Practical 1 A: Breadth First Search

RMP.py > ...

```python
dict_gn = dict(
        Arad = dict(Zerind=75,Timisoara=118,Sibiu=140),
        Bucharest = dict(Urziceni=85,Giurgiu=90,Pitesti=101,Fagaras=211),
        Craiova = dict(Drobeta=120,Pitesti=138,Rimnicu=146),
        Drobeta = dict(Mehadia=75,Craiova=120),
        Eforie = dict(Hirsova=86),
        Fagaras = dict(Sibiu=99,Bucharest=211),
        Giurgiu = dict(Bucharest=90),
        Hirsova = dict(Eforie=86,Urziceni=98),
        Iasi = dict(Neamt=87,Vaslui=92),
        Lugoj = dict(Mehadia=70,Timisoara=111),
        Mehadia = dict(Lugoj=70,Drobeta=75),
        Neamt = dict(Iasi=87),
        Oradea = dict(Zerind=71,Sibiu=151),
        Pitesti = dict(Rimnicu=97,Bucharest=101,Craiova=138),
        Rimnicu = dict(Sibiu=80,Pitesti=97,Craiova=146),
        Sibiu = dict(Rimnicu=80,Fagaras=99,Arad=140,Oradea=151),
        Timisoara = dict(Lugoj=111,Arad=118),
        Urziceni = dict(Bucharest=85,Hirsova=98,Vaslui=142),
        Vaslui = dict(Iasi=92,Urziceni=142),
        Zerind = dict(Oradea=71,Arad=75)
)
```

practical_1.py > ...

```python
import queue as Q
from RMP import dict_gn
start = 'Arad'
goal = "Bucharest"
result=''
def BFS(city,cityq,visitedq):
    global result
    if city==start:
        result = result + "" + city
    for eachcity in dict_gn[city].keys():
        if eachcity==goal:
            result = result + " " + eachcity
            return
        if eachcity not in cityq.queue and eachcity not in visitedq.queue:
            cityq.put(eachcity)
            result = result + " " + eachcity
    visitedq.put(city)
    BFS(cityq.get(),cityq,visitedq)
def main():
    cityq = Q.Queue()
    visitedq = Q.Queue()
    BFS(start,cityq,visitedq)
    print("BFS Traversal From ", start," to " , goal, "is :")
    print(result)
main()
```

```
pwsh  D:\College\TYCS\AI
python -u "d:\College\TYCS\AI\practical_1.py"
BFS Traversal From  Arad  to  Bucharest is :
Arad Zerind Timisoara Sibiu Oradea Lugoj Rimnicu Fagaras Mehadia Pitesti Craiova Bucharest
```

Practical 1 B: Iterative Depth First Search

practical_1B.py > ...

```python
import queue as Q
from RMP import dict_gn
start = "Arad"
goal = "Bucharest"
result = ""

def DLS(city,visitedstack,startlimit,endlimit):
    global result
    found = 0
    result = result + city + " "
    visitedstack.append(city)
    if city == goal:
        return 1
    if startlimit == endlimit:
        return 0
    for eachcity in dict_gn[city].keys():
        if eachcity not in visitedstack:
            found = DLS(eachcity,visitedstack,startlimit+1,endlimit)
        if found:
            return found

def IDDFS(city,visitedstack,endlimit):
    global result
    for i in range(0,endlimit):
        print("Seaching at Limit:", i)
        found = DLS(city,visitedstack, 0 , i)
        if found:
            print("Found")
            break
        else:
            print("Not Found!")
            print(result)
            print("______")
            result=""
            visitedstack = []

def main():
    visitedstack = []
    IDDFS(start,visitedstack,9)
    print("IDDFS Traversal from ", start, " to ",goal," is:")
    print(result)
main()
```

```
pwsh  D:\College\TYCS\AI                                                    22:29:48
> python -u "d:\College\TYCS\AI\practical_1B.py"
Seaching at Limit: 0
Not Found!
Arad

______
Seaching at Limit: 1
Not Found!
Arad Zerind Timisoara Sibiu

______
Seaching at Limit: 2
Not Found!
Arad Zerind Oradea Timisoara Lugoj Sibiu Rimnicu Fagaras

______
Seaching at Limit: 3
Not Found!
Arad Zerind Oradea Sibiu Timisoara Lugoj Mehadia

______
Seaching at Limit: 4
Not Found!
Arad Zerind Oradea Sibiu Rimnicu Fagaras Timisoara Lugoj Mehadia Drobeta

______
Seaching at Limit: 5
Found
IDDFS Traversal from  Arad  to  Bucharest  is:
Arad Zerind Oradea Sibiu Rimnicu Pitesti Craiova Fagaras Bucharest
```

Practical 2 A: A* Search

RMP.py > ...

```python
dict_hn={
        'Arad':336,'Bucharest':0,'Craiova':160,'Drobeta':242,'Eforie':161,
        'Fagaras':176,'Giurgiu':77,'Hirsova':151,'Iasi':226,'Lugoj':244,
        'Mehadia':241,'Neamt':234,'Oradea':380,'Pitesti':100,'Rimnicu':193,
        'Sibiu':253,'Timisoara':329,'Urziceni':80,'Vaslui':199,'Zerind':374
}

dict_gn = dict(
        Arad = dict(Zerind=75,Timisoara=118,Sibiu=140),
        Bucharest = dict(Urziceni=85,Giurgiu=90,Pitesti=101,Fagaras=211),
        Craiova = dict(Drobeta=120,Pitesti=138,Rimnicu=146),
        Drobeta = dict(Mehadia=75,Craiova=120),
        Eforie = dict(Hirsova=86),
        Fagaras = dict(Sibiu=99,Bucharest=211),
        Giurgiu = dict(Bucharest=90),
        Hirsova = dict(Eforie=86,Urziceni=98),
        Iasi = dict(Neamt=87,Vaslui=92),
        Lugoj = dict(Mehadia=70,Timisoara=111),
        Mehadia = dict(Lugoj=70,Drobeta=75),
        Neamt = dict(Iasi=87),
        Oradea = dict(Zerind=71,Sibiu=151),
        Pitesti = dict(Rimnicu=97,Bucharest=101,Craiova=138),
        Rimnicu = dict(Sibiu=80,Pitesti=97,Craiova=146),
        Sibiu = dict(Rimnicu=80,Fagaras=99,Arad=140,Oradea=151),
        Timisoara = dict(Lugoj=111,Arad=118),
        Urziceni = dict(Bucharest=85,Hirsova=98,Vaslui=142),
        Vaslui = dict(Iasi=92,Urziceni=142),
        Zerind = dict(Oradea=71,Arad=75)
)
```

practical_2.py > ...

```python
import queue as Q
from RMP import dict_gn
from RMP import dict_hn

start = 'Arad'
goal = 'Bucharest'
result = ''

def get_fn(citystr):
    cities=citystr.split(",")
    hn=gn=0
    for ctr in range(0, len(cities)-1):
        gn=gn+dict_gn[cities[ctr]][cities[ctr+1]]
    hn=dict_hn[cities[len(cities)-1]]
    return(hn+gn)

def expand(cityq):
    global result
    tot, citystr, thiscity=cityq.get()
    if thiscity==goal:
        result=citystr+"::"+str(tot)
        return
    for cty in dict_gn[thiscity]:
        cityq.put((get_fn(citystr+","+cty),citystr+","+cty,cty))
    expand(cityq)

def main():
    cityq=Q.PriorityQueue()
    thiscity=start
    cityq.put((get_fn(start),start,thiscity))
    expand(cityq)
    print("The A* path with the total is: ")
    print(result)

main()
```

```
pwsh  D:\College\TYCS\AI
> python -u "d:\College\TYCS\AI\practical_2.py"
The A* path with the total is:
Arad,Sibiu,Rimnicu,Pitesti,Bucharest::418
```

