*Seminar Report*

*on*

# Classification of Astronomical data


Submitted by

**Ritusree Mahapatra**

**Shruti Nautiyal**

Master of Science (Tech.) in Engineering Physics

*Under the supervision of*

**Dr. Ravi Kumar Jatoth**




**National Institute of Technology Warangal**

**Warangal – 506004, Telangana, India**

# ACKNOWLEDGEMENT

The successful completion of the this task would be put incomplete without the mention of the people who made it possible, whose constant guidance and encouragement had pushed us to the edge of excellence.

We wish to express our deep sense of gratitude to Dr.Ravi Kumar Jatoth, Associate Professor , Department of Electronics and Communication Engineering, National Institute of Technology Warangal, for his able guidance and useful suggestions.

We would also like to thank our family and friends for their support and help.

# Abstract

Our goal in this project is to classify sky objects as star or a galaxy using SDSS (Sloan Digital Sky Survey) data.

This Sloan Digital Sky Survey or SDSS is a major multi-spectral imaging and spectroscopic redshift survey using a dedicated 2.5-m wide-angle optical telescope .SDSS measures magnitudes in five different colors by taking images through five color filters. datasets were constructed from the SDSS Data Release

A filter is a kind of screen that blocks out all light except for light with a specific color. The SDSS telescope's filters are green (g), red (r), and three colors that correspond to light not visible to the human eye: ultraviolet (u), and two infrared wavelengths ( i and z). On SkyServer, the five magnitudes (through the five filters) of a star are symbolized by u, g, r, i, and z. The astronomers who planned the SDSS chose these filters to view a wide range of colors, while focusing on the colors of interesting celestial objects.

We introduce the Quantum Machine Learning algorithms, describe how the models were optimised using a spectroscopically selected training dataset, and give an in-depth classification of sky objects with a better accuracy.

# Introduction

Quantum machine learning is a research area that explores the interplay of ideas from quantum computing and machine learning.

For example, we might want to find out whether quantum computers can speed up the time it takes to train or evaluate a machine learning model. On the other hand, we can leverage techniques from machine learning to help us uncover quantum error-correcting codes, estimate the properties of quantum systems, or develop new quantum algorithms.

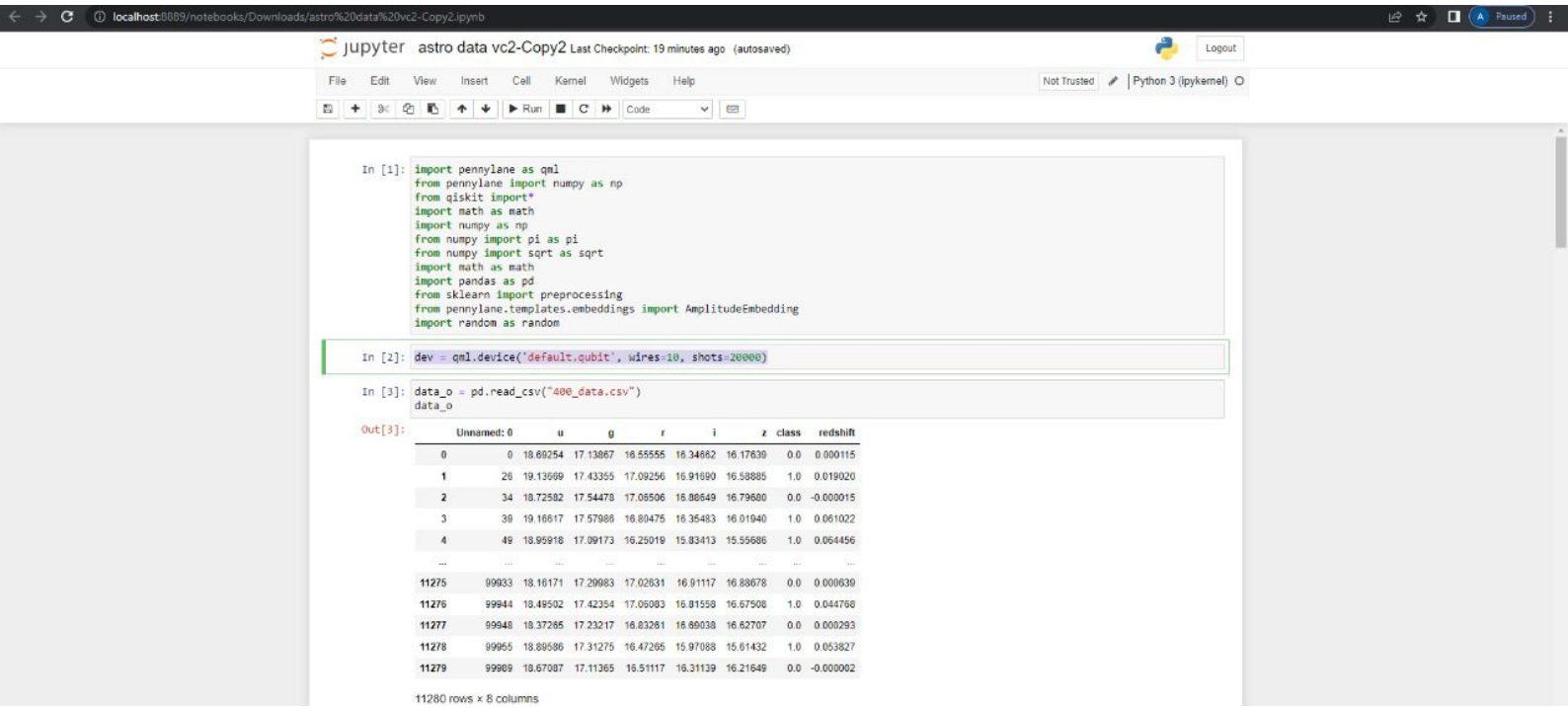
In this project we had attained accuracy by using Gradient descent and Random walk hybrid optimization techniques.

Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks. Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates. Until the function is close to or equal to zero, the model will continue to adjust its parameters to yield the smallest possible error.

In machine learning, a "random walk" approach can be applied in various ways to help the technology sift through the large training data sets that provide the basis for the machine's eventual comprehension.

A random walk, mathematically, is something that can be described in several different technical ways. Some describe it as a randomized collection of variables; others might call it a "stochastic process." Regardless, the random walk contemplates a scenario where a variable set takes a path that is a pattern based on random increments, according to an integer set: For example, a walk on a number line where the variable moves plus or minus one at every step.

The codes of this project are shown below:



localhost:8889/notebooks/Downloads/astro%20data%20vc2-Copy2.ipynb

jupyter astro data vc2-Copy2 Last Checkpoint: 19 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

In [1]:
```
import pennylane as qml
from pennylane import numpy as np
from qiskit import*
import math as math
import numpy as np
from numpy import pi as pi
from numpy import sqrt as sqrt
import math as math
import pandas as pd
from sklearn import preprocessing
from pennylane.templates.embeddings import AmplitudeEmbedding
import random as random
```

In [2]:
```
dev = qml.device('default.qubit', wires=10, shots=20000)
```

In [3]:
```
data_o = pd.read_csv("400_data.csv")
data_o
```

Out[3]:

| | Unnamed: 0 | u | g | r | i | z | class | redshift |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 18.69254 | 17.13867 | 16.55555 | 16.34662 | 16.17639 | 0.0 | 0.000115 |
| 1 | 26 | 19.13669 | 17.43355 | 17.09256 | 16.91690 | 16.58885 | 1.0 | 0.019020 |
| 2 | 34 | 18.72582 | 17.54478 | 17.06506 | 16.88649 | 16.79680 | 0.0 | -0.000015 |
| 3 | 39 | 19.16617 | 17.57986 | 16.80475 | 16.35483 | 16.01940 | 1.0 | 0.061022 |
| 4 | 49 | 18.95918 | 17.09173 | 16.25019 | 15.83413 | 15.55686 | 1.0 | 0.064456 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 11275 | 99933 | 18.16171 | 17.29983 | 17.02631 | 16.91117 | 16.88678 | 0.0 | 0.000639 |
| 11276 | 99944 | 18.49502 | 17.42354 | 17.06083 | 16.81558 | 16.67508 | 1.0 | 0.044768 |
| 11277 | 99948 | 18.37265 | 17.23217 | 16.83261 | 16.69038 | 16.62707 | 0.0 | 0.000293 |
| 11278 | 99955 | 18.89586 | 17.31275 | 16.47265 | 15.97088 | 15.61432 | 1.0 | 0.053827 |
| 11279 | 99989 | 18.67087 | 17.11365 | 16.51117 | 16.31139 | 16.21649 | 0.0 | -0.000002 |

