1

2 # (1). Implement A\* Search algorithm.

3

4 def aStarAlgo(start\_node, stop\_node):

5

1. open\_set = set(start\_node)
2. closed\_set = set()

8

9 g = {}

10 parents = {}

11

1. g[start\_node] = 0
2. parents[start\_node] = start\_node

14

1. while len(open\_set) > 0:
2. n = None

17

1. for v in open\_set:
2. if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):

|  |  |  |  |
| --- | --- | --- | --- |
| 20 |  | | n = v |
| 21 |  | |  |
| 22 | if n == | | stop\_node or Graph\_nodes[n] == None: |
| 23 | pass | |  |
| 24 |  | |  |
| 25 | else: | |  |
| 26 | for | | (m, weight) in get\_neighbors(n): |
| 27 |  | |  |
| 28 |  | | if m not in open\_set and m not in closed\_set: |
| 29 |  | | open\_set.add(m) |
| 30 |  | | parents[m] = n |
| 31 |  | | g[m] = g[n] + weight |
| 32 |  | |  |
| 33 |  | | else: |
| 34 |  | | if g[m] > g[n] + weight: |
| 35 |  | | g[m] = g[n] + weight |
| 36 |  | | parents[m] = n |
| 37 |  | |  |
| 38 |  | | if m in closed\_set: |
| 39 |  | | closed\_set.remove(m) |
| 40 |  | | open\_set.add(m) |
| 41 |  | |  |
| 42 | if n == | | None: |
| 43 |  | print('Path does not exist!') | |
| 44 |  | return None | |
| 45 |  |  | |
| 46 |  | if n == stop\_node: | |
| 47 |  |  | |
| 48 |  | path = [] | |
| 49 |  | while parents[n] != n: | |
| 50 |  | path.append(n) | |
| 51 |  | n = parents[n] | |
| 52 |  |  | |
| 53 |  | path.append(start\_node) | |
| 54 |  | path.reverse() | |
| 55 |  | print('Path found: {}'.format(path)) | |
| 56 |  | return path | |
| 57 |  |  | |
| 58 |  | open\_set.remove(n) | |
| 59 |  | closed\_set.add(n) | |
| 60 |  |  | |
| 61 |  | print('Path does not exist!') | |
| 62 |  | return None | |
| 63 |  |  | |
| 64 |  |  | |
| 65 | def | get\_neighbors(v): | |
| 66 |  | if v in Graph\_nodes: | |
| 67 |  | return Graph\_nodes[v] | |
| 68 |  | else: | |
| 69 |  | return None | |

70

1. def heuristic(n):
2. H\_dist = {

73 'A': 2,

74 'B': 6,

75 'C': 2,

76 'D': 3,

77 'S': 4,

78 'G': 0,

79 }

80 return H\_dist[n]

81

82

83 Graph\_nodes = {

84 'A': [('B', 3), ('C', 1)],

85 'B': [('D', 3)],

86 'C': [('D', 1), ('G', 5)],

87 'D': [('G', 3)],

88 'S': [('A', 1)],

89 'G': []

90 }

1. aStarAlgo('S', 'G')
2. aStarAlgo('A', 'B')
3. aStarAlgo('B', 'S')

94

95 '''

96 ---Output---

97 Path found: ['S', 'A', 'C', 'D', 'G']

1. Path found: ['A', 'B']
2. Path does not exist!

100 '''

1

2 # (2). Implement AO\* Search algorithm.

3

4

5 class Graph:

6

7 def init (self, graph, heuristicNodeList, startNode):

8

1. self.graph = graph
2. self.H = heuristicNodeList
3. self.start = startNode
4. self.parent = {}
5. self.status = {}
6. self.solutionGraph = {}

15

1. def applyAOStar(self):
2. self.aoStar(self.start, False)

18

1. def getNeighbors(self, v):
2. return self.graph.get(v, '')

21

1. def getStatus(self, v):
2. return self.status.get(v, 0)

24

1. def setStatus(self, v, val):
2. self.status[v] = val

27

1. def getHeuristicNodeValue(self, n):
2. return self.H.get(n, 0)

30

1. def setHeuristicNodeValue(self, n, value):
2. self.H[n] = value

33

1. def printSolution(self):
2. print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE:"
3. , self.start)

37 print("===========================================================")

38 print(self.solutionGraph)

39 print("===========================================================")

40

41

42 def computeMinimumCostChildNodes(self, v):