Practical 2 B: Recursive Best -First Search

practical_2B.py > ...

```python
import queue as Q
from RMP import dict_gn
from RMP import dict_hn

start = 'Arad'
goal = 'Bucharest'
result = ''

def get_fn(citystr):
    cities=citystr.split(",")
    hn=gn=0
    for ctr in range(0, len(cities)-1):
        gn=gn+dict_gn[cities[ctr]][cities[ctr+1]]
    hn=dict_hn[cities[len(cities)-1]]
    return(hn+gn)

def printout(cityq):
    for i in range(0, cityq.qsize()):
        print(cityq.queue[i])

def expand(cityq):
    global result
    tot, citystr, thiscity = cityq.get()
    nexttot = 999
    if not cityq.empty():
        nexttot,nextcitystr,nextthiscity=cityq.queue[0]
    if thiscity== goal and tot < nexttot:
        result = citystr + "::" + str(tot)
        return
    print("Expaded city ---------", thiscity)
    print("second best f(n)---------", nexttot)
    tempq = Q.PriorityQueue()
    for cty in dict_gn[thiscity]:
        tempq.put((get_fn(citystr+','+cty), citystr+','+cty, cty))
    for ctr in range(1,3):
        ctrtot, ctrcitystr ,ctrthiscity = tempq.get()
        if ctrtot < nexttot:
            cityq.put((ctrtot, ctrcitystr,ctrthiscity))
        else:
            cityq.put((ctrtot, citystr, thiscity))
            break
    printout(cityq)
    expand(cityq)
def main():
    cityq=Q.PriorityQueue()
    thiscity=start
    cityq.put((999, "NA", "NA"))
    cityq.put((get_fn(start), start, thiscity))
    expand(cityq)
    print(result)

main()
```

```
pwsh  D:\College\TYCS\AI
> python -u "d:\College\TYCS\AI\practical_2B.py"
Expaded city --------- Arad
second best f(n)--------- 999
(393, 'Arad,Sibiu', 'Sibiu')
(999, 'NA', 'NA')
(447, 'Arad,Timisoara', 'Timisoara')
Expaded city --------- Sibiu
second best f(n)--------- 447
(413, 'Arad,Sibiu,Rimnicu', 'Rimnicu')
(415, 'Arad,Sibiu,Fagaras', 'Fagaras')
(447, 'Arad,Timisoara', 'Timisoara')
(999, 'NA', 'NA')
Expaded city --------- Rimnicu
second best f(n)--------- 415
(415, 'Arad,Sibiu,Fagaras', 'Fagaras')
(417, 'Arad,Sibiu,Rimnicu', 'Rimnicu')
(447, 'Arad,Timisoara', 'Timisoara')
(999, 'NA', 'NA')
Expaded city --------- Fagaras
second best f(n)--------- 417
(417, 'Arad,Sibiu,Rimnicu', 'Rimnicu')
(450, 'Arad,Sibiu,Fagaras', 'Fagaras')
(447, 'Arad,Timisoara', 'Timisoara')
(999, 'NA', 'NA')
Expaded city --------- Rimnicu
second best f(n)--------- 447
(417, 'Arad,Sibiu,Rimnicu,Pitesti', 'Pitesti')
(447, 'Arad,Timisoara', 'Timisoara')
(999, 'NA', 'NA')
(450, 'Arad,Sibiu,Fagaras', 'Fagaras')
(526, 'Arad,Sibiu,Rimnicu', 'Rimnicu')
Expaded city --------- Pitesti
second best f(n)--------- 447
(418, 'Arad,Sibiu,Rimnicu,Pitesti,Bucharest', 'Bucharest')
(447, 'Arad,Timisoara', 'Timisoara')
(607, 'Arad,Sibiu,Rimnicu,Pitesti', 'Pitesti')
(526, 'Arad,Sibiu,Rimnicu', 'Rimnicu')
(450, 'Arad,Sibiu,Fagaras', 'Fagaras')
(999, 'NA', 'NA')
Arad,Sibiu,Rimnicu,Pitesti,Bucharest::418
```

Practical 3: Decision Tree Learning

[1]:
```
import numpy as np
import pandas as pd
```

[2]:
```
PlayTennis = pd.read_csv("C:/Users/dines/Downloads/jupyter_download_files/playTennis/playTennis.csv")
```

[3]:
```
PlayTennis
```

[3]:

	Outlook	Temperature	Humidity	Wind	Play Tennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rain	Mild	High	Strong	No

[4]:
```
from sklearn.preprocessing import LabelEncoder
Le = LabelEncoder()

PlayTennis['Outlook'] = Le.fit_transform(PlayTennis['Outlook'])
PlayTennis['Temperature'] = Le.fit_transform(PlayTennis['Temperature'])
PlayTennis['Humidity'] = Le.fit_transform(PlayTennis['Humidity'])
PlayTennis['Wind'] = Le.fit_transform(PlayTennis['Wind'])
PlayTennis['Play Tennis'] = Le.fit_transform(PlayTennis['Play Tennis'])
```

[4]:
```
from sklearn.preprocessing import LabelEncoder
Le = LabelEncoder()

PlayTennis['Outlook'] = Le.fit_transform(PlayTennis['Outlook'])
PlayTennis['Temperature'] = Le.fit_transform(PlayTennis['Temperature'])
PlayTennis['Humidity'] = Le.fit_transform(PlayTennis['Humidity'])
PlayTennis['Wind'] = Le.fit_transform(PlayTennis['Wind'])
PlayTennis['Play Tennis'] = Le.fit_transform(PlayTennis['Play Tennis'])
```

[5]:
```
PlayTennis
```

[5]:

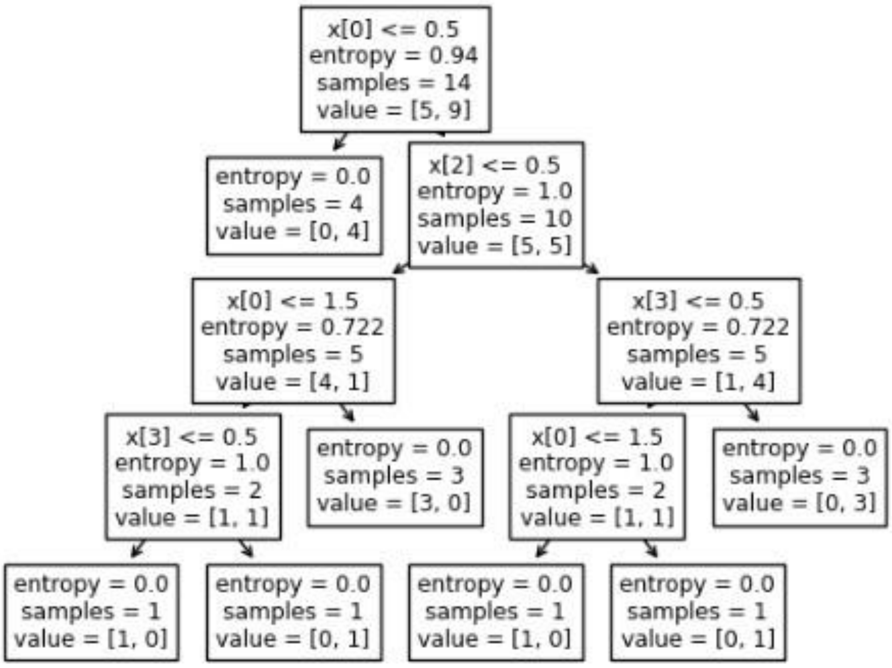
	Outlook	Temperature	Humidity	Wind	Play Tennis
0	2	1	0	1	0
1	2	1	0	0	0
2	0	1	0	1	1
3	1	2	0	1	1
4	1	0	1	1	1
5	1	0	1	0	0
6	0	0	1	0	1
7	2	2	0	1	0
8	2	0	1	1	1
9	1	2	1	1	1
10	2	2	1	0	1
11	0	2	0	0	1
12	0	1	1	1	1
13	1	2	0	0	0

[6]:
```
y = PlayTennis['Play Tennis']
x = PlayTennis.drop(['Play Tennis'],axis=1)
```

[7]:
```
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion = 'entropy')
clf = clf.fit(x, y)
```

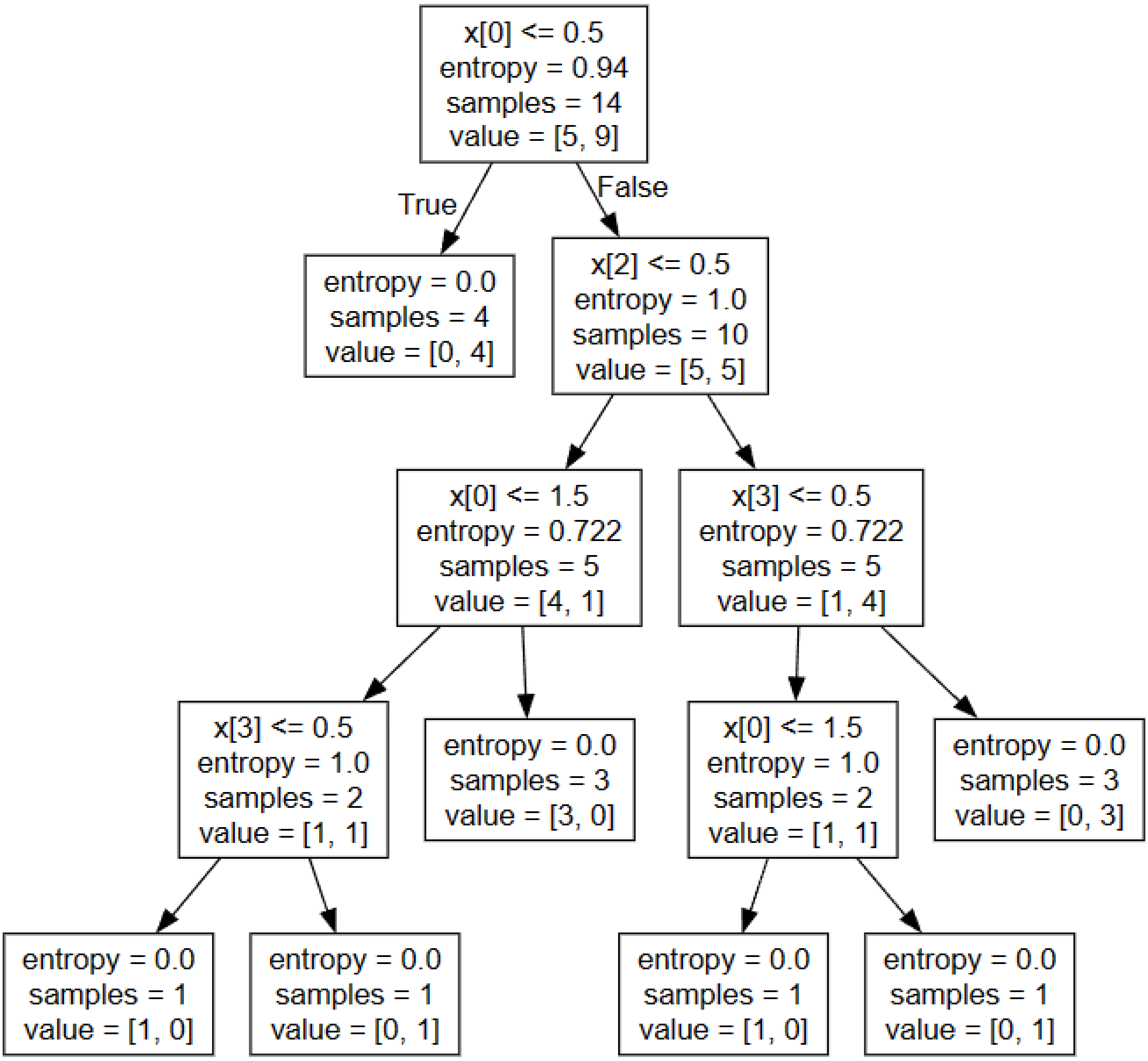
[8]:
```
tree.plot_tree(clf)
```

[8]:
```
[Text(0.4444444444444444, 0.9, 'x[0] <= 0.5\nentropy = 0.94\nsamples = 14\nvalue = [5, 9]'),
 Text(0.3333333333333333, 0.7, 'entropy = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.5555555555555556, 0.7, 'x[2] <= 0.5\nentropy = 1.0\nsamples = 10\nvalue = [5, 5]'),
 Text(0.3333333333333333, 0.5, 'x[0] <= 1.5\nentropy = 0.722\nsamples = 5\nvalue = [4, 1]'),
 Text(0.2222222222222222, 0.3, 'x[3] <= 0.5\nentropy = 1.0\nsamples = 2\nvalue = [1, 1]'),
 Text(0.1111111111111111, 0.1, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.3333333333333333, 0.1, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.4444444444444444, 0.3, 'entropy = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.7777777777777778, 0.5, 'x[3] <= 0.5\nentropy = 0.722\nsamples = 5\nvalue = [1, 4]'),
 Text(0.6666666666666666, 0.3, 'x[0] <= 1.5\nentropy = 1.0\nsamples = 2\nvalue = [1, 1]'),
 Text(0.5555555555555556, 0.1, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.7777777777777778, 0.1, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.8888888888888888, 0.3, 'entropy = 0.0\nsamples = 3\nvalue = [0, 3]')]
```



[9]:
```
import graphviz
dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
graph
```

[9]:



[10]:
```
X_pred = clf.predict(x)
```

[11]:
```
X_pred == y
```

[11]:
```
0     True
1     True
2     True
3     True
4     True
5     True
6     True
7     True
8     True
9     True
10    True
11    True
12    True
13    True
Name: Play Tennis, dtype: bool
```