11280 rows × 8 columns

jupyter astro data vc2-Copy2 Last Checkpoint: 22 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

11280 rows × 6 columns

In [4]:
```
data = data_o[[ 'u', 'g', 'r', 'i', 'z', 'redshift']]
data
```

Out[4]:

| | u | g | r | i | z | redshift |
|---|---|---|---|---|---|---|
| 0 | 18.69254 | 17.13867 | 16.55555 | 16.34662 | 16.17639 | 0.000115 |
| 1 | 19.13689 | 17.43355 | 17.09256 | 16.91690 | 16.58885 | 0.019020 |
| 2 | 18.72582 | 17.54478 | 17.06506 | 16.88649 | 16.79680 | -0.000015 |
| 3 | 19.16617 | 17.57986 | 16.80475 | 16.35483 | 16.01940 | 0.061022 |
| 4 | 18.95918 | 17.09173 | 16.25019 | 15.83413 | 15.55686 | 0.064456 |
| ... | ... | ... | ... | ... | ... | ... |
| 11275 | 18.16171 | 17.29983 | 17.02631 | 16.91117 | 16.88678 | 0.000639 |
| 11276 | 18.49502 | 17.42354 | 17.06083 | 16.81558 | 16.67508 | 0.044768 |
| 11277 | 18.37265 | 17.23217 | 16.83261 | 16.69038 | 16.62707 | 0.000293 |
| 11278 | 18.89586 | 17.31275 | 16.47265 | 15.97088 | 15.61432 | 0.053827 |
| 11279 | 18.67087 | 17.11365 | 16.51117 | 16.31139 | 16.21649 | -0.000002 |

11280 rows × 6 columns

In [5]:
```
data = preprocessing.normalize(data)
data
```

Out[5]:
```
array([[ 4.91548439e-01,  4.50687091e-01,  4.35353074e-01,
         4.29858946e-01,  4.25382492e-01,  3.01498877e-06],
       [ 4.90254408e-01,  4.46622417e-01,  4.37886745e-01,
         4.33386589e-01,  4.24982421e-01,  4.87261648e-04],
       [ 4.80785333e-01,  4.50462137e-01,  4.38145329e-01,
         4.33560545e-01,  4.31257755e-01, -3.87071602e-07],
       ...,
       [ 4.78729338e-01,  4.49012273e-01,  4.38601086e-01,
         4.34895051e-01,  4.33245406e-01,  7.64533826e-06],
       [ 5.00210505e-01,  4.58302531e-01,  4.36063432e-01,
         4.22780594e-01,  4.13341750e-01,  1.42491605e-03],
       [ 4.91486232e-01,  4.50494453e-01,  4.34634055e-01,
         4.29376007e-01,  4.26877889e-01, -5.69752623e-08]])
```

localhost:8889/notebooks/Downloads/astro%20data%20vc2-Copy2.ipynb

jupyter astro data vc2-Copy2 Last Checkpoint: 23 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

In [6]:
```
data= pd.DataFrame(data)
data
```

Out[6]:

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0.491548 | 0.450687 | 0.435353 | 0.429859 | 0.425382 | 3.014989e-06 |
| 1 | 0.490254 | 0.446622 | 0.437887 | 0.433387 | 0.424982 | 4.872616e-04 |
| 2 | 0.480785 | 0.450462 | 0.438145 | 0.433561 | 0.431258 | -3.870718e-07 |
| 3 | 0.497714 | 0.456520 | 0.436392 | 0.424708 | 0.415997 | 1.584632e-03 |
| 4 | 0.505195 | 0.455434 | 0.433010 | 0.421923 | 0.414535 | 1.717527e-03 |
| ... | ... | ... | ... | ... | ... | ... |
| 11275 | 0.470476 | 0.448149 | 0.441064 | 0.438081 | 0.437449 | 1.656554e-05 |
| 11276 | 0.477931 | 0.450243 | 0.440870 | 0.434533 | 0.430902 | 1.156843e-03 |
| 11277 | 0.478729 | 0.449012 | 0.438601 | 0.434895 | 0.433245 | 7.645338e-06 |
| 11278 | 0.500211 | 0.458303 | 0.436063 | 0.422781 | 0.413342 | 1.424916e-03 |
| 11279 | 0.491486 | 0.450494 | 0.434635 | 0.429376 | 0.426878 | -5.697526e-08 |

11280 rows × 6 columns

```
@qml.qnode(dev)
def circuit(data, oom):

    for i in range(len(data)):
        AmplitudeEmbedding(features=data, wires=range(N), pad_with=N, normalize=True)
        for i in range(N):
            qml.RY(oom[i], wires=i)
        for i in range(N-1):
            if (i ==0):
                qml.CRX(oom[N],wires=(N, 0))
            qml.CRX(oom[N+i],wires=(i, i+1))


        for i in range(N):
            qml.RY(oom[2*N +i], wires=i)
        for i in range(N-1):
            if (i ==0):
                qml.CRX(oom[3*N ],wires=(N, 0))
            qml.CRX(oom[3*N+i],wires=(i, i+1))
```

```
    return qml.probs(wires=[0, 1, 2])
```

In [7]:
```
@qml.qnode(dev)
def circuit(data, omm):

    for i in range(len(data)):
        AmplitudeEmbedding(features=data, wires=range(N), pad_with=N, normalize=True)
        for i in range(N):
            qml.RX(omm[i], wires=i)
        for i in range(N):
            qml.RZ(omm[N+i], wires=i)
        for i in range(N-1):
            if (i ==0):
                qml.CRZ(omm[N],wires=(N, 0))
            qml.CRX(omm[2*N+i],wires=(i, i+1))




    return qml.probs(wires=[0, 1, 2])
```

In [8]:
```
N=3
data1 = data.iloc[:1,]
data1 = np.array(data1[0])
data1_c=data_o.iloc[:1,]['class']
print(np.array(data1)[0])
```

```
0.4915484389740754
```

In [9]:
```
def parity(k):
    c=0
    k='{0:03b}'.format(k)

    for i in k:
        if(i=='1'):
            c=c+1
        else:
            pass
    if(c%2!=0 ):
        p=1
    else:
        p=0
    return p
def costs(omm):
    cost = 0
    reg = circuit(data1, omm)
```

```
    if(c%2!=0 ):
        p=1
    else:
        p=0
    return p
def costs(omm):
    cost = 0
    reg = circuit(data1, omm)
    for i in range(len(reg)):
        if(parity(i)==int(data1_c)):
            cost = cost + reg[i]
    cost= 1- cost
    return  cost
```

In [10]:
```
inf = np.array([0.001, 0.001, 0.7, 0.0, 0.001, 0.001, 0.01, 0.001, 0.001, 0.001, 0.001, 0])
omm = inf
cost = costs(inf)
cost
```

Out[10]:
```
tensor(0.5496, requires_grad=True)
```

In [11]:
```
def rnd(stepsize, omm):
    for i in range(len(omm)):
        x = round(random.uniform(-stepsize,stepsize),3)
        omm[i] = omm[i]+x
    return omm
```

In [12]:
```
def rndopt(cost, omm):

    cost1 = costs(omm)
    cost = costs(omm)
    while(cost1<cost):
        omm1 = rnd(stepsize, omm)
        cost1 = costs(omm1)
    omm1 = omm
    return omm
```

In [13]:
```
stepsize=0.05
steps = 350
```

```
opt = qml.MomentumOptimizer(stepsize=0.01, momentum=0.9)
steps = 250
omm = inf
```

localhost:8889/notebooks/Downloads/astro%20data%20vc2-Copy2.ipynb

jupyter astro data vc2-Copy2 Last Checkpoint: 25 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

```
    omm1 = omm
    return omm
```

