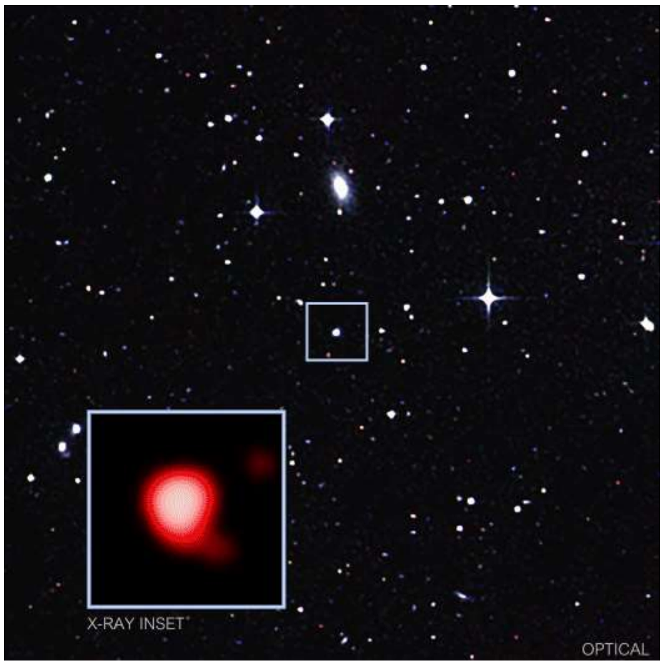
Predicting Stars, Galaxies & Quasars with ML Model

Project ID: PSGQ0922

Project Name: Predicting Stars, Galaxies & Quasars with ML Model



Workflow of this notebook

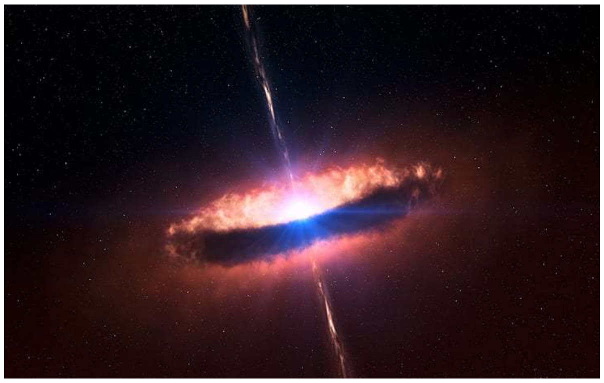
About the Problem & Our Dataset -

So what exactly are stars, galaxies, and quasars?

- A GALAXY is a gravitationally bound system of stars, stellar remnants, interstellar gas, dust, and dark matter. Galaxies are categorised according to their visual morphology as elliptical, spiral, or irregular. Many galaxies are thought to have supermassive black holes at their active centers.

- A STAR is a type of astronomical object consisting of a luminous spheroid of plasma held together by its own gravity. The nearest star to Earth is the Sun.

- A QUASAR, also known as a quasi-stellar object, is an extremely luminous active galactic nucleus (AGN). The power radiated by quasars is enormous. A typical quasar is 27 trillion times brighter than our sun! If you were to place a quasar at the distance of Pluto, it would vaporise all of Earth's oceans to steam in a fifth of a second.



The dataset we'll be using for this model is the [Sloan Digital Sky Survey DR14 (https://www.kaggle.com/datasets/lucidlenn/sloan-digital-sky-survey)](https://www.kaggle.com/datasets/lucidlenn/sloan-digital-sky-survey)

The data consists of 10,000 observations of space taken by the SDSS. Every observation is described by 17 feature columns and 1 class column which identifies it to be either a star, galaxy or quasar. 30% of it is used in testing the model performance and 70% in training of the

Note: Upload the notebook to a kaggle session at the dataset link and run the cells

Importing necessary libraries and modules for the dataset

First of all we need to import all the packages we need. Numpy and Pandas for data manipulation and all the modules from sklearn for the machine learning feature

In [ ]:
```
#Importing Necessary Libraries

'''Operating System Functionality'''
import os

'''Data Handling & Linear Algebra'''
import numpy as np
import pandas as pd

'''Visualisation'''
import matplotlib.pyplot as plt
import seaborn as sns

'''Manipulating Data and Model Building'''
import tensorflow as tf
from tensorflow import keras

'''Data Analysis'''
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

'''Measuring Classification Performance'''
from sklearn.metrics import accuracy_score

'''Classification & Regression'''
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

'''Ignore warnings'''
import warnings
warnings.filterwarnings("ignore")

%matplotlib inline
```