43

1. minimumCost = 0
2. costToChildNodeListDict = {}
3. costToChildNodeListDict[minimumCost] = []
4. flag = True

48

1. for nodeInfoTupleList in self.getNeighbors(v):
2. cost = 0
3. nodeList = []

52

1. for c, weight in nodeInfoTupleList:
2. cost = cost + self.getHeuristicNodeValue(c) + weight
3. nodeList.append(c)

56

1. if flag == True:
2. minimumCost = cost
3. costToChildNodeListDict[minimumCost] = nodeList
4. flag = False
5. else:
6. if minimumCost > cost:
7. minimumCost = cost
8. costToChildNodeListDict[minimumCost] = nodeList

65

66 return minimumCost, costToChildNodeListDict[minimumCost]

67

68

69

70

71 def aoStar(self, v, backTracking):

72

1. print("HEURISTIC VALUES :", self.H)
2. print("SOLUTION GRAPH :", self.solutionGraph)
3. print("PROCESSING NODE :", v)
4. print(" ")

77

78 if self.getStatus(v) >= 0:

79

1. minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)
2. self.setHeuristicNodeValue(v, minimumCost)
3. self.setStatus(v, len(childNodeList))
4. solved = True

84

1. for childNode in childNodeList:
2. self.parent[childNode] = v
3. if self.getStatus(childNode) != -1:
4. solved = solved & False

89

1. if solved == True:
2. self.setStatus(v, -1)
3. self.solutionGraph[v] = childNodeList

93

1. if v != self.start:
2. self.aoStar(self.parent[v], True)

96

1. if backTracking == False:
2. for childNode in childNodeList:
3. self.setStatus(childNode, 0)
4. self.aoStar(childNode, False)

103 h1 = {'A': 38, 'B': 17, 'C': 9, 'D': 27, 'E': 5, 'F': 10, 'G': 3,

104 'H': 4,'I': 15, 'J': 10}

105

106 graph1 = {

107 'A': [[('B', 1), ('C', 1)], [('D', 1)]],

108 'B': [[('E', 1)], [('F', 1)]],

109 'C': [[('G', 1)], [('H', 1)]],

110 'D': [[('I', 1), ('J', 1)]]

111 }

112

1. G1 = Graph(graph1, h1, 'A')
2. G1.applyAOStar()
3. G1.printSolution()
4. print("HEURISTIC VALUES :", G1.H)
5. print("SOLUTION GRAPH :", G1.solutionGraph)
6. print('status:', G1.status)
7. print('parent:', G1.parent)

F eli h/: o me m/ o h a mme d D/ e s k to p te/ mp c/ a n d id a te E mil ni a oti n p. y P a g e 1 o 1f

1

* 1. (3).For a given set of training data examples stored in a .CSV file, implement and
  2. demonstrate the  CANDIDATE-ELIMINATION ALGORITHM  to output a description
  3. of the set of allhypotheses consistent with the training examples.

5

6

1. import numpy as np
2. import pandas as pd

9

10 data = pd.read\_csv('data1.csv')

11

1. concepts = np.array(data.iloc[:, 0:-1])
2. print(concepts)

14

1. target = np.array(data.iloc[:, -1])
2. print(target)

17

18

19 def learn(concepts, target):

20

1. specific\_h = concepts[0].copy()
2. print('initialization of specific\_h and general\_h')
3. print(specific\_h)

24

1. general\_h = [['?' for i in range(len(specific\_h))] for i in
2. range(len(specific\_h))]
3. print(general\_h)

28

1. for i, h in enumerate(concepts):
2. if target[i] == 'yes':
3. for x in range(len(specific\_h)):
4. if h[x] != specific\_h[x]:
5. specific\_h[x] = '?'
6. general\_h[x][x] = '?'
7. print(specific\_h)
8. print(specific\_h)

37

1. if target[i] == 'no':
2. for x in range(len(specific\_h)):
3. if h[x] != specific\_h[x]:
4. general\_h[x][x] = specific\_h[x]
5. else:
6. general\_h[x][x] = '?'

44

1. print('steps of candidate Elimation Algorithm ', i + 1)
2. print(specific\_h)
3. print(general\_h)

48

49 indeces = [i for i, val in enumerate(general\_h) if val ==

50 ['?', '?', '?', '?', '?', '?']]

51

52 for i in indeces:

53 general\_h.remove(['?', '?', '?', '?', '?', '?'])