Practical 4: Feed Forward Backpropagation Neural Network

practical_4B.py > ...

```python
from doctest import OutputChecker
import numpy as np

class NeuralNetwork():
  def __init__(self):
    np.random.seed()
    self.synaptic_weights=2*np.random.random((3,1))-1

  def sigmoid(self,x):
    return 1/(1+np.exp(-x))

  def sigmoid_derivative(self,x):
    return x*(1-x)

  def train(self,training_inputs,training_outputs,training_iterations):
    for iteration in range(training_iterations):
        output=self.think(training_inputs)
        error = training_outputs-output
        adjustments=np.dot(training_inputs.T,error*self.sigmoid_derivative(output))
        self.synaptic_weights +=adjustments

  def think(self,inputs):
    inputs=inputs.astype(float)
    output=self.sigmoid(np.dot(inputs,self.synaptic_weights))
    return output

if __name__ == "__main__":
  neural_network = NeuralNetwork()
  print("Beginning Randomly Generated Weights: ")
  print(neural_network.synaptic_weights)

  training_inputs = np.array([[0,0,1],
                              [1,1,1],
                              [1,0,1],
                              [0,1,1]])

  training_outputs = np.array([[0,1,1,0]]).T

  neural_network.train(training_inputs, training_outputs, 15000)
  print("Ending Weights After Training: ")
  print(neural_network.synaptic_weights)
  user_input_one = str(input("User Input One: "))
  user_input_two = str(input("User Input Two: "))
  user_input_three = str(input("User Input Three: "))
  print("Considering New Situation: ", user_input_one, user_input_two, user_input_three)
  print("New Output data: ")
  print(neural_network.think(np.array([user_input_one, user_input_two, user_input_three])))
```

```
pwsh  D:\College\TYCS\AI
> python -u "d:\College\TYCS\AI\practical_4B.py"
Beginning Randomly Generated Weights:
[[-0.79779341]
 [-0.76420848]
 [ 0.83210476]]
Ending Weights After Training:
[[10.0870228 ]
 [-0.20772497]
 [-4.83692503]]
User Input One: 2
User Input Two: 3
User Input Three: 2
Considering New Situation:  2 3 2
New Output data:
[0.99994866]
```

Practical 5: Support Vector Machine

[1]:
```
# from warnings import filterwarnings
```

[2]:
```
# pip install skompiler
```

[3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn.linear_model import LogisticRegression , LogisticRegressionCV
from sklearn.metrics import mean_squared_error , r2_score
from sklearn.model_selection import train_test_split , cross_val_score , cross_val_predict, GridSearchCV
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import scale
from sklearn import model_selection
from sklearn.metrics import roc_auc_score , roc_curve
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix , accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier , BaseEnsemble, GradientBoostingClassifier
from sklearn.svm import SVC, LinearSVC
import time
from matplotlib.colors import ListedColormap
from xgboost import XGBRegressor
from skompiler import skompile
from lightgbm import LGBMRegressor
```

[4]:
```
pd.set_option('display.max_rows',1000)
pd.set_option('display.max_columns',1000)
pd.set_option('display.width',1000)
```

---

[5]:
```
df = pd.read_csv('D:\AI_Datasets\diabetes.csv')
df.head()
```

[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

[6]:
```
df.shape
```

[6]: (768, 9)

[7]:
```
df.describe()
```

[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

[8]:
```
x = df.drop("Outcome",axis=1)
y = df["Outcome"]
```

[9]:
```
x_train = x.iloc[:600]
x_test = x.iloc[600:]
y_train = y[:600]
y_test = y[600:]

print("x_train shape : ", x_train.shape)
print("x_test shape : ", x_test.shape)
print("y_train shape : ", y_train.shape)
print("y_test shape : ", y_test.shape)
```

```
x_train shape :  (600, 8)
x_test shape :  (168, 8)
y_train shape :  (600,)
y_test shape :  (168,)
```

[10]:
```
support_vector_classifier = SVC(kernel = "linear").fit(x_train , y_train)
```

[11]:
```
support_vector_classifier
```

[11]:
```
SVC
SVC(kernel='linear')
```

[12]:
```
support_vector_classifier.C
```

[12]:
```
1.0
```

[13]:
```
support_vector_classifier
```

[13]:
```
SVC
SVC(kernel='linear')
```

[14]:
```
y_pred = support_vector_classifier.predict(x_test)
```

[15]:
```
cm = confusion_matrix(y_test, y_pred)
```

[16]:
```
cm
```

[16]:
```
array([[96, 12],
       [27, 33]], dtype=int64)
```

[17]:
```
print("Our Auucracy is : ", (cm[0][0]+cm[1][1]/(cm[0][0]+cm[1][1]+cm[0][1]+cm[1][0])))
```

```
Our Auucracy is :  96.19642857142857
```

[18]:
```
accuracy_score(y_test,y_pred)
```

[18]:
```
0.7678571428571429
```

[19]:
```
print(classification_report(y_test , y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.89      0.83       108
           1       0.73      0.55      0.63        60

    accuracy                           0.77       168
   macro avg       0.76      0.72      0.73       168
weighted avg       0.76      0.77      0.76       168
```

[20]:
```
support_vector_classifier
```

[20]:
```
SVC
SVC(kernel='linear')
```

[21]:
```
accuracies = cross_val_score(estimator = support_vector_classifier ,
                             X = x_train, y=y_train,
                             cv=10)
print("Average Accuracy : {:.2f} %".format(accuracies.mean()*100))
print("Standart Deviation of Accuracies : {:.2f} %".format(accuracies.std()*100))
```

```
Average Accuracy : 77.33 %
Standart Deviation of Accuracies : 4.90 %
```

[22]:
```
support_vector_classifier.predict(x_test)[:10]
```

[22]:
```
array([0, 0, 0, 1, 1, 0, 1, 0, 1, 0], dtype=int64)
```

[23]:
```
svm_params = {"C" : np.arange(1,20)}
```

[24]:
```
svm = SVC(kernel="linear")
svm_cv = GridSearchCV(svm, svm_params,cv=8)
```

[25]:
```
start_time = time.time()

svm_cv.fit(x_train, y_train)

elapsed_time = time.time() - start_time

print(f"Elapsed time for support vector regerssion cross validation :" f"{elapsed_time:.3f} seconds")
```

```
Elapsed time for support vector regerssion cross validation :3729.608 seconds
```

[26]:
```
svm_cv.best_score_
```

[26]:
```
0.7716666666666667
```

[27]:
```
svm_cv.best_params_
```

[27]:
```
{'C': 2}
```

[30]:
```
svm_tuned = SVC(kernel = "linear",C=2).fit(x_train,y_train)
```

[31]:
```
svm_tuned
```

[31]:
```
SVC
SVC(C=2, kernel='linear')
```

[32]:
```
y_pred = svm_tuned.predict(x_test)
```

[33]:
```
cm = confusion_matrix(y_test, y_pred)
```

[34]:
```
cm
```

[34]:
```
array([[96, 12],
       [27, 33]], dtype=int64)
```

[36]:
```
print("our Accuracy is : ",(cm[0][0]+cm[1][1])/(cm[0][0]+cm[1][1]+cm[0][1]+cm[1][0]))
```

