} - \text{output}^{(i)})\, x_2^{(i)}$



Perceptron algorithm - Python code - For IRIS Dataset.

```
import numpy as np
class perceptron (object):
  '''eta : float
    n_iter : ~~int~~ int
    w_ : 1d-Array
    errors_ : list '''
    def __init__ (self, eta = 0.01, n_iter=50 ,random_state =1):
           self.eta = eta
           self.n_iter = n_iter
           self.random_state = random_state
    def fit (self, x, y) :
       rgen = np.random.RandomState (self.random_state)
       self.w_ : rgen.normal ( loc=0.0, scale: 0.01, size=1 +
                                              x.shape[1])
```

4/5

Nikitha M S
1AN17IS051
ISE ~~B~~ 'A'
180321 -IML

```
self.errors_ = []
for _ in range(self.n_iter):
    errors = 0
    for xi, target in zip(x, y):
        update = self.eta * (target - self.predict(xi))
        self.w_[1:] += update * xi
        self.w_[0] += update
        errors += int(update != 0.0)
    self.errors_.append(errors)
return self
def net_input(self, x):
    return np.dot(X, self.w_[1:]) + self.w_[0]
def predict(self, X):
    return np.where(self.net_input(x) >= 0.0, 1, -1)
```

5 of 5