54

55 return specific\_h, general\_h

56

57

1. s\_final, g\_final = learn(concepts, target)
2. print(' final answer \n')
3. print('final specific\_h: ', s\_final, sep='\n')
4. print('final general\_h: ', g\_final, sep='\n')

62

63

64

1

2 (4). Write a program to demonstrate the working of the decision tree based

3  ID3 algorithm . Use an appropriate data set for building the decision

4 tree and apply this knowledge to classify a new sample.

5

6

1. import math
2. import pandas as pd
3. from pprint import pprint
4. from collections import Counter

11

1. def entropy(probs):
2. return sum([-prob \* math.log(prob, 2) for prob in probs])

14

15

16 def entropy\_list(a\_list):

17

1. cnt = Counter(x for x in a\_list)
2. num\_instance = len(a\_list) \* 1.0
3. probs = [x / num\_instance for x in cnt.values()]
4. return entropy(probs)

22

23

24 def info\_gain(df, split, target, trace=0):

25

1. df\_split = df.groupby(split)
2. nobs = len(df.index) \* 1.0
3. df\_agg\_ent = df\_split.agg({target: [entropy\_list, lambda x: len(x) / nobs]})
4. df\_agg\_ent.columns = ["entropy", "propObserved"]

30

1. new\_entropy = sum(df\_agg\_ent["entropy"] \* df\_agg\_ent["propObserved"])
2. old\_entropy = entropy\_list(df[target])
3. return old\_entropy - new\_entropy

34

35

36 def id3(df, target, attribute\_name, default\_class=None):

37

1. cnt = Counter(x for x in df[target])
2. if len(cnt) == 1:
3. return next(iter(cnt))

41

1. elif df.empty or (not attribute\_name):
2. return default\_class

44

1. else:
2. default\_class = max(cnt.keys())
3. gains = [info\_gain(df, attr, target) for attr in attribute\_name]
4. index\_max = gains.index(max(gains))
5. best\_attr = attribute\_name[index\_max]
6. tree = {best\_attr: {}}
7. remaining\_attr = [x for x in attribute\_name if x != best\_attr]

52

1. for attr\_val, data\_subset in df.groupby(best\_attr):
2. subtree = id3(data\_subset, target, remaining\_attr, default\_class)
3. tree[best\_attr][attr\_val] = subtree

56

57 return tree

58

59

1. def classify(instance, tree, default=None):
2. attribute = next(iter(tree))
3. if instance[attribute] in tree[attribute].keys():
4. result = tree[attribute][instance[attribute]]
5. if isinstance(result, dict):
6. return classify(instance, result)
7. else:
8. return result
9. else:
10. return default

70

1. df\_tennis = pd.read\_csv('id3.csv')
2. print(df\_tennis)

73

1. attribute\_names = list(df\_tennis.columns)
2. attribute\_names.remove('PlayTennis')

76

77 tree = id3(df\_tennis, 'PlayTennis', attribute\_names)

78

1. print('\n\n The resultant decision tree is: \n\n')
2. pprint(tree)

F eli h/: o me m/ o h a mme d D/ e s k to p te/ mp b/ a c k P ro p a g a oti n p. y P a g e 1 o 1f

1

1. (5). Build an Artificial Neural Network by implementing the  Backpropagation 
2. algorithm and test the same using appropriate data sets.

4

5

6 import numpy as np

7

1. input\_neurons = 2
2. hidden\_layer\_neurons = 2
3. output\_neurons = 2

11

1. input\_ = np.random.randint(1, 100, input\_neurons)
2. output = np.array([1.0, 0.0])

14

1. hidden\_biass = np.random.rand(1, hidden\_layer\_neurons)
2. output\_biass = np.random.rand(1, output\_neurons)
3. hidden\_weight = np.random.rand(input\_neurons, hidden\_layer\_neurons)
4. output\_weight = np.random.rand(hidden\_layer\_neurons, output\_neurons)

19

20

1. def sigmoid(layer):
2. return 1 / (1 + np.exp(-layer))

23

24

1. def gradient(layer):
2. return layer \* (1 - layer)

27

28

29 for i in range(2000):

30

1. hidden\_layer = np.dot(input\_, hidden\_weight)
2. hidden\_layer = sigmoid(hidden\_layer + hidden\_biass)

33

1. output\_layer = np.dot(hidden\_layer, output\_weight)
2. output\_layer = sigmoid(output\_layer + output\_biass)

36

1. error = (output - output\_layer)
2. gradient\_outputLayer = gradient(output\_layer)

