NAGARJUNA COLLEGE OF ENGINEERING AND TECHNOLOGY
(An Autonomous College under VTU)
VenkatagiriKote post, Devanahalli, Bengaluru-562164

Department of CSE (AIML)

Fundamentals of Machine Learning - LABORATORY MANUAL
21CII43

Prepared By:
Dr. R. Vijayanand
Associate Professor
Dept. Of CSE (AIML)

Course Objectives:

This course will enable students to:

- Understand the basic concepts of machine learning
- Understand the well posed learning techniques.
- Understand decision tree algorithms
- Learn Artificial Neural Networks with multilayer perceptron's.
- Understand Reinforcement learning concept

Course Outcomes:

At the end of the course, students should be able to

- Analyze the models using PAC framework, Rademacher complexity and VC Dimension.
- Solve the problems using support vector machines with suitable kernel models.
- Apply dimensionality reduction methods and reinforcement algorithms to solve the problems
- Implement the suitable regression and boosting algorithms based on the applications.
- Solve multiclass problems and evaluate with various ranking techniques.

Text Books:

- Foundations of machine learning, Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar.
- Understanding machine learning, Shai Shalev-Shwartz and Shai Ben-David

Reference Books:

1. Introduction to Artificial Neural Systems-J.M. Zurada, Jaico Publications 1994.
2. Artificial Neural Networks-B. Yegnanarayana, PHI, New Delhi 1998.

LIST OF EXPERIMENTS	
S.No	**Title**
1	Design a Simple 3 class dataset with 30 records and the features are represented in 4 columns. Apply rule based classification method to predict the classes in python.
2	Extract the input and output data from the CSV file using python and split the training and testing data in the ratio of 70:30
3	Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.
4	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.
5	Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.
6	Write a python program to demonstrate the working of C4.5 algorithm using a dataset for building the decision tree and validate it.
7	Build an Artificial neural network for identifying the classes of IRIS dataset and validate the results using confusion matrix in python
8	Design a back propagation algorithm to predict the class of the input data for a medical problem with multiple classes.
9	Consider an undirected graph with 8 points from 0 to 7, 0 ->1, 1 ->5, 1->2, 5->4, 5->6, 2->3 and 2->7. The bot is in position 0 and needs to position 7. Design a q-learning model to help the bot to reach the position 7
10	Design a qlearning based model to help the humanoid bot to reach village near the river. If it finds the desert, it is in wrong direction. The map is represented in an undirected graph as 0->1, 1->5, 5->6, 5->4, 1->2, 2->3 and 2->7, 7->8, 8->9. The bot current position is 0, the river is located in position 2 and the village is in 7. The position 4, 5, 6, 8 and 9 has the desert.

Experiment 1

Design a Simple 3 class dataset with 30 records and the features are represented in 4 columns. Apply rule based classification method to predict the classes in python.

Program

```
import pandas as pd
df = pd.read_csv("Ex1.csv")
#print(df.to_string())
display(df.iloc[1])
for i in range (30):
  if df["Color"].iloc[i] == "White" or "Black" or "BW" :
    if df["Sound"].iloc[i] > 79 and df["Sound"].iloc[i] < 101:
      if df["Tail Length"].iloc[i] > 0.6:
        if df["Height"].iloc[i] > 2.9:
          print("Horse")
  if df["Color"].iloc[i] == "White" or "Black" or "BW" or "Brown" or "BBrW":
    if df["Sound"].iloc[i] < 71:
      if df["Tail Length"].iloc[i] < 0.4:
        if df["Height"].iloc[i] < 1:
          print("Cat")
  if df["Color"].iloc[i] == "White" or "Black" or "BW" or "Brown" or "BBrW":
    if df["Sound"].iloc[i] > 100:
      if df["Tail Length"].iloc[i] > 0.4 and df["Tail Length"].iloc[i] < 0.7:
        if df["Height"].iloc[i] > 1 and df["Height"].iloc[i] < 2:
          print("Dog")
```

OUTPUT

Horse
Horse
Horse
Horse
Horse
Horse
Horse
Horse
Horse
Horse
Cat
Cat
Cat
Cat
Cat
Cat
Cat
Cat
Cat
Cat
Dog
Dog
Dog
Dog
Dog
Dog