```
our Accuracy is :  0.7678571428571429
```

[37]:
```
accuracy_score(y_test,y_pred)
```

[37]:
```
0.7678571428571429
```

[39]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.89      0.83       108
           1       0.73      0.55      0.63        60

    accuracy                           0.77       168
   macro avg       0.76      0.72      0.73       168
weighted avg       0.76      0.77      0.76       168
```

Practical 6: Adaboost Ensemble Learning

```
[1]: # Load libraries
from sklearn.ensemble import AdaBoostClassifier
from sklearn import datasets
# Import train_test_split function
from sklearn.model_selection import train_test_split
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
```

```
[2]: # Load data
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

```
[3]: # 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) # 70% training and 30% test
```

```
[4]: # Create adaboost classifer object
abc = AdaBoostClassifier(n_estimators=50,
                         learning_rate=1)
# Train Adaboost Classifer
model = abc.fit(X_train, y_train)

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

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

```
Accuracy: 0.9111111111111111
```

```
[6]: # Load libraries
from sklearn.ensemble import AdaBoostClassifier

# Import Support Vector Classifier
from sklearn.svm import SVC
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
svc=SVC(probability=True, kernel='linear')

# Create adaboost classifer object
abc =AdaBoostClassifier(n_estimators=50, estimator=svc,learning_rate=1)

# Train Adaboost Classifer
model = abc.fit(X_train, y_train)

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


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

```
Accuracy: 0.9555555555555556
```

```
[7]: import pandas
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 30
#kfold makes trees with split number.
#kfold = model_selection.KFold(n_splits=10, random_state=seed)
#n_estimators : This is the number of trees you want to build before predictions.
#Higher number of trees give you better voting optionsand perfomance performance
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
#cross_val_score method is used to calculate the accuracy of model sliced into x, y
#cross validator cv is optional cv=kfold
results = model_selection.cross_val_score(model, X, Y)
print(results.mean())
```

```
0.7617774382480265
```

Practical 7: Naïve Bayes Classifiers

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB , CategoricalNB , GaussianNB
from sklearn.metrics import accuracy_score
import seaborn as sns
```

[2]:
```
df = pd.read_csv("D:\TYCS Dinesh\AI Practical\Diesases.csv")
```

[3]:
```
df.head(11)
```

[3]:

	Sore Throat	Fever	Swollen Glands	Congestion	Headache	Diagnosis
0	Yes	Yes	Yes	Yes	Yes	Strep thoat
1	No	No	No	Yes	Yes	Allergy
2	Yes	Yes	No	Yes	No	Cold
3	Yes	No	Yes	No	No	Strep thoat
4	No	Yes	No	Yes	No	Cold
5	No	No	No	Yes	No	Allergy
6	No	No	Yes	No	No	Strep thoat
7	Yes	No	No	Yes	Yes	Allergy
8	No	Yes	No	Yes	Yes	Cold
9	Yes	Yes	No	Yes	Yes	Cold

[4]:
```
df.tail()
```

[4]:

	Sore Throat	Fever	Swollen Glands	Congestion	Headache	Diagnosis
5	No	No	No	Yes	No	Allergy
6	No	No	Yes	No	No	Strep thoat
7	Yes	No	No	Yes	Yes	Allergy
8	No	Yes	No	Yes	Yes	Cold
9	Yes	Yes	No	Yes	Yes	Cold

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Sore Throat     10 non-null     object
 1   Fever           10 non-null     object
 2   Swollen Glands  10 non-null     object
 3   Congestion      10 non-null     object
 4   Headache        10 non-null     object
 5   Diagnosis       10 non-null     object
dtypes: object(6)
memory usage: 608.0+ bytes
```

[6]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Sore Throat'] = le.fit_transform(df['Sore Throat'])
df['Fever'] = le.fit_transform(df['Fever'])
df['Swollen Glands'] = le.fit_transform(df['Swollen Glands'])
df['Congestion'] = le.fit_transform(df['Congestion'])
df['Headache'] = le.fit_transform(df['Headache'])
df['Diagnosis'] = le.fit_transform(df['Diagnosis'])
```