39

1. error\_terms\_output = gradient\_outputLayer \* error
2. error\_terms\_hidden = gradient(hidden\_layer) \*
3. np.dot(error\_terms\_output, output\_weight.T)

43

1. gradient\_hidden\_weights = np.dot(input\_.reshape(input\_neurons, 1),
2. error\_terms\_hidden.reshape(1, hidden\_layer\_neurons))
3. gradient\_output\_weights = np.dot(hidden\_layer.reshape(hidden\_layer\_neurons, 1),
4. error\_terms\_output.reshape(1, output\_neurons))

48

1. hidden\_weight = hidden\_weight + 0.05 \* gradient\_hidden\_weights
2. output\_weight = output\_weight + 0.05 \* gradient\_output\_weights

51

52 print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

1. print('Iteration: ', i, ':::', error)
2. print('####- output - #####', output\_layer)

55

56

57

58

F eli h/: o me m/ o h a mme d D/ e s k to p te/ mp b/ a si e p. y P a g e 1 o 1f

1

* 1. (6). Write a program to implement the  naïve Bayesian classifier  for a sample
  2. training data set stored as a .CSV file. Compute the accuracy of the
  3. classifier, considering few test data sets.

5

6

1. import pandas as pd
2. from sklearn.model\_selection import train\_test\_split
3. from sklearn.naive\_bayes import GaussianNB
4. from sklearn import metrics

11

1. df = pd.read\_csv("pima\_indian.csv")
2. feature\_col\_names = ['num\_preg', 'glucose\_conc', 'diastolic\_bp', 'thickness',
3. 'insulin', 'bmi', 'diab\_pred', 'age']
4. predicted\_class\_names = ['diabetes']

16

1. X = df[feature\_col\_names].values
2. y = df[predicted\_class\_names].values

19

20 xtrain, xtest, ytrain, ytest = train\_test\_split(X, y, test\_size=0.33)

21

1. print('\n the total number of Training Data :', ytrain.shape)
2. print('\n the total number of Test Data :', ytest.shape)

24

1. clf = GaussianNB().fit(xtrain, ytrain.ravel())
2. predicted = clf.predict(xtest)

27

28 predictTestData = clf.predict([[1, 189, 60, 23, 846, 30.1, 0.398, 59]])

29

1. print('\n Confusion matrix')
2. print(metrics.confusion\_matrix(ytest, predicted))

32

1. print('Accuracy of the classifier is', metrics.accuracy\_score(ytest, predicted))
2. print('The value of Precision', metrics.precision\_score(ytest, predicted))
3. print('The value of Recall', metrics.recall\_score(ytest, predicted))

36

37 print("Predicted Value for individual Test Data:", predictTestData)

38

39

40

41

F eli h/: o me m/ o h a mme d D/ e s k to p te/ mp K/ M- e a n s .p y P a g e 1 o 1f

1

1. (7). Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same
2. data set for clustering using  k-Means algorithm . Compare the results of
3. these two algorithms and comment on the quality of clustering. You can add
4. Java/Python ML library classes/API in the program.

6

7

8

1. import matplotlib.pyplot as plt
2. import numpy as np
3. import pandas as pd
4. import sklearn.metrics as metrics
5. from sklearn.cluster import KMeans
6. from sklearn.mixture import GaussianMixture

15

16 names = ['Sepal\_Length', 'Sepal\_Width', 'Petal\_Length', 'Petal\_Width', 'Class']

17

18 dataset = pd.read\_csv("Kdataset.csv", names=names)

19

20 X = dataset.iloc[:, :-1]

21

22 label = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}

23

24 y = [label[c] for c in dataset.iloc[:, -1]]

25

1. plt.figure(figsize=(14, 7))
2. colormap = np.array(['red', 'lime', 'black'])

28

29 # REAL PLOT

30

1. plt.subplot(1, 3, 1)
2. plt.title('Real')
3. plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y])

34

35 # K-PLOT

36

1. model = KMeans(n\_clusters=3, random\_state=0).fit(X)
2. plt.subplot(1, 3, 2)
3. plt.title('KMeans')
4. plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[model.labels\_])

41

1. print('The accuracy score K-Mean: ', metrics.accuracy\_score(y, model.labels\_))
2. print('The Confusion matrix K-Mean:\n', metrics.confusion\_matrix(y, model.labels\_))

44

45 # GMM PLOT

46

1. gmm = GaussianMixture(n\_components=3, random\_state=0).fit(X)
2. y\_cluster\_gmm = gmm.predict(X)