In [13]:
```
stepsize=0.05
steps = 350
```

```
opt = qml.MomentumOptimizer(stepsize=0.01, momentum=0.9)
steps = 250
omm = inf
```

In [14]:
```
for i in range(steps):
    omm = rndopt(cost, omm)
    if (i +1)%10 ==0:
        print(costs(omm),omm)
        pass
    pass
print(costs(omm),omm)
```

```
0.56525 [-0.023 -0.102  0.673  0.25   0.054 -0.029  0.384 -0.228  0.115 -0.007
  0.035  0.154]
0.5781499999999999 [-0.19  -0.077  0.575  0.246  0.056 -0.168  0.515 -0.049  0.166  0.049
  0.112  0.132]
0.57735 [-0.181 -0.306  0.412  0.281 -0.032 -0.119  0.521 -0.066 -0.082  0.037
  0.034 -0.015]
0.59185 [-3.34000000e-01 -2.02000000e-01  5.03000000e-01  4.71000000e-01
 -5.89805982e-17 -9.10000000e-02  5.34000000e-01  9.40000000e-02
 -2.50000000e-02 -5.20000000e-02  9.00000000e-02  6.80000000e-02]
0.5942000000000001 [-0.392 -0.297  0.677  0.383 -0.065 -0.149  0.385  0.112 -0.094  0.075
  0.212  0.171]
0.5861000000000001 [-0.428 -0.119  0.607  0.323 -0.119 -0.375  0.423  0.281 -0.017 -0.02
  0.316  0.191]
0.6123 [ 3.988  2.83   1.362 -1.014  4.531 -4.27   0.587  2.566 -3.336 -0.729
  2.375 -0.997]
0.6338 [ 4.367  4.345  2.716  0.431  6.164 -5.227  1.858 -1.25  -0.927  1.987
  6.094  0.203]
0.6349 [ 4.343  4.49   2.696  0.391  6.306 -5.319  1.943 -1.173 -0.99   1.942
  6.008  0.343]
```

In [ ]:

In [ ]:

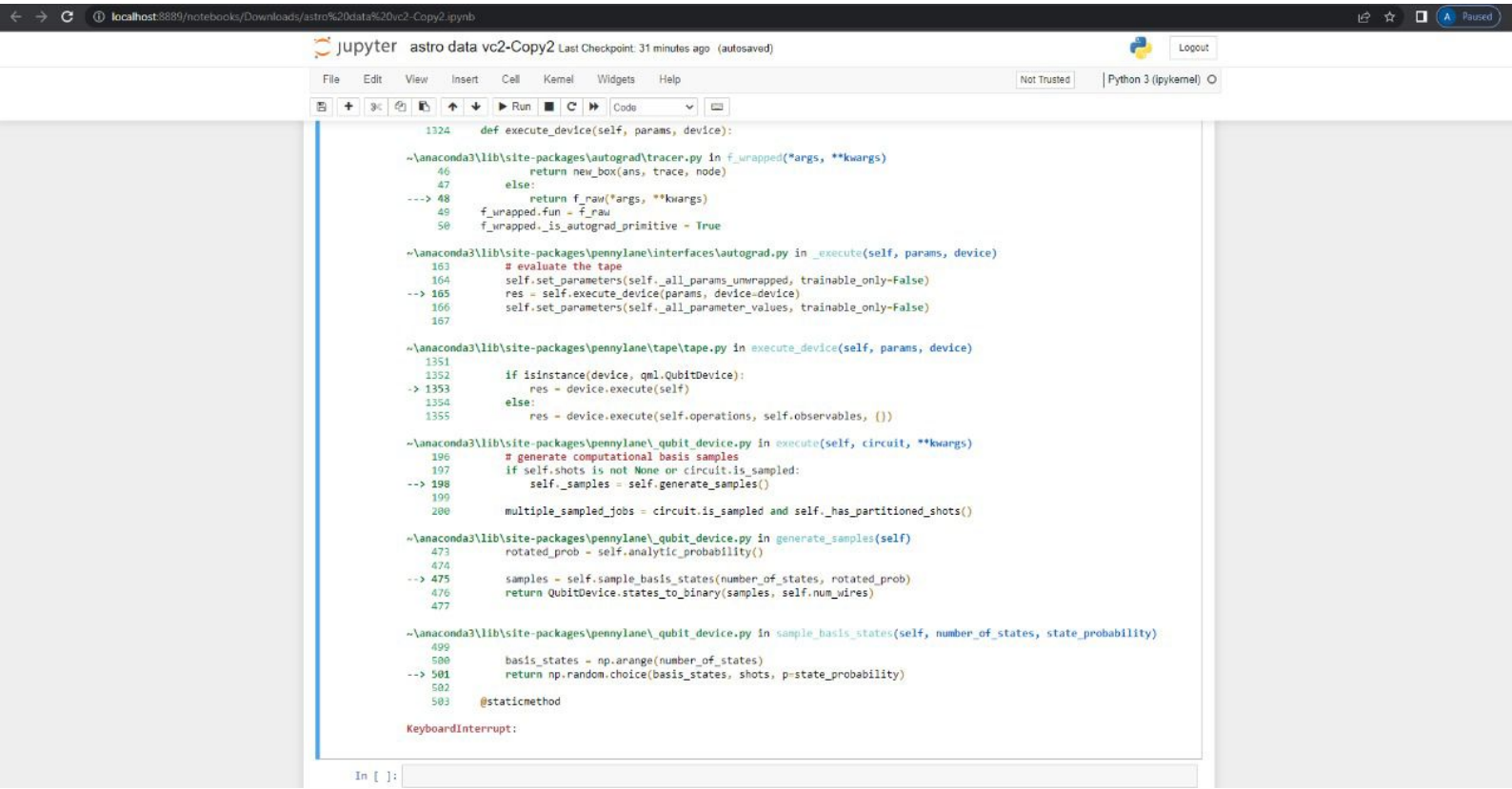
In [ ]:

In [14]:
```
for i in range(steps):
    omm = rndopt(cost, omm)
    if (i +1)%10 ==0:
        print(costs(omm),omm)
        pass
    pass
print(costs(omm),omm)
```

```
0.56525 [-0.023 -0.102  0.673  0.25   0.054 -0.029  0.384 -0.228  0.115 -0.007
  0.035  0.154]
0.5781499999999999 [-0.19  -0.077  0.575  0.246  0.056 -0.168  0.515 -0.049  0.166  0.049
  0.112  0.132]
0.57735 [-0.181 -0.306  0.412  0.281 -0.032 -0.119  0.521 -0.066 -0.082  0.037
  0.034 -0.015]
0.59185 [-3.34000000e-01 -2.02000000e-01  5.03000000e-01  4.71000000e-01
 -5.89805982e-17 -9.10000000e-02  5.34000000e-01  9.40000000e-02
 -2.50000000e-02 -5.20000000e-02  9.00000000e-02  6.80000000e-02]
0.5942000000000001 [-0.392 -0.297  0.677  0.383 -0.065 -0.149  0.385  0.112 -0.094  0.075
  0.212  0.171]
0.5861000000000001 [-0.428 -0.119  0.607  0.323 -0.119 -0.375  0.423  0.281 -0.017 -0.02
  0.316  0.191]
0.6123 [ 3.988  2.83   1.362 -1.014  4.531 -4.27   0.587  2.566 -3.336 -0.729
  2.375 -0.997]
0.6338 [ 4.367  4.345  2.716  0.431  6.164 -5.227  1.858 -1.25  -0.927  1.987
  6.094  0.203]
0.6349 [ 4.343  4.49   2.696  0.391  6.306 -5.319  1.943 -1.173 -0.99   1.942
  6.008  0.343]
0.6377 [ 4.539  4.496  2.531  0.376  6.547 -5.291  2.076 -1.308 -1.337  2.119
  5.849  0.368]
0.64055 [ 4.721  4.469  2.613  0.251  6.373 -5.285  2.139 -1.236 -1.315  2.277
  5.906  0.464]
0.64605 [ 5.123  4.963  1.451 -0.15   4.627 -7.657  3.444 -1.565 -1.6    3.333
  5.151 -0.051]
0.6574 [ 5.026  4.918  1.483 -0.154  4.674 -7.725  3.518 -1.532 -1.525  3.321
  5.077 -0.089]
0.66815 [ 5.009  4.792  1.494 -0.062  4.739 -7.709  3.685 -1.509 -1.472  3.496
  5.088  0.019]
0.68495 [ 5.056  4.811  1.587 -0.068  4.605 -7.548  3.856 -1.488 -1.344  3.589
  4.958  0.228]
0.7032 [ 4.936  5.238  1.299  0.208  4.662 -7.47   4.114 -1.715 -1.221  3.626
  5.127 -0.021]
0.7355 [ 5.081  5.429  1.313  0.397  4.461 -7.658  4.253 -1.465 -1.099  3.732
  5.145 -0.096]
0.73695 [ 5.154  5.544  1.448  0.359  4.398 -7.762  4.251 -1.382 -1.36   3.438
  5.441 -0.197]
0.7575 [ 0.048  2.645  5.266 -6.364  4.643  4.212 -4.228 -4.374  0.717 -4.051
  4.737 -9.55 ]
0.7681 [-0.123  2.358  5.167 -6.367  4.852  4.17  -4.291 -4.513  0.731 -3.966
  4.878 -9.937]
0.78025 [ 0.017  2.25   5.18  -6.512  4.626  4.202 -4.442 -4.614  0.498 -3.654
```

```
  5.145 -0.096]
0.73695 [ 5.154  5.544  1.448  0.359  4.398 -7.762  4.251 -1.382 -1.36   3.438
  5.441 -0.197]
0.7575 [ 0.048  2.645  5.266 -6.364  4.643  4.212 -4.228 -4.374  0.717 -4.051
  4.737 -9.55 ]
0.7681 [-0.123  2.358  5.167 -6.367  4.852  4.17  -4.291 -4.513  0.731 -3.966
  4.878 -9.937]
0.78025 [ 0.017  2.25   5.18  -6.512  4.626  4.202 -4.442 -4.614  0.498 -3.654
  4.931 -9.956]
0.78095 [ 7.0000e-03  2.1770e+00  4.9280e+00 -6.4490e+00  4.6590e+00  4.3260e+00
 -4.5460e+00 -4.5770e+00  5.1900e-01 -3.5510e+00  5.0670e+00 -1.0078e+01]

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-14-b7b5d167af65> in <module>
      1 for i in range(steps):
----> 2     omm = rndopt(cost, omm)
      3     if (i +1)%10 ==0:
      4         print(costs(omm),omm)
      5         pass

<ipython-input-12-094c844c5524> in rndopt(cost, omm)
      5     while(cost1<cost):