Experiment 2

Extract the input and output data from the CSV file using python and split the training and testing data in the ratio of 70:30

Program

```
import pandas as pd
df = pd.read_csv("Ex1.csv")
from sklearn.model_selection import train_test_split
Features=["Color","Sound","Tail Length","Height"]
X=df.loc[:, Features]
Y=df.loc[:,['Class']]
X_train, X_test, y_train, y_test = train_test_split(X,Y, random_state=104,
                                test_size=0.25, shuffle=True)

print(X_train.head())
print(X_test.head())
print(y_train.head())
print(y_test.head())
```

Output

```
    Color  Sound  Tail Length  Height
17   Black     55          0.2     0.5
18   White     63          0.3     0.4
9    Black    100          1.0     3.0
19  Brown      73          0.2     0.2
32     BrW    115          0.6     1.5
    Color  Sound  Tail Length  Height
11   Black     50          0.2     0.6
30   White    134          0.6     1.4
10      BW     45          0.3     0.9
31  Brown     128          0.5     1.3
22      BW    120          0.5     1.2
   Class
17   Cat
18   Cat
9  Horse
19   Cat
32   Dog
  Class
11   Cat
30   Dog
10   Cat
31   Dog
22   Dog
```

Experiment 3

Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

Program

```
import pandas as pd
import numpy as np
d = pd.read_csv("Tennis.csv")
a = np.array(d)[:,:-1]
print(" The attributes are: ",a)
t = np.array(d)[:,-1]
print("The target is: ",t)
def train(c,t):
 for i, val in enumerate(t):
  if val == "Yes":
   specific_hypothesis = c[i].copy()
   pass
 for i, val in enumerate(c):
  if t[i] == "Yes":
   for x in range(len(specific_hypothesis)):
    if val[x] != specific_hypothesis[x]:
     specific_hypothesis[x] = '?'
    else:
     pass
   return specific_hypothesis
print(" The final hypothesis is:",train(a,t))
```

Output

```
The attributes are:  [['Sunny' 'Hot' 'High' 'Weak']
 ['Sunny' 'Hot' 'High' 'Strong']
 ['Overcast' 'Hot' 'High' 'Weak']
 ['Rain' 'Mild' 'High' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Strong']
 ['Overcast' 'Cool' 'Normal' 'Strong']
 ['Sunny' 'Mild' 'High' 'Weak']
 ['Sunny' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Mild' 'Normal' 'Weak']
 ['Sunny' 'Mild' 'Normal' 'Strong']
 ['Overcast' 'Mild' 'High' 'Strong']
 ['Overcast' 'Hot' 'Normal' 'Weak']
 ['Rain' 'Mild' 'High' 'Strong']]
The target is:  ['No' 'No' 'Yes' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes' 'Yes' 'Yes' 'Yes' 'Yes'  'No']
 The final hypothesis is: ['Overcast' 'Hot' '?' 'Weak']
```