49

1. plt.subplot(1, 3, 3)
2. plt.title('GMM Classification')
3. plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y\_cluster\_gmm])

53

1. print('The accuracy score of EM: ', metrics.accuracy\_score(y, y\_cluster\_gmm))
2. print('The Confusion matrix of EM:\n ', metrics.confusion\_matrix(y, y\_cluster\_gmm))

56

57 plt.show()

58

59

60

F eli h/: o me m/ o h a mme d D/ e s k to p te/ mp K/ N N p. y P a g e 1 o 1f

1

1. (8). Write a program to implement  k-Nearest Neighbour  algorithm to classify
2. the iris data set.Print both correct and wrong predictions. Java/Python ML
3. library classes can be used for this problem.

5

1. import pandas as pd
2. from sklearn.neighbors import KNeighborsClassifier
3. from sklearn.model\_selection import train\_test\_split
4. from sklearn import metrics

10

11 names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

12

1. dataset = pd.read\_csv('Kdataset.csv')
2. X = dataset.iloc[:, :-1]
3. y = dataset.iloc[:, -1]

16

1. print('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
2. print(X.head())
3. print('Target value')
4. print(y.head())

21

1. Xtrain, Xtest, ytrain, ytest = train\_test\_split(X, y, test\_size=0.10)
2. classifier = KNeighborsClassifier(n\_neighbors=5).fit(Xtrain, ytrain)

24

25 ypred = classifier.predict(Xtest)

26

1. print("\n ")
2. print('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))
3. print(" ")

30

31 i = 0

32 for label in ytest:

33 print('%-25s %-25s' % (label, ypred[i]), end="")

1. if label == ypred[i]:
2. print(' %-25s' % 'Correct')
3. else:
4. print(' %-25s' % 'Wrong')

38 i = i + 1

39

1. print(" ")
2. print("\nConfusion Matrix:\n", metrics.confusion\_matrix(ytest, ypred))
3. print(" ")
4. print("\nClassification Report:\n", metrics.classification\_report(ytest, ypred))
5. print(" ")
6. print('Accuracy of the classifer is %0.2f' % metrics.accuracy\_score(ytest, ypred))
7. print(" ")

47

48

49

F eli h/: o me m/ o h a mme d D/ e s k to p te/ mp er/ g re s s oi n p. y P a g e 1 o 1f

1

1. (9). Implement the non-parametric  Locally Weighted Regression  algorithm in
2. order to fit data points. Select appropriate data set for your experiment
3. and draw graphs.

5

6

1. import numpy as np
2. import numpy as np1
3. import pandas as pd
4. import matplotlib.pyplot as plt

11

12

1. def kernel(point, xmat, k):
2. m, n = np.shape(xmat)
3. weights = np.mat(np1.eye((m)))
4. for j in range(m):
5. diff = point - X[j]
6. weights[j, j] = np.exp(diff \* diff.T / (-2.0 \* k \*\* 2))
7. return weights

20

21

1. def localWeight(point, xmat, ymat, k):
2. wei = kernel(point, xmat, k)
3. W = (X.T \* (wei \* X)).I \* (X.T \* (wei \* ymat.T))
4. return W

26

27

1. def localWeightRegression(xmat, ymat, k):
2. m, n = np.shape(xmat)
3. ypred = np.zeros(m)
4. for i in range(m):
5. ypred[i] = xmat[i] \* localWeight(xmat[i], xmat, ymat, k)
6. return ypred

34

35

1. # load data points
2. data = pd.read\_csv('10-dataset.csv')
3. bill = np.array(data.total\_bill)
4. tip = np.array(data.tip)

40

1. # preparing and add 1 in bill
2. mbill = np.mat(bill)
3. mtip = np.mat(tip)

44

1. m = np.shape(mbill)[1]
2. one = np.mat(np1.ones(m))
3. X = np.hstack((one.T, mbill.T))

48

1. # set k here
2. ypred = localWeightRegression(X, mtip, 0.5)
3. SortIndex = X[:, 1].argsort(0)
4. xsort = X[SortIndex][:, 0]

53

1. fig = plt.figure()
2. ax = fig.add\_subplot(1, 1, 1)

56

1. ax.scatter(bill, tip, color='green')
2. ax.plot(xsort[:, 1], ypred[SortIndex], color='red', linewidth=5)

59

1. plt.xlabel('Total bill')
2. plt.ylabel('Tip')
3. plt.show()

63

64

65