**Programming Part for Report Crew**

**// Link to our github repository :** [**https://github.com/bhanvimenghani/Data-Driven-Astronomy**](https://github.com/bhanvimenghani/Data-Driven-Astronomy)

**//**

**Code for Cross matching the**

import numpy as np

import statistics

import time

from astropy.coordinates import SkyCoord

from astropy import units as u

import pandas as pd

def crossmatch(cat1, cat2, max\_dist):

matches = []

nomatches = []

start = time.perf\_counter()

skycat1 = SkyCoord(cat1\*u.degree, frame='icrs')

skycat2 = SkyCoord(cat2\*u.degree, frame='icrs')

closest\_ids, closest\_dists, closest\_dists3d = skycat1.match\_to\_catalog\_sky(skycat2)

closest\_dists\_deg = closest\_dists.value

for cat1idx in range(len(cat1)):

if closest\_dists\_deg[cat1idx] > max\_dist:

nomatches.append(cat1idx)

else:

matches.append((cat1idx,closest\_ids[cat1idx],closest\_dists\_deg[cat1idx]))

#closest\_dist.value returns an array of degrees

#print("vals", closest\_ids)

#print("dists", closest\_dists)

#print("dists.val", closest\_dists.value)

return (matches, nomatches, time.perf\_counter() - start)

#to create a text file

def conv(match):

match\_1 = matches

f = open('newfiles.txt', 'w')

for t in match\_1:

line = ' '.join(str(x) for x in t)

f.write(line + '\n')

f.close()

# You can use this to test your function.

# Any code inside this `if` statement will be ignored by the automarker.

if \_\_name\_\_ == '\_\_main\_\_':

# The example in the question

cat1 = np.genfromtxt('gmrt.csv', delimiter=',',skip\_header=55,usecols=[5,6],max\_rows=5434)

cat2 = np.genfromtxt('opticaldata.csv', delimiter=',',skip\_header=1,usecols=[1,2],max\_rows=252198)

matches, no\_matches, time\_taken = crossmatch(cat1, cat2, 5)

text = conv(matches)

print('matches:', matches)

print('unmatched:', no\_matches)

print('time taken:', time\_taken)

**Output : We find the optical dataset that could be used to find RED shift**

**Link to the Optical Dataset formed after cross matching :** [**https://www.kaggle.com/bhanvimenghani/optical-csv**](https://www.kaggle.com/bhanvimenghani/optical-csv)

import numpy as np

import pandas as pd

from matplotlib import pyplot as plt

from sklearn.tree import DecisionTreeRegressor

def get\_features\_targets(data):

features = np.zeros((data.shape[0], 4))

features[:, 0] = data['u'] - data['g']

features[:, 1] = data['g'] - data['r']

features[:, 2] = data['r'] - data['i']

features[:, 3] = data['i'] - data['z']

targets = data['redshift']

return features, targets

def median\_diff(predicted, actual):

return np.median(np.abs(predicted - actual))

def validate\_model(model, features, targets):

split = 2\*features.shape[0]//3

train\_features, test\_features = features[:split], features[split:]

train\_targets, test\_targets = targets[:split], targets[split:]

model.fit(train\_features, train\_targets)

predictions = model.predict(test\_features)

return median\_diff(test\_targets, predictions)

if \_\_name\_\_ == "\_\_main\_\_":

data = pd.read\_csv('../input/optical-csv/optical\_data.csv')

features, targets = get\_features\_targets(data)

dtr = DecisionTreeRegressor()

diff = validate\_model(dtr, features, targets)

print('Median difference: {:f}'.format(diff))

if \_\_name\_\_ == "\_\_main\_\_":

data = pd.read\_csv('../input/optical-csv/optical\_data.csv')

cmap = plt.get\_cmap('YlOrRd')

u\_g = data['u'] - data['g']

r\_i = data['r'] - data['i']

redshift = data['redshift']

plot = plt.scatter(u\_g, r\_i, s=0.5, lw=0, c=redshift, cmap=cmap)

cb = plt.colorbar(plot)

cb.set\_label('Redshift')

plt.xlabel('Colour index u-g')

plt.ylabel('Colour index r-i')

plt.title('Redshift (colour) u-g versus r-i')

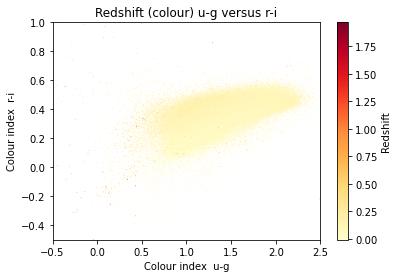
plt.xlim(-0.5, 2.5)

plt.ylim(-0.5, 1)

plt.show()

**Output :**

**Median difference: 0.017306**



import numpy as np

import pandas as pd

from matplotlib import pyplot as plt

from sklearn.tree import DecisionTreeRegressor

# paste your get\_features\_targets function here

def get\_features\_targets(data):

features = np.zeros((data.shape[0], 4))

features[:, 0] = data['u'] - data['g']

features[:, 1] = data['g'] - data['r']

features[:, 2] = data['r'] - data['i']

features[:, 3] = data['i'] - data['z']

targets = data['redshift']

return features, targets

# paste your median\_diff function here

def median\_diff(predicted, actual):

return np.median(np.abs(predicted - actual))

# Complete the following function

def accuracy\_by\_treedepth(features, targets, depths):

# split the data into testing and training sets

split = features.shape[0]//2

train\_features, test\_features = features[:split], features[split:]

train\_targets, test\_targets = targets[:split], targets[split:]

# Initialise arrays or lists to store the accuracies for the below loop

train\_diffs = []

test\_diffs = []

# Loop through depths

for depth in depths:

# initialize model with the maximum depth.

dtr = DecisionTreeRegressor(max\_depth=depth)

# train the model using the training set

dtr.fit(train\_features, train\_targets)

# Get the predictions for the training set and calculate their med\_diff

predictions = dtr.predict(train\_features)

train\_diffs.append(median\_diff(train\_targets, predictions))

# Get the predictions for the testing set and calculate their med\_diff

predictions = dtr.predict(test\_features)

test\_diffs.append(median\_diff(test\_targets, predictions))

# Return the accuracies for the training and testing sets

return train\_diffs, test\_diffs

if \_\_name\_\_ == "\_\_main\_\_":

data = pd.read\_csv('../input/finald/Skyserver\_SQL11\_13\_2020 9\_51\_34 AM\_opticaldata.csv')

features, targets = get\_features\_targets(data)

# Generate several depths to test

tree\_depths = [i for i in range(1, 36, 2)]

# Call the function

train\_med\_diffs, test\_med\_diffs = accuracy\_by\_treedepth(features, targets, tree\_depths)

print("Depth with lowest median difference : {}".format(tree\_depths[test\_med\_diffs.index(min(test\_med\_diffs))]))

# Plot the results

train\_plot = plt.plot(tree\_depths, train\_med\_diffs, label='Training set')

test\_plot = plt.plot(tree\_depths, test\_med\_diffs, label='Validation set')

plt.xlabel("Maximum Tree Depth")

plt.ylabel("Median of Differences")

plt.legend()

plt.show()

**Output:**