Experiment 4

For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Program

```
import csv
# open the CSVFile and keep all rows as list of tuples
with open('EnjoySport.csv') as csvFile:
   examples = [tuple(line) for line in csv.reader(csvFile)]
print(examples)
# To obtain the domain of attribute values defined in the instances X
def get_domains(examples):
   # set function returns the unordered collection of items with no duplicates
   d = [set() for i in examples[0]]
   for x in examples:
      #Enumerate() function adds a counter to an iterable and returns it in a form of enumerate
object i.e(index,value)
      for i, xi in enumerate(x):
         d[i].add(xi)
   return [list(sorted(x)) for x in d]
# Test the get_domains function
get_domains(examples)

# Repeat the '?' and '0' length of domain no of times
def g_0(n):
   return ('?',)*n

def s_0(n):
   return ('0',)*n

# Function to check generality between two hypothesis
def more_general(h1, h2):
   more_general_parts = []
   for x, y in zip(h1, h2):
      mg = x == '?' or (x != '0' and (x == y or y == '0'))
      more_general_parts.append(mg)
   return all(more_general_parts) # Returns true if all elements of list or tuple are true

# Function to check whether train examples are consistent with hypothesis
def consistent(hypothesis,example):
   return more_general(hypothesis, example)

# Function to add min_generalizations
```

```
def min_generalizations(h, x):
    h_new = list(h)
    for i in range(len(h)):
        if not consistent(h[i:i+1],x[i:i+1]):
            if h[i] != '0':
                h_new[i] = '?'
            else:
                h_new[i] = x[i]
    return [tuple(h_new)]

# Function to generalize Specific hypto
def generalize_S(x, G, S):
    S_prev = list(S)
    for s in S_prev:
        if s not in S:
            continue
        if not consistent(s,x):
            S.remove(s)
            Splus = min_generalizations(s, x)
            # Keep only generalizations that have a counterpart in G
            S.update([h for h in Splus if any([more_general(g,h)
                                               for g in G])])
            # Remove from S any hypothesis more general than any other hypothesis in S
            S.difference_update([h for h in S if
                                 any([more_general(h, h1)
                                      for h1 in S if h != h1])])
    return S

# Function to add min_specializations
def min_specializations(h, domains, x):
    results = []
    for i in range(len(h)):
        if h[i] == '?':
            for val in domains[i]:
                if x[i] != val:
                    h_new = h[:i] + (val,) + h[i+1:]
                    results.append(h_new)
        elif h[i] != '0':
            h_new = h[:i] + ('0',) + h[i+1:]
            results.append(h_new)
    return results

# Function to specialize General hypotheses boundary
def specialize_G(x, domains, G, S):
```

```
    G_prev = list(G)
    for g in G_prev:
        if g not in G:
            continue
        if consistent(g,x):
            G.remove(g)
            Gminus = min_specializations(g, domains, x)
            # Keep only specializations that have a counterpart in S
            G.update([h for h in Gminus if any([more_general(h, s)
                                                for s in S])])
            # Remove hypothesis less general than any other hypothesis in G
            G.difference_update([h for h in G if
                                 any([more_general(g1, h)
                                      for g1 in G if h != g1])])
    return G

# Function to perform CandidateElimination
def candidate_elimination(examples):
    domains = get_domains(examples)[:-1]

    G = set([g_0(len(domains))])
    S = set([s_0(len(domains))])
    i=0
    print('All the hypotheses in General and Specific boundary are:\n')
    print('\n G[{0}]:'.format(i),G)
    print('\n S[{0}]:'.format(i),S)
    for xcx in examples:
        i=i+1
        x, cx = xcx[:-1], xcx[-1]  # Splitting data into attributes and decisions
        if cx=='Yes': # x is positive example
            G = {g for g in G if consistent(g,x)}
            S = generalize_S(x, G, S)
        else: # x is negative example
            S = {s for s in S if not consistent(s,x)}
            G = specialize_G(x, domains, G, S)
        print('\n G[{0}]:'.format(i),G)
        print('\n S[{0}]:'.format(i),S)
    return

candidate_elimination(examples)
```

Output

[('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'), ('Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'), ('Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No'), ('Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes')]

All the hypotheses in General and Specific boundary are:

G[0]: {('?', '?', '?', '?', '?', '?')}

S[0]: {('0', '0', '0', '0', '0', '0')}

G[1]: {('?', '?', '?', '?', '?', '?')}

S[1]: {('Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same')}

G[2]: {('?', '?', '?', '?', '?', '?')}

S[2]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}

G[3]: {('?', '?', '?', '?', '?', 'Same'), ('Sunny', '?', '?', '?', '?', '?'), ('?', 'Warm', '?', '?', '?', '?')}