[7]:
```
df.info()
```

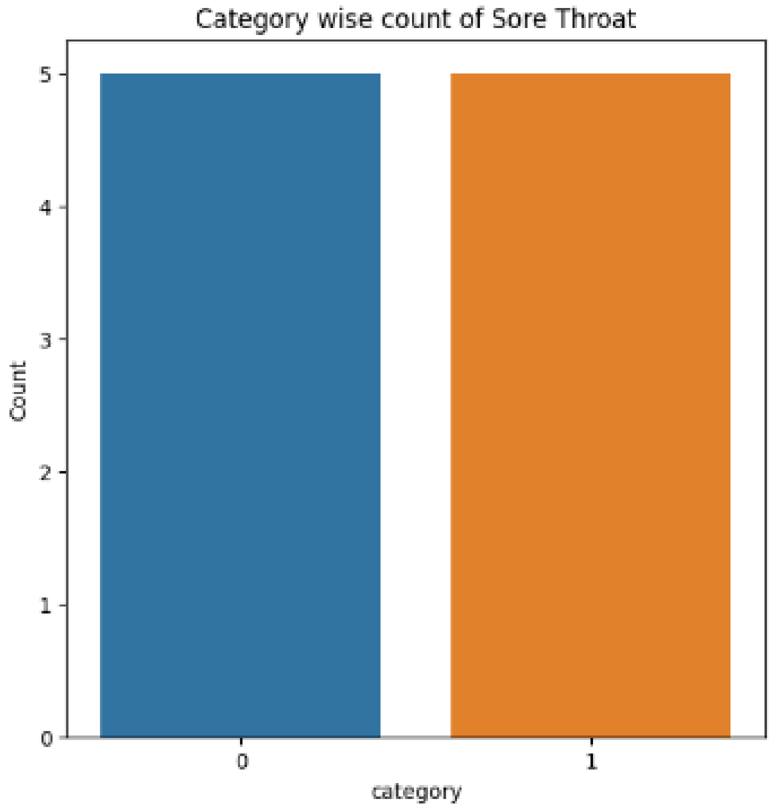
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sore Throat     10 non-null     int32
 1   Fever           10 non-null     int32
 2   Swollen Glands  10 non-null     int32
 3   Congestion      10 non-null     int32
 4   Headache        10 non-null     int32
 5   Diagnosis       10 non-null     int32
dtypes: int32(6)
memory usage: 368.0 bytes
```

[8]:
```
df.head(11)
```

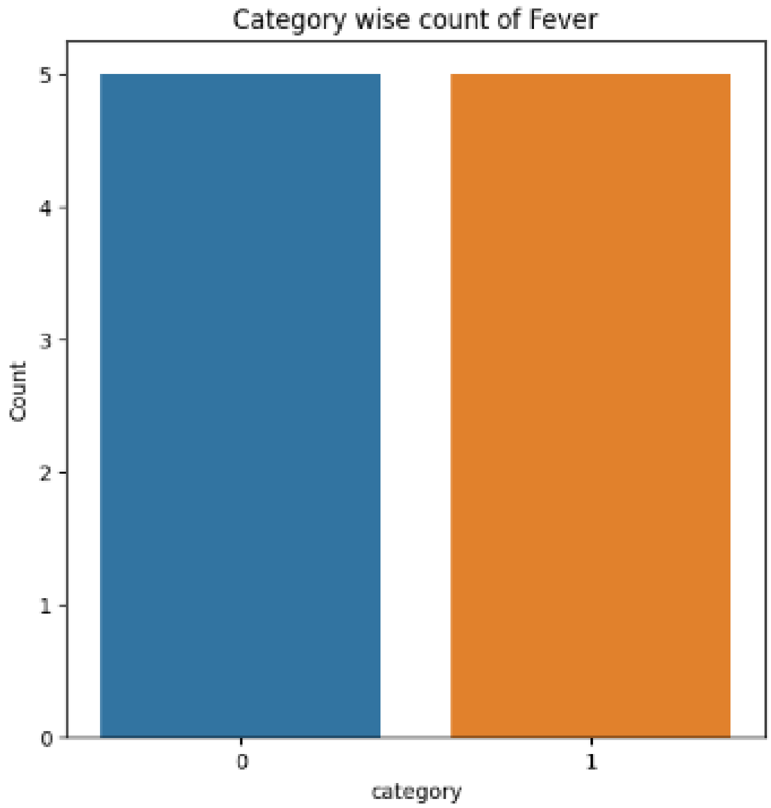
[8]:

	Sore Throat	Fever	Swollen Glands	Congestion	Headache	Diagnosis
0	1	1	1	1	1	2
1	0	0	0	1	1	0
2	1	1	0	1	0	1
3	1	0	1	0	0	2
4	0	1	0	1	0	1
5	0	0	0	1	0	0
6	0	0	1	0	0	2
7	1	0	0	1	1	0
8	0	1	0	1	1	1
9	1	1	0	1	1	1

[9]:
```
fig , ax = plt.subplots(figsize=(6,6))
sns.countplot(x=df['Sore Throat'],data=df)
plt.title("Category wise count of Sore Throat")
plt.xlabel("category")
plt.ylabel("Count")
plt.show()
```

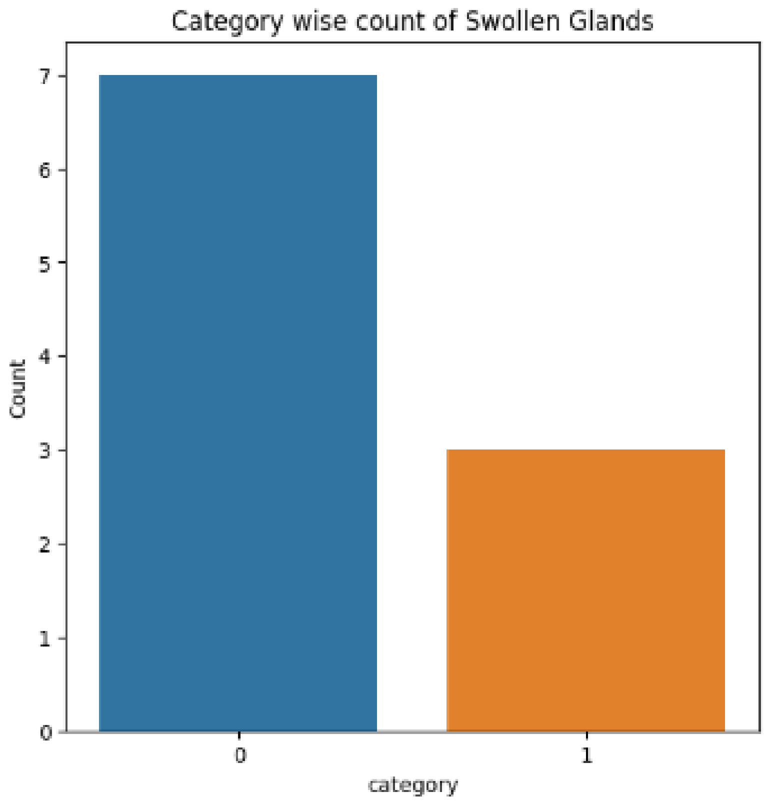


[10]:
```
fig , ax = plt.subplots(figsize=(6,6))
sns.countplot(x=df['Fever'],data=df)
plt.title("Category wise count of Fever")
plt.xlabel("category")
plt.ylabel("Count")
plt.show()
```



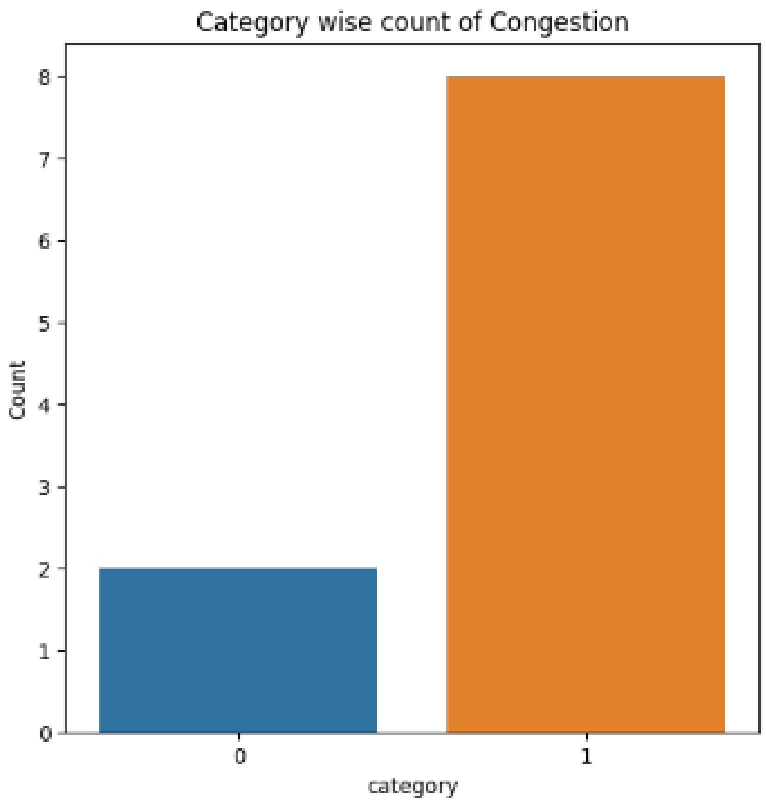
[11]:

```
fig , ax = plt.subplots(figsize=(6,6))
sns.countplot(x=df['Swollen Glands'],data=df)
plt.title("Category wise count of Swollen Glands")
plt.xlabel("category")
plt.ylabel("Count")
plt.show()
```

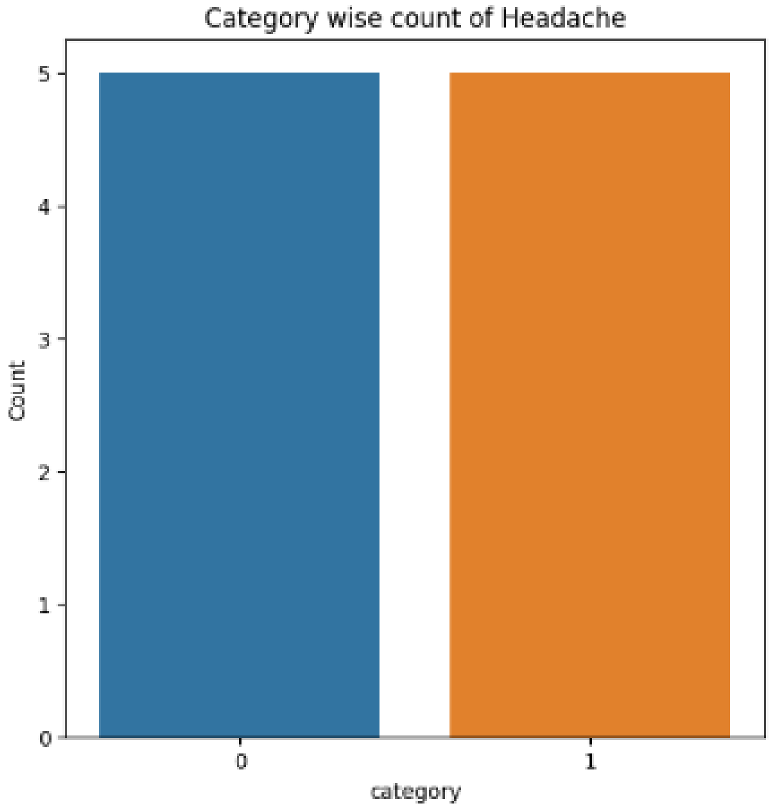


[12]:

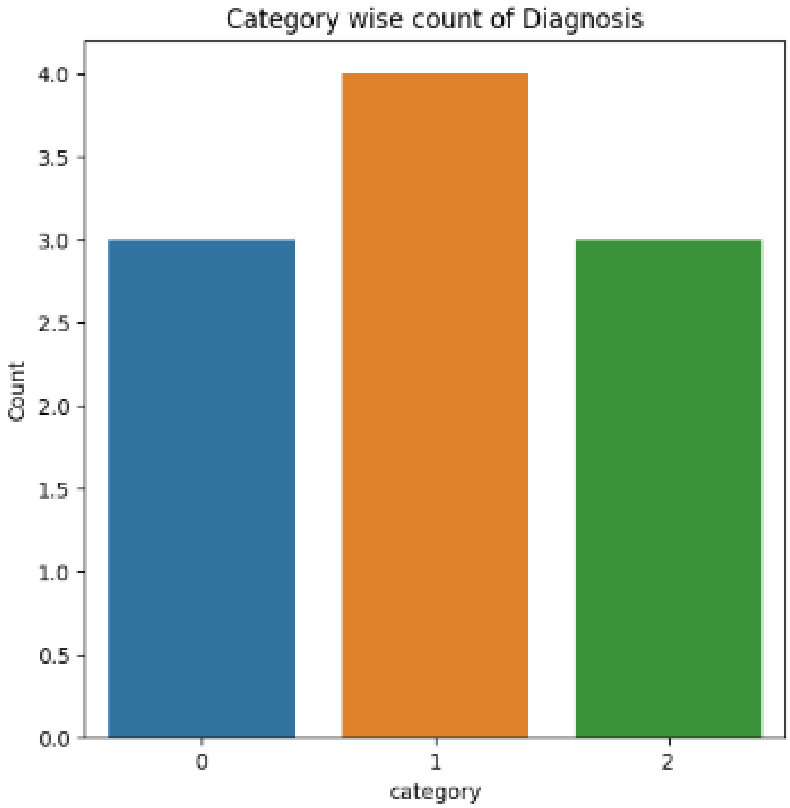
```
fig , ax = plt.subplots(figsize=(6,6))
sns.countplot(x=df['Congestion'],data=df)
plt.title("Category wise count of Congestion")
plt.xlabel("category")
plt.ylabel("Count")
plt.show()
```



[13]:
```
fig , ax = plt.subplots(figsize=(6,6))
sns.countplot(x=df['Headache'],data=df)
plt.title("Category wise count of Headache")
plt.xlabel("category")
plt.ylabel("Count")
plt.show()
```



[14]:
```
fig , ax = plt.subplots(figsize=(6,6))
sns.countplot(x=df['Diagnosis'],data=df)
plt.title("Category wise count of Diagnosis")
plt.xlabel("category")
plt.ylabel("Count")
plt.show()
```



[15]:
```
X = df.drop('Diagnosis',axis=1)
y=df['Diagnosis']
```

[16]:
```
classifier = MultinomialNB()
classifier.fit(X,y)
```

[16]:
```
▾ MultinomialNB
MultinomialNB()
```

[17]:
```
classifier = CategoricalNB()
classifier.fit(X,y)
```

[17]:
```
▾ CategoricalNB
CategoricalNB()
```

[18]:
```
classifier = GaussianNB()
classifier.fit(X,y)
```

[18]:
```
▾ GaussianNB
GaussianNB()
```

[19]:
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, precision_score, recall_score, f1_score
```

[20]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

[21]:
```
classifier = MultinomialNB()
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
print("confusion matrix\n",confusion_matrix(y_test,y_pred))
print("Accuracy : ", accuracy_score(y_test, y_pred))
print("Presicion : ",precision_score(y_test, y_pred, zero_division='warn'))
print("Recall : ", recall_score(y_test,y_pred))
print("F1 score : ",f1_score(y_test,y_pred))
print("Classification report :\n", classification_report(y_test,y_pred))
```

```
confusion matrix
 [[1 0]
 [0 1]]
Accuracy :  1.0
Presicion :  1.0
Recall :  1.0
F1 score :  1.0
Classification report :
               precision    recall  f1-score   support

           1       1.00      1.00      1.00         1
           2       1.00      1.00      1.00         1

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2
```