      6         omm1 = rnd(stepsize, omm)
----> 7         cost1 = costs(omm1)
      8     omm1 = omm
      9     return omm

<ipython-input-9-7d37d06f5d66> in costs(omm)
     15 def costs(omm):
     16     cost = 0
---> 17     reg = circuit(data1, omm)
     18     for i in range(len(reg)):
     19         if(parity(i)==int(data1_c)):

~\anaconda3\lib\site-packages\pennylane\qnode.py in __call__(self, *args, **kwargs)
    685             res = fn(results)
    686         else:
--> 687             res = self.qtape.execute(device=self.device)
    688
    689         finite_diff = any(

~\anaconda3\lib\site-packages\pennylane\tape\tape.py in execute(self, device, params)
   1320             params = self.get_parameters()
   1321
-> 1322         return self._execute(params, device=device)
   1323
   1324     def execute_device(self, params, device):
```

```
~\anaconda3\lib\site-packages\pennylane\qnode.py in __call__(self, *args, **kwargs)
    685             res = fn(results)
    686         else:
--> 687             res = self.qtape.execute(device=self.device)
    688
    689         finite_diff = any(

~\anaconda3\lib\site-packages\pennylane\tape\tape.py in execute(self, device, params)
   1320             params = self.get_parameters()
   1321
-> 1322         return self._execute(params, device=device)
   1323
   1324     def execute_device(self, params, device):

~\anaconda3\lib\site-packages\autograd\tracer.py in f_wrapped(*args, **kwargs)
     46             return new_box(ans, trace, node)
     47         else:
---> 48             return f_raw(*args, **kwargs)
     49     f_wrapped.fun = f_raw
     50     f_wrapped._is_autograd_primitive = True

~\anaconda3\lib\site-packages\pennylane\interfaces\autograd.py in _execute(self, params, device)
    163         # evaluate the tape
    164         self.set_parameters(self._all_params_unwrapped, trainable_only=False)
--> 165         res = self.execute_device(params, device=device)
    166         self.set_parameters(self._all_parameter_values, trainable_only=False)
    167

~\anaconda3\lib\site-packages\pennylane\tape\tape.py in execute_device(self, params, device)
   1351
   1352         if isinstance(device, qml.QubitDevice):
-> 1353             res = device.execute(self)
   1354         else:
   1355             res = device.execute(self.operations, self.observables, {})

~\anaconda3\lib\site-packages\pennylane\_qubit_device.py in execute(self, circuit, **kwargs)
    196         # generate computational basis samples
    197         if self.shots is not None or circuit.is_sampled:
--> 198             self._samples = self.generate_samples()
    199
    200         multiple_sampled_jobs = circuit.is_sampled and self._has_partitioned_shots()

~\anaconda3\lib\site-packages\pennylane\_qubit_device.py in generate_samples(self)
    473         rotated_prob = self.analytic_probability()
    474
--> 475         samples = self.sample_basis_states(number_of_states, rotated_prob)
    476         return QubitDevice.states_to_binary(samples, self.num_wires)
    477

~\anaconda3\lib\site-packages\pennylane\_qubit_device.py in sample_basis_states(self, number_of_states, state_probability)
```