S[3]: {('Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same')}

G[4]: {('Sunny', '?', '?', '?', '?', '?'), ('?', 'Warm', '?', '?', '?', '?')}

S[4]: {('Sunny', 'Warm', '?', 'Strong', '?', '?')}

Experiment 5

Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Program

Simple Decision Tree implementation

```
# Load libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
# load dataset
pima = pd.read_csv("diabetes.csv", header=None, names=col_names)
#split dataset in features and target variable
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
y = pima.label # Target variable
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70%
training and 30% test
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Output

Accuracy: 0.6666666666

Example 6

Write a python program to demonstrate the working of C4.5 algorithm using a dataset for building the decision tree and validate it.

Program

DT using Gini function

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# Function importing Dataset
def importdata():
    balance_data = pd.read_csv(
'https://archive.ics.uci.edu/ml/machine-learning-'+
'databases/balance-scale/balance-scale.data',
    sep= ',', header = None)

    # Printing the dataswet shape
    print ("Dataset Length: ", len(balance_data))
    print ("Dataset Shape: ", balance_data.shape)

    # Printing the dataset obseravtions
    print ("Dataset: ",balance_data.head())
    return balance_data

# Function to split the dataset
def splitdataset(balance_data):

    # Separating the target variable
    X = balance_data.values[:, 1:5]
    Y = balance_data.values[:, 0]

    # Splitting the dataset into train and test
    X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size = 0.3, random_state = 100)

    return X, Y, X_train, X_test, y_train, y_test
```

```
# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):

    # Creating the classifier object
    clf_gini = DecisionTreeClassifier(criterion = "gini",
            random_state = 100,max_depth=3, min_samples_leaf=5)

    # Performing training
    clf_gini.fit(X_train, y_train)
    return clf_gini

# Function to perform training with entropy.
def tarin_using_entropy(X_train, X_test, y_train):

    # Decision tree with entropy
    clf_entropy = DecisionTreeClassifier(
            criterion = "entropy", random_state = 100,
            max_depth = 3, min_samples_leaf = 5)

    # Performing training
    clf_entropy.fit(X_train, y_train)
    return clf_entropy

# Function to make predictions
def prediction(X_test, clf_object):

    # Predicton on test with giniIndex
    y_pred = clf_object.predict(X_test)
    print("Predicted values:")
    print(y_pred)
    return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

    print("Confusion Matrix: ",
        confusion_matrix(y_test, y_pred))

    print ("Accuracy : ",
    accuracy_score(y_test,y_pred)*100)

    print("Report : ",
    classification_report(y_test, y_pred))
```

```
# Driver code
def main():

    # Building Phase
    data = importdata()
    X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
    clf_gini = train_using_gini(X_train, X_test, y_train)
    clf_entropy = tarin_using_entropy(X_train, X_test, y_train)

    # Operational Phase
    print("Results Using Gini Index:")

    # Prediction using gini
    y_pred_gini = prediction(X_test, clf_gini)
    cal_accuracy(y_test, y_pred_gini)

    print("Results Using Entropy:")
    # Prediction using entropy
    y_pred_entropy = prediction(X_test, clf_entropy)
    cal_accuracy(y_test, y_pred_entropy)

# Calling main function
if __name__=="__main__":
    main()
```

OUTPUT

```
'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:  [[ 0  6  7]
 [ 0 67 18]
 [ 0 19 71]]
Accuracy :  73.40425531914893
Report :              precision    recall  f1-score   support

           B       0.00      0.00      0.00        13
           L       0.73      0.79      0.76        85
           R       0.74      0.79      0.76        90

    accuracy                           0.73       188
   macro avg       0.49      0.53      0.51       188
weighted avg       0.68      0.73      0.71       188
```

Results Using Entropy:
Predicted values:
['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'
 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix: [[0 6 7]
 [0 63 22]
 [0 20 70]]
Accuracy : 70.74468085106383
Report :

	precision	recall	f1-score	support
B	0.00	0.00	0.00	13
L	0.71	0.74	0.72	85
R	0.71	0.78	0.74	90
accuracy			0.71	188
macro avg	0.47	0.51	0.49	188
weighted avg	0.66	0.71	0.68	188

Example 7

Build an Artificial neural network for identifying the classes of IRIS dataset and validate the results using confusion matrix in python