Practical 8: K-Nearest Neighbors

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

[2]:
```
df = pd.read_csv('D:\AI_Datasets\diabetes.csv')
```

[3]:
```
df.head()
```

[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

[4]:
```
df.shape
```

[4]:
```
(768, 9)
```

[5]:
```
df.dtypes
```

[5]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

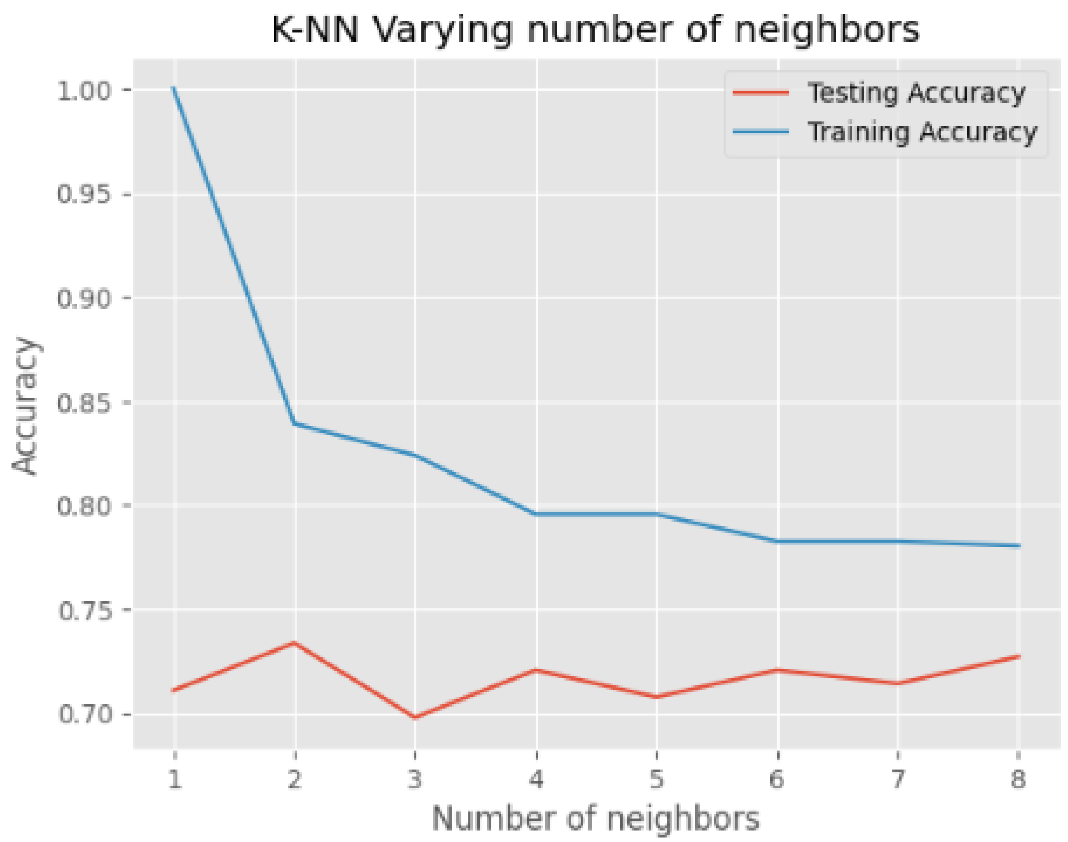
[6]:
```
x = df.drop('Outcome',axis=1).values
y = df['Outcome'].values
```

[7]:
```
from sklearn.model_selection import train_test_split
```

[8]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.4,random_state=42)
```

[9]:
```
from sklearn.neighbors import KNeighborsClassifier
neighbors = np.arange(1,9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train, y_train)
    train_accuracy[i] = knn.score(x_train, y_train)
    test_accuracy[i] = knn.score(x_test, y_test)
```

[10]:
```
plt.title("K-NN Varying number of neighbors")
plt.plot(neighbors, test_accuracy, label="Testing Accuracy")
plt.plot(neighbors, train_accuracy, label="Training Accuracy")
plt.legend()
plt.xlabel("Number of neighbors")
plt.ylabel("Accuracy")
plt.show()
```



[11]:
```
knn = KNeighborsClassifier(n_neighbors=7)
```

[12]:
```
knn.fit(x_train, y_train)
```

[12]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

[13]:
```
knn.score(x_test, y_test)
```

[13]:
```
0.7142857142857143
```

[14]:
```
from sklearn.metrics import confusion_matrix
```

[15]:
```
y_pred = knn.predict(x_test)
```

[16]:
```
confusion_matrix(y_test, y_pred)
```

[16]:
```
array([[163,  43],
       [ 45,  57]], dtype=int64)
```

[17]:
```python
from sklearn.metrics import classification_report
```

[18]:
```python
print(classification_report(y_test, y_pred))
```

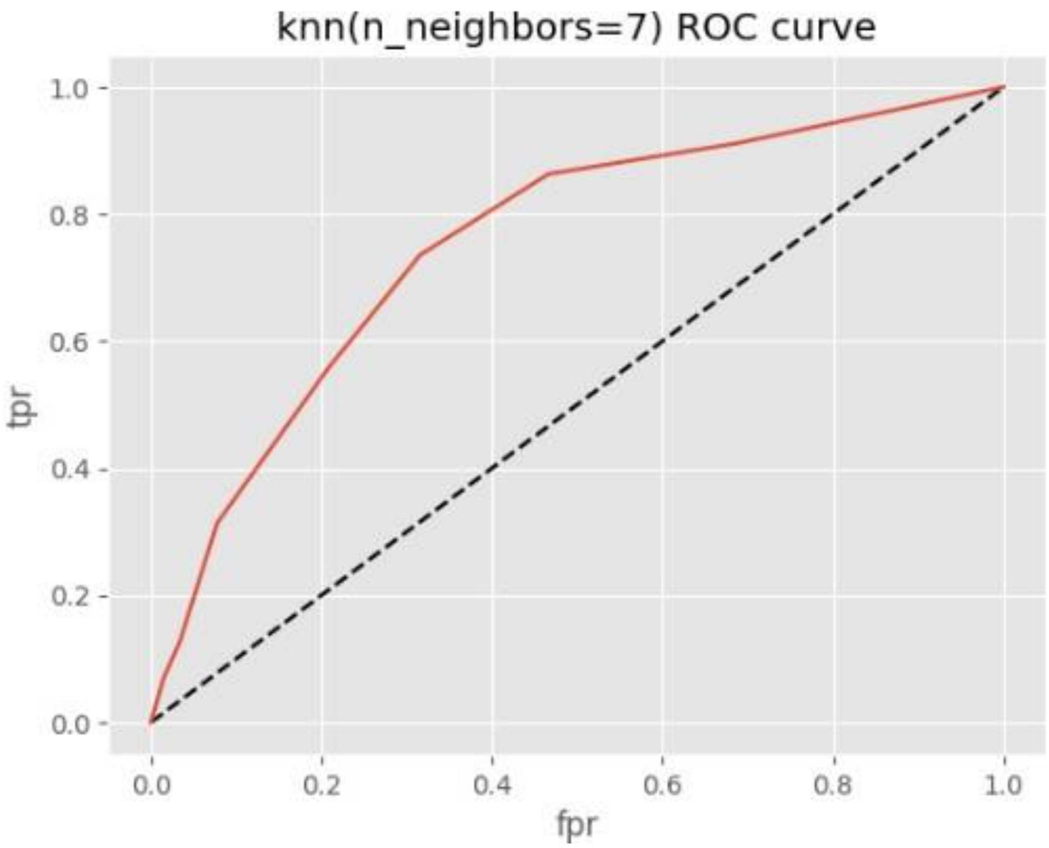
```
              precision    recall  f1-score   support

           0       0.78      0.79      0.79       206
           1       0.57      0.56      0.56       102

    accuracy                           0.71       308
   macro avg       0.68      0.68      0.68       308
weighted avg       0.71      0.71      0.71       308
```

[19]:
```python
y_pred_proba = knn.predict_proba(x_test)[:,1]
```

[20]:
```python
from sklearn.metrics import roc_curve
```

[21]:
```python
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
```

[22]:
```python
plt.plot([0,1],[0,1],"k--")
plt.plot(fpr,tpr,label="knn")
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("knn(n_neighbors=7) ROC curve")
plt.show()
```



[23]:
```python
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,y_pred_proba)
```

[23]:
```
0.7536645726251665
```

[24]:
```python
from sklearn.model_selection import GridSearchCV
```

[25]:
```python
param_grid = {'n_neighbors' : np.arange(1,50)}
```

[26]:
```python
knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn,param_grid,cv=5)
knn_cv.fit(x,y)
```

[26]:
```
▸ GridSearchCV
▸ estimator: KNeighborsClassifier
    ▸ KNeighborsClassifier
```

[27]:
```python
knn_cv.best_score_
```

[27]:
```
0.7578558696205755
```

[28]:
```python
knn_cv.best_params_
```

[28]:
```
{'n_neighbors': 14}
```