6-implement-k-means-algorithm

November 8, 2023

[1]:
```
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

[2]:
```
np.random.seed(0)
```

[3]:
```
X=np.random.rand(100,2)
```

[4]:
```
k=3
```

[5]:
```
kmeans=KMeans(n_clusters=k)
```

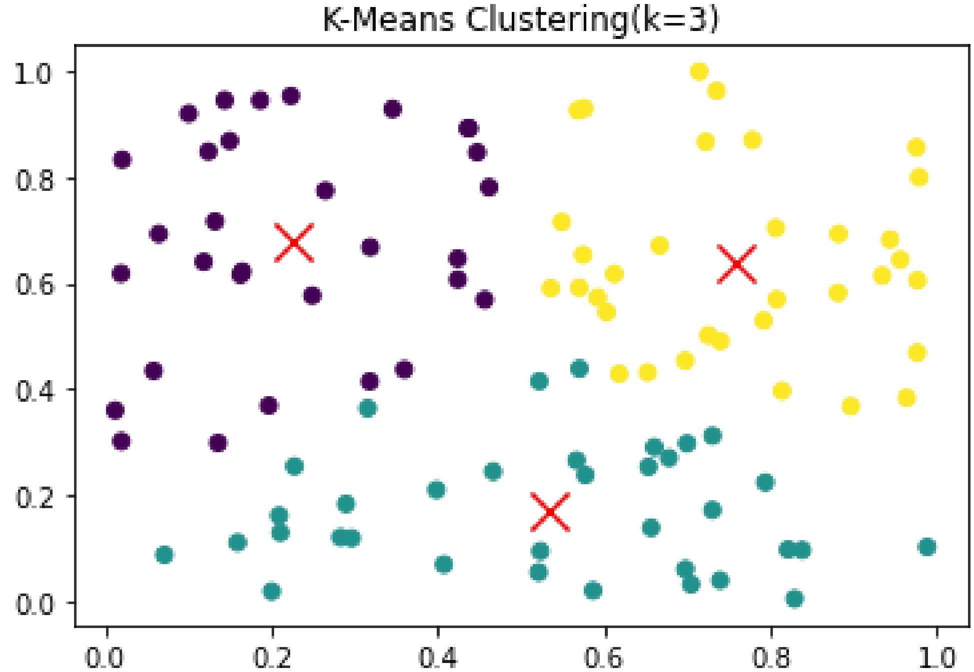
[6]:
```
kmeans.fit(X)
```

[6]:
```
KMeans(n_clusters=3)
```

[7]:
```
cluster_centers=kmeans.cluster_centers_
```

[9]:
```
labels=kmeans.labels_
```

[12]:
```
plt.scatter(X[:,0],X[:,1],c=labels)
plt.scatter(cluster_centers[:,0],cluster_centers[:
 ↪,1],marker='x',s=200,color='red')
plt.title(f"K-Means Clustering(k={k})")
plt.show()
```


K-Means Clustering(k=3)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

```
[ ]:
```

ment-k-nearest-neighbour-algorithm

November 8, 2023

[19]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

[20]:
```
np.random.seed(0)
X=np.random.rand(100,2)
y=np.random.choice([0,1],size=100)
```

[21]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
  ↪2,random_state=42)
```

[22]:
```
k=3
knn_classifier=KNeighborsClassifier(n_neighbors=k)
knn_classifier.fit(X_train,y_train)
```

[22]:
```
KNeighborsClassifier(n_neighbors=3)
```

[23]:
```
y_pred=knn_classifier.predict(X_test)
```

[24]:
```
accuracy=accuracy_score(y_test,y_pred)
print(f"Accuracy : {accuracy*100:.2f}%")
```

```
Accuracy : 30.00%
```

5-tsp-usiing-hueristic-approach

November 8, 2023

[1]:
```python
import math
```

[2]:
```python
def distance(point1,point2) :
    return math.sqrt((point1[0]-point2[0])**2 + (point1[1]-point2[1])**2)
```

[3]:
```python
def nearest_neighbours(points) :

    n = len(points)
    unvisited =  set(range(n))
    tour = [0]
    unvisited.remove(0)

    while unvisited :

        current_point  = tour[-1]
        nearest_point = min(unvisited,key =  lambda x:
 ↪distance(points[current_point],points[x]))
        tour.append(nearest_point)
        unvisited.remove(nearest_point)

    tour.append(tour[0])

    return tour
```