Program

```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor

# Import necessary modules
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import r2_score


df = pd.read_csv('diabetes.csv')
print(df.shape)
df.describe().transpose()


target_column = ['Outcome']
predictors = list(set(list(df.columns))-set(target_column))
df[predictors] = df[predictors]/df[predictors].max()
df.describe().transpose()


X = df[predictors].values
y = df[target_column].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=40)
print(X_train.shape); print(X_test.shape)

from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(8,8,8), activation='relu', solver='adam',
max_iter=500)
mlp.fit(X_train,y_train)
```

```
predict_train = mlp.predict(X_train)
predict_test = mlp.predict(X_test)

from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_train,predict_train))
print(classification_report(y_train,predict_train))

print('Confusion matrix',confusion_matrix(y_test,predict_test))
print(classification_report(y_test,predict_test))
```

Output

```
(768, 9)
(537, 8)
(231, 8)
/usr/local/lib/python3.10/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:1098: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
[[318  40]
 [ 70 109]]
              precision    recall  f1-score   support

           0       0.82      0.89      0.85       358
           1       0.73      0.61      0.66       179

    accuracy                           0.80       537
   macro avg       0.78      0.75      0.76       537
weighted avg       0.79      0.80      0.79       537

Confusion matrix [[123  19]
 [ 39  50]]
              precision    recall  f1-score   support

           0       0.76      0.87      0.81       142
           1       0.72      0.56      0.63        89

    accuracy                           0.75       231
   macro avg       0.74      0.71      0.72       231
weighted avg       0.75      0.75      0.74       231
```

Example 8

Design a back propagation algorithm to predict the class of the input data for a medical problem with multiple classes.

Program

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# reading csv file and extracting class column to y.
x = pd.read_csv("iscxdata2.csv")
a = np.array(x)

X = a[:,1:75]
y = a[:,76] # classes having 0 and 1
X_train , X_test , y_train, y_test = train_test_split(X, y, random_state=0)
clf = MLPClassifier(hidden_layer_sizes=(2,7), random_state=5, verbose=True,
learning_rate_init=0.05)
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
print(confusion_matrix(y_test,y_pred))
```

Output

```
[305  11; 15    304]
```

Example 9

Consider an undirected graph with 8 points from 0 to 7, 0 ->1, 1 ->5, 1->2, 5->4, 5->6, 2->3 and 2->7. The bot is in position 0 and needs to position 7. Design a q-learning model to help the bot to reach the position 7

Program

```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar 28 12:59:23 2020

Assignment 2 - Agents and Reinforcement Learning

@author: Ronan Murphy - 15397831
"""


import numpy as np
import random
import matplotlib.pyplot as plt



#set the rows and columns length
BOARD_ROWS = 5
BOARD_COLS = 5

#initalise start, win and lose states
START = (0, 0)
WIN_STATE = (4, 4)
HOLE_STATE = [(1,0),(3,1),(4,2),(1,3)]

#class state defines the board and decides reward, end and next position
class State:
    def __init__(self, state=START):
        #initalise the state to start and end to false
        self.state = state
        self.isEnd = False

    def getReward(self):
        #give the rewards for each state -5 for loss, +1 for win, -1 for others
        for i in HOLE_STATE:
```

```
            if self.state == i:
                return -5
        if self.state == WIN_STATE:
            return 1

        else:
            return -1

    def isEndFunc(self):
        #set state to end if win/loss
        if (self.state == WIN_STATE):
            self.isEnd = True

        for i in HOLE_STATE:
            if self.state == i:
                self.isEnd = True

    def nxtPosition(self, action):
        #set the positions from current action - up, down, left, right
        if action == 0:
            nxtState = (self.state[0] - 1, self.state[1]) #up
        elif action == 1:
            nxtState = (self.state[0] + 1, self.state[1]) #down
        elif action == 2:
            nxtState = (self.state[0], self.state[1] - 1) #left
        else:
            nxtState = (self.state[0], self.state[1] + 1) #right


        #check if next state is possible
        if (nxtState[0] >= 0) and (nxtState[0] <= 4):
            if (nxtState[1] >= 0) and (nxtState[1] <= 4):
                    #if possible change to next state
                    return nxtState
        #Return current state if outside grid
        return self.state



#class agent to implement reinforcement learning through grid
class Agent:

    def __init__(self):
```

```
    #inialise states and actions
    self.states = []
    self.actions = [0,1,2,3]    # up, down, left, right
    self.State = State()
    #set the learning and greedy values
    self.alpha = 0.5
    self.gamma = 0.9
    self.epsilon = 0.1
    self.isEnd = self.State.isEnd