localhost:8889/notebooks/Downloads/astro%20data%20vc2-Copy2.ipynb
jupyter astro data vc2-Copy2 Last Checkpoint: 31 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
1324 def execute_device(self, params, device):
~\anaconda3\lib\site-packages\autograd\tracer.py in f_wrapped(*args, **kwargs)
46 return new_box(ans, trace, node)
47 else:
---> 48 return f_raw(*args, **kwargs)
49 f_wrapped.fun = f_raw
50 f_wrapped._is_autograd_primitive = True
~\anaconda3\lib\site-packages\pennylane\interfaces\autograd.py in _execute(self, params, device)
163 # evaluate the tape
164 self.set_parameters(self._all_params_unwrapped, trainable_only=False)
--> 165 res = self.execute_device(params, device=device)
166 self.set_parameters(self._all_parameter_values, trainable_only=False)
167
~\anaconda3\lib\site-packages\pennylane\tape\tape.py in execute_device(self, params, device)
1351
1352 if isinstance(device, qml.QubitDevice):
-> 1353 res = device.execute(self)
1354 else:
1355 res = device.execute(self.operations, self.observables, {})
~\anaconda3\lib\site-packages\pennylane_qubit_device.py in execute(self, circuit, **kwargs)
196 # generate computational basis samples
197 if self.shots is not None or circuit.is_sampled:
--> 198 self._samples = self.generate_samples()
199
200 multiple_sampled_jobs = circuit.is_sampled and self._has_partitioned_shots()
~\anaconda3\lib\site-packages\pennylane_qubit_device.py in generate_samples(self)
473 rotated_prob = self.analytic_probability()
474
--> 475 samples = self.sample_basis_states(number_of_states, rotated_prob)
476 return QubitDevice.states_to_binary(samples, self.num_wires)
477
~\anaconda3\lib\site-packages\pennylane_qubit_device.py in sample_basis_states(self, number_of_states, state_probability)
499
500 basis_states = np.arange(number_of_states)
--> 501 return np.random.choice(basis_states, shots, p=state_probability)
502
503 @staticmethod
KeyboardInterrupt:
In [ ]:

## CONCLUSION

In this project, we had tried to attain greater accuracy in classification of astronomical data as stars and galaxies.

We did it by implementing gradient descent and random walk hybrid optimization techniques.