[5]:
```python
if __name__ == "__main__" :

    points = [(0,0),(1,2),(2,3),(3,4),(4,2)]

    tour = nearest_neighbours(points)

    print("Optimal Tour:",tour)
```

```
Optimal Tour: [0, 1, 2, 3, 4, 0]
```

[ ]:
```python

```

1-best-first-search-algorithm

November 8, 2023

[1]:
```python
from queue import PriorityQueue
v = 14
graph = [[] for i in range(v)]
# Function For Implementing Best First Search
# Gives output path having lowest cost
def best_first_search(actual_Src, target, n):
    visited = [False] * n
    pq = PriorityQueue()
    pq.put((0, actual_Src))
    visited[actual_Src] = True
    while pq.empty() == False:
        u = pq.get()[1]
        # Displaying the path having lowest cost
        print(u, end=" ")
        if u == target:
            break
        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()
# Function for adding edges to graph
def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))
# The nodes shown in above example(by alphabets) are
# implemented using integers addedge(x,y,cost);
addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
addedge(8, 9, 5)
addedge(8, 10, 6)
```

```
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)
source = 0
target = 9
best_first_search(source, target,v)
```

```
0 1 3 2 8 9
```

[ ]:

```

```

3-water-jug-problem

November 8, 2023

[3]:
```
def water_jug_dfs(jug1_capacity, jug2_capacity, target_capacity):
    def dfs(jug1, jug2, path):
        if jug1 == target_capacity or jug2 == target_capacity:
            print("Solution found:", path)
            return
# Fill jug1
        if jug1 < jug1_capacity:
            new_jug1 = jug1_capacity
            new_jug2 = jug2
            if (new_jug1, new_jug2) not in visited:
                visited.add((new_jug1, new_jug2))
                dfs(new_jug1, new_jug2, path + f"Fill Jug1\n")

# Fill jug2
        if jug2 < jug2_capacity:
            new_jug1 = jug1
            new_jug2 = jug2_capacity
            if (new_jug1, new_jug2) not in visited:
                visited.add((new_jug1, new_jug2))
                dfs(new_jug1, new_jug2, path + f"Fill Jug2\n")

#pour water from jug1 to jug2
        if jug1>0 and jug2<jug2_capacity:
            pour_amount=min(jug1,jug2_capacity-jug2)
            new_jug1= jug1-pour_amount
            new_jug2 = jug2 + pour_amount
            if (new_jug1, new_jug2) not in visited:
                visited.add((new_jug1, new_jug2))
                dfs(new_jug1, new_jug2, path + f"Pour Jug1 into Jug2\n")

# Pour water from jug2 to jug1
        if jug2 > 0 and jug1 < jug1_capacity:
            pour_amount = min(jug2, jug1_capacity - jug1)
            new_jug1 = jug1 + pour_amount
            new_jug2 = jug2 - pour_amount
            if (new_jug1, new_jug2) not in visited:
                visited.add((new_jug1, new_jug2))
```

```
                dfs(new_jug1, new_jug2, path + f"Pour Jug2 into Jug1\n")
# Empty jug1
        if jug1 > 0:
            new_jug1 = 0
            new_jug2 = jug2
            if (new_jug1, new_jug2) not in visited:
                visited.add((new_jug1, new_jug2))
                dfs(new_jug1, new_jug2, path + f"Empty Jug1\n")

# Empty jug2
        if jug2 > 0:
            new_jug1 = jug1
            new_jug2 = 0
            if (new_jug1, new_jug2) not in visited:
                visited.add((new_jug1, new_jug2))
                dfs(new_jug1, new_jug2, path + f"Empty Jug2\n")

    visited= set()
    dfs(0, 0, "")

# Example usage:
jug1_capacity = 4
jug2_capacity = 3
target_capacity = 2


water_jug_dfs(jug1_capacity, jug2_capacity, target_capacity)
```

```
Solution found: Fill Jug1
Fill Jug2
Empty Jug1
Pour Jug2 into Jug1
Fill Jug2
Pour Jug2 into Jug1

Solution found: Fill Jug1
Pour Jug1 into Jug2
Empty Jug2
Pour Jug1 into Jug2
Fill Jug1
Pour Jug1 into Jug2
```