    # array to retain reward values for plot
    self.plot_reward = []

    #initalise Q values as a dictionary for current and new
    self.Q = {}
    self.new_Q = {}
    #initalise rewards to 0
    self.rewards = 0

    #initalise all Q values across the board to 0, print these values
    for i in range(BOARD_ROWS):
        for j in range(BOARD_COLS):
            for k in range(len(self.actions)):
                self.Q[(i, j, k)] =0
                self.new_Q[(i, j, k)] = 0

    print(self.Q)



#method to choose action with Epsilon greedy policy, and move to next state
def Action(self):
    #random value vs epsilon
    rnd = random.random()
    #set arbitraty low value to compare with Q values to find max
    mx_nxt_reward =-10
    action = None

    #9/10 find max Q value over actions
    if(rnd >self.epsilon) :
        #iterate through actions, find Q  value and choose best
        for k in self.actions:
```

```
            i,j = self.State.state

            nxt_reward = self.Q[(i,j, k)]

            if nxt_reward >= mx_nxt_reward:
                action = k
                mx_nxt_reward = nxt_reward

    #else choose random action
    else:
        action = np.random.choice(self.actions)

    #select the next state based on action chosen
    position = self.State.nxtPosition(action)
    return position,action


#Q-learning Algorithm
def Q_Learning(self,episodes):
    x = 0
    #iterate through best path for each episode
    while(x < episodes):
        #check if state is end
        if self.isEnd:
            #get current rewrard and add to array for plot
            reward = self.State.getReward()
            self.rewards += reward
            self.plot_reward.append(self.rewards)

            #get state, assign reward to each Q_value in state
            i,j = self.State.state
            for a in self.actions:
                self.new_Q[(i,j,a)] = round(reward,3)

            #reset state
            self.State = State()
            self.isEnd = self.State.isEnd

            #set rewards to zero and iterate to next episode
            self.rewards = 0
            x+=1
        else:
            #set to arbitrary low value to compare net state actions
```

```
            mx_nxt_value = -10
            #get current state, next state, action and current reward
            next_state, action = self.Action()
            i,j = self.State.state
            reward = self.State.getReward()
            #add reward to rewards for plot
            self.rewards +=reward

            #iterate through actions to find max Q value for action based on next state action
            for a in self.actions:
                nxtStateAction = (next_state[0], next_state[1], a)
                q_value = (1-self.alpha)*self.Q[(i,j,action)] + self.alpha*(reward +
self.gamma*self.Q[nxtStateAction])

                #find largest Q value
                if q_value >= mx_nxt_value:
                    mx_nxt_value = q_value

            #next state is now current state, check if end state
            self.State = State(state=next_state)
            self.State.isEndFunc()
            self.isEnd = self.State.isEnd

            #update Q values with max Q value for next state
            self.new_Q[(i,j,action)] = round(mx_nxt_value,3)

        #copy new Q values to Q table
        self.Q = self.new_Q.copy()
    #print final Q table output
    print(self.Q)

  #plot the reward vs episodes
  def plot(self,episodes):

    plt.plot(self.plot_reward)
    plt.show()


  #iterate through the board and find largest Q value in each, print output
  def showValues(self):
    for i in range(0, BOARD_ROWS):
      print('-------------------------------------------')
      out = '| '
```

```
            for j in range(0, BOARD_COLS):
                mx_nxt_value = -10
                for a in self.actions:
                    nxt_value = self.Q[(i,j,a)]
                    if nxt_value >= mx_nxt_value:
                        mx_nxt_value = nxt_value
                out += str(mx_nxt_value).ljust(6) + ' | '
            print(out)
        print('-------------------------------------------')