[ ]:

```

```

4-8-queens

November 8, 2023

[1]:
```
print("Taking the number of queens")

N = int(input())

board =  [[0]*N for _ in range(N)]

def attack(i,j) :

    for k in range(0,N) :
        if board[i][k] ==1  or board[k][j] ==1 :
            return True

    for k in range(0,N) :
            for l in range(0,N) :
                if(k+l==i+j) or (k-l==i-j) :
                    if board[k][l] == 1 :
                        return True

    return False

def N_queens(n):

    if n==0 :
        return True

    for i in range(0,N) :
        for j in range(0,N) :

            if (not(attack(i,j))) and (board[i][j]!=1) :
                board[i][j] = 1

                if N_queens(n-1) == True :
                    return True
                board[i][j] = 0
    return False

N_queens(N)
```

```python
for i in board :
    print (i)
```

```
Taking the number of queens
8
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
```

[ ]:

```python

```

ard-chaining-or-backward-chaining

November 8, 2023

1 Forawrd Chaining

[1]:
```
class rule :
    def __init__(self,antecedents,consequent) :

        self.antecedents =  antecedents
        self.consequent  = consequent
```

[2]:
```
class KnowledgeBase :

    def __init__(self) :

        self.facts  =  set()
        self.rules =  []

    def add_facts(self,facts) :

        self.facts.add(facts)

    def add_rules(self,rule) :

        self.rules.append(rule)

    def apply_forward_chaining(self) :

        new_facts_derived =  True

        while new_facts_derived :

            new_facts_derived = False

            for rule in self.rules  :

                if all(antecedent in self.facts for antecedent in rule.
↪antecedents) and rule.consequent not in self.facts  :

                    self.facts.add(rule.consequent)
```

```
                    new_facts_derived =  True
```

[3]:

```
if __name__  == "__main__"  :

    kb = KnowledgeBase()

    #Define rules and facts


    rule1 =  rule(["A","C"],"E")
    rule2  = rule(["A","E"],"G")
    rule3 =  rule(["B"],"E")
    rule4 =  rule(["G"],"D")
    kb.add_rules(rule1)
    kb.add_rules(rule2)
    kb.add_rules(rule3)
    kb.add_rules(rule4)

    kb.add_facts("A")
    kb.add_facts("C")

    kb.apply_forward_chaining()

    print("Derived Facts :",kb.facts)
```

```
Derived Facts : {'G', 'A', 'E', 'C', 'D'}
```

9-backward-chaining

November 8, 2023

[3]:
```
knowledge_base  = {

    "rule1" : {

        "if" : ["A","B"]  ,
        "then":"C"
    },
    "rule2" : {

        "if" : ["D"] ,
        "then" : "A"

    } ,
    "rule3" : {
        "if" : ["F"] ,
        "then" : "B"
    } ,
    "rule4" : {
        "if" : ["F"] ,
        "then" : "D"
    } ,
    "rule5" : {
        "if" : ["G"] ,
        "then" : "E"
    }
}

#Define a function to perform backward chaining

def backward_chaining(goal,known_facts) :

    if goal in known_facts :

        return True

    for rule , value  in knowledge_base.items() :

```

```
        if goal in value["if"] :

            all_conditions_met = all(condition in known_facts for condition in␣
 ↪value["if"])

            if all_conditions_met and backward_chaining(value["then"] ,
                                                        known_facts) :
                return True
            return False

goal  = "C"
known_facts = ["G","F","E"]

    # check if goal can be reached using backward chaining

if backward_chaining(goal,known_facts) :

        print(f"The goal '{goal}' can be reached .")

else  :
        print(f"The goal '{goal}' cannot  be reached .")
```

```
The goal 'C' cannot  be reached .
```

[ ]:

```

```

10-implement-svm

November 8, 2023

[1]:
```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

[3]:
```
X,y=datasets.
 ↪make_classification(n_samples=500,n_features=3,n_informative=2,n_redundant=0,random_state=4
```

[4]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
 ↪2,random_state=42)
```

[5]:
```
svm_classifier=SVC(kernel='linear',C=1.0)
```

[6]:
```
svm_classifier.fit(X_train,y_train)
```

[6]:
```
SVC(kernel='linear')
```

[7]:
```
y_pred=svm_classifier.predict(X_test)
```

[9]:
```
accuracy=accuracy_score(y_test,y_pred)
```

[10]:
```
print("Accuracy : ",accuracy)
```

```
Accuracy :  0.87
```