if __name__ == "__main__":
    #create agent for 10,000 episdoes implementing a Q-learning algorithm plot and show values.
    ag = Agent()
    episodes = 10000
    ag.Q_Learning(episodes)
    ag.plot(episodes)
    ag.showValues()
```

OUTPUT

{(0, 0, 0): 0, (0, 0, 1): 0, (0, 0, 2): 0, (0, 0, 3): 0, (0, 1, 0): 0, (0, 1, 1): 0, (0, 1, 2): 0, (0, 1, 3): 0, (0, 2, 0): 0, (0, 2, 1): 0, (0, 2, 2): 0, (0, 2, 3): 0, (0, 3, 0): 0, (0, 3, 1): 0, (0, 3, 2): 0, (0, 3, 3): 0, (0, 4, 0): 0, (0, 4, 1): 0, (0, 4, 2): 0, (0, 4, 3): 0, (1, 0, 0): 0, (1, 0, 1): 0, (1, 0, 2): 0, (1, 0, 3): 0, (1, 1, 0): 0, (1, 1, 1): 0, (1, 1, 2): 0, (1, 1, 3): 0, (1, 2, 0): 0, (1, 2, 1): 0, (1, 2, 2): 0, (1, 2, 3): 0, (1, 3, 0): 0, (1, 3, 1): 0, (1, 3, 2): 0, (1, 3, 3): 0, (1, 4, 0): 0, (1, 4, 1): 0, (1, 4, 2): 0, (1, 4, 3): 0, (2, 0, 0): 0, (2, 0, 1): 0, (2, 0, 2): 0, (2, 0, 3): 0, (2, 1, 0): 0, (2, 1, 1): 0, (2, 1, 2): 0, (2, 1, 3): 0, (2, 2, 0): 0, (2, 2, 1): 0, (2, 2, 2): 0, (2, 2, 3): 0, (2, 3, 0): 0, (2, 3, 1): 0, (2, 3, 2): 0, (2, 3, 3): 0, (2, 4, 0): 0, (2, 4, 1): 0, (2, 4, 2): 0, (2, 4, 3): 0, (3, 0, 0): 0, (3, 0, 1): 0, (3, 0, 2): 0, (3, 0, 3): 0, (3, 1, 0): 0, (3, 1, 1): 0, (3, 1, 2): 0, (3, 1, 3): 0, (3, 2, 0): 0, (3, 2, 1): 0, (3, 2, 2): 0, (3, 2, 3): 0, (3, 3, 0): 0, (3, 3, 1): 0, (3, 3, 2): 0, (3, 3, 3): 0, (3, 4, 0): 0, (3, 4, 1): 0, (3, 4, 2): 0, (3, 4, 3): 0, (4, 0, 0): 0, (4, 0, 1): 0, (4, 0, 2): 0, (4, 0, 3): 0, (4, 1, 0): 0, (4, 1, 1): 0, (4, 1, 2): 0, (4, 1, 3): 0, (4, 2, 0): 0, (4, 2, 1): 0, (4, 2, 2): 0, (4, 2, 3): 0, (4, 3, 0): 0, (4, 3, 1): 0, (4, 3, 2): 0, (4, 3, 3): 0, (4, 4, 0): 0, (4, 4, 1): 0, (4, 4, 2): 0, (4, 4, 3): 0}

{(0, 0, 0): -5.735, (0, 0, 1): -5.499, (0, 0, 2): -5.735, (0, 0, 3): -5.262, (0, 1, 0): -5.262, (0, 1, 1): -4.736, (0, 1, 2): -5.735, (0, 1, 3): -4.736, (0, 2, 0): -4.736, (0, 2, 1): -4.152, (0, 2, 2): -5.262, (0, 2, 3): -4.153, (0, 3, 0): -4.152, (0, 3, 1): -5.489, (0, 3, 2): -4.735, (0, 3, 3): -3.504, (0, 4, 0): -3.496, (0, 4, 1): -2.783, (0, 4, 2): -4.143, (0, 4, 3): -3.503, (1, 0, 0): -5, (1, 0, 1): -5, (1, 0, 2): -5, (1, 0, 3): -5, (1, 1, 0): -5.262, (1, 1, 1): -4.152, (1, 1, 2): -5.499, (1, 1, 3): -4.152, (1, 2, 0): -4.736, (1, 2, 1): -3.503, (1, 2, 2): -4.736, (1, 2, 3): -5.499, (1, 3, 0): -5, (1, 3, 1): -5, (1, 3, 2): -5, (1, 3, 3): -5, (1, 4, 0): -3.501, (1, 4, 1): -1.982, (1, 4, 2): -5.498, (1, 4, 3): -2.783, (2, 0, 0): -5.156, (2, 0, 1): -4.252, (2, 0, 2): -4.176, (2, 0, 3): -4.14, (2, 1, 0): -4.691, (2, 1, 1): -5.499, (2, 1, 2): -4.641, (2, 1, 3): -

3.503, (2, 2, 0): -4.152, (2, 2, 1): -2.782, (2, 2, 2): -4.152, (2, 2, 3): -2.782, (2, 3, 0): -5.499, (2, 3, 1): -1.98, (2, 3, 2): -3.503, (2, 3, 3): -1.981, (2, 4, 0): -2.783, (2, 4, 1): -1.091, (2, 4, 2): -2.782, (2, 4, 3): -1.981, (3, 0, 0): -4.336, (3, 0, 1): -4.245, (3, 0, 2): -4.389, (3, 0, 3): -4.125, (3, 1, 0): -5, (3, 1, 1): -5, (3, 1, 2): -5, (3, 1, 3): -5, (3, 2, 0): -3.42, (3, 2, 1): -5.481, (3, 2, 2): -5.499, (3, 2, 3): -1.98, (3, 3, 0): -2.782, (3, 3, 1): -1.089, (3, 3, 2): -2.782, (3, 3, 3): -1.09, (3, 4, 0): -1.981, (3, 4, 1): -0.1, (3, 4, 2): -1.98, (3, 4, 3): -1.09, (4, 0, 0): -4.426, (4, 0, 1): -4.209, (4, 0, 2): -4.245, (4, 0, 3): -4.262, (4, 1, 0): -4.125, (4, 1, 1): -4.297, (4, 1, 2): -4.276, (4, 1, 3): -4.125, (4, 2, 0): -5, (4, 2, 1): -5, (4, 2, 2): -5, (4, 2, 3): -5, (4, 3, 0): -1.98, (4, 3, 1): -1.089, (4, 3, 2): -5.499, (4, 3, 3): -0.099, (4, 4, 0): 1, (4, 4, 1): 1, (4, 4, 2): 1, (4, 4, 3): 1}

-5.262	-4.736	-4.152	-3.504	-2.783
-5	-4.152	-3.503	-5	-1.982
-4.14	-3.503	-2.782	-1.98	-1.091
-4.125	-5	-1.98	-1.089	-0.1
-4.209	-4.125	-5	-0.099	1

Example 10

Design a qlearning based model to help the humanoid bot to reach village near the river. If it finds the desert, it is in wrong direction. The map is represented in an undirected graph as 0->1, 1->5, 5->6, 5->4, 1->2, 2->3 and 2->7, 7->8, 8->9. The bot current position is 0, the river is located in position 2 and the village is in 7. The position 4, 5, 6, 8 and 9 has the desert.

Program

```
from get_dict import get_dict
from get_R_Q import initial_R
from get_R_Q import initial_Q
from get_result import get_result
import pandas as pd
import time
data = pd.read_csv("graph_1.csv")
graph = get_dict(data)
A = graph["A"]
Z = graph["Z"]
weight = graph["weight"]
A_Z_dict = graph["A_Z_dict"]
start = 1
end = [9]
R = initial_R(A,Z,weight,A_Z_dict)
Q = initial_Q(R)
alpha = 0.7 # learning rate
epsilon = 0.1 #greedy policy
n_episodes = 1000
time0 = time.time()
result = get_result(R,Q,alpha,epsilon,n_episodes,start,end)
print("time is:",time.time() - time0)
print(result["ends_find"])
print(result["cost"])
print(result["routes_number"])
```

OUTPUT

```
loop: 0
nodes: [0, 0, 9]
nodes: [0, 0, 9, 4]
nodes: [0, 0, 9, 4, 4]
nodes: [0, 0, 9, 4, 4, 4]
time is: 0.014706611633300781
[7]
{18: 1}
 {7: 1}
```