Documents of the above used libraries and modules in case you aren't familiar and want to know more about it:-

- os (https://docs.python.org/3/library/os.html)

- numpy (https://numpy.org/doc/1.23/user/absolute_beginners.html)
- pandas (https://pandas.pydata.org/docs/user_guide/index.html#user-guide)
- matplotlib.pyplot (https://matplotlib.org/stable/tutorials/introductory/pyplot.html)
- seaborn (https://seaborn.pydata.org/tutorial/introduction.html)
- tensorflow (https://www.tensorflow.org/guide)
- keras (https://keras.io/guides/)
- All sklearn libraries (https://scikit-learn.org/stable/user_guide.html)

In [ ]:
```python
#Importing the dataset
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

Exploratory Analysis

In [ ]:
```python
#Loading data into a dataframe
data = pd.read_csv("/content/Skyserver_SQL2_27_2018 6_51_39 PM.csv")
```

In [ ]:
```python
#Displaying the first 5 rows of the dataset
data.head()
```

Out[7]:

	objid	ra	dec	u	g	r	i	z	run	re
0	1.237650e+18	183.531326	0.089693	19.47406	17.04240	15.94699	15.50342	15.22531	752	
1	1.237650e+18	183.598370	0.135285	18.66280	17.21449	16.67637	16.48922	16.39150	752	
2	1.237650e+18	183.680207	0.126185	19.38298	18.19169	17.47428	17.08732	16.80125	752	
3	1.237650e+18	183.870529	0.049911	17.76536	16.60272	16.16116	15.98233	15.90438	752	
4	1.237650e+18	183.883288	0.102557	17.55025	16.26342	16.43869	16.55492	16.61326	752	

In [ ]:
```python
#Displaying the dimensions of the dataset
data.shape
```

Out[8]:
```
(10000, 18)
```

The object id columns are of little use in the analysis hence we can delete them from the dataset.

In [ ]:
```python
#Drop the object id columns, they are of no use in the analysis
data.drop(['objid','specobjid'], axis=1, inplace=True)
```

```
In [ ]: #Data after dropping columns
        data.head(10)
```

Out[10]:

	ra	dec	u	g	r	i	z	run	rerun	camcol
0	183.531326	0.089693	19.47406	17.04240	15.94699	15.50342	15.22531	752	301	4
1	183.598370	0.135285	18.66280	17.21449	16.67637	16.48922	16.39150	752	301	4
2	183.680207	0.126185	19.38298	18.19169	17.47428	17.08732	16.80125	752	301	4
3	183.870529	0.049911	17.76536	16.60272	16.16116	15.98233	15.90438	752	301	4
4	183.883288	0.102557	17.55025	16.26342	16.43869	16.55492	16.61326	752	301	4
5	183.847174	0.173694	19.43133	18.46779	18.16451	18.01475	18.04155	752	301	4
6	183.864379	0.019201	19.38322	17.88995	17.10537	16.66393	16.36955	752	301	4
7	183.900081	0.187473	18.97993	17.84496	17.38022	17.20673	17.07071	752	301	4
8	183.924588	0.097246	17.90616	16.97172	16.67541	16.53776	16.47596	752	301	4
9	183.973498	0.081626	18.67249	17.71375	17.49362	17.28284	17.22644	752	301	4

```
In [ ]: data.shape
Out[11]: (10000, 16)
```

```
In [ ]: data.describe()
```

Out[12]:

	ra	dec	u	g	r	i	
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	100
mean	175.529987	14.836148	18.619355	17.371931	16.840963	16.583579	
std	47.783439	25.212207	0.828656	0.945457	1.067764	1.141805	
min	8.235100	-5.382632	12.988970	12.799550	12.431600	11.947210	
25%	157.370946	-0.539035	18.178035	16.815100	16.173333	15.853705	
50%	180.394514	0.404166	18.853095	17.495135	16.858770	16.554985	
75%	201.547279	35.649397	19.259232	18.010145	17.512675	17.258550	
max	260.884382	68.542265	19.599900	19.918970	24.802040	28.179630	

In [ ]:
```
#Checking for null values to determine completeness of the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   ra        10000 non-null  float64
 1   dec       10000 non-null  float64
 2   u         10000 non-null  float64
 3   g         10000 non-null  float64
 4   r         10000 non-null  float64
 5   i         10000 non-null  float64
 6   z         10000 non-null  float64
 7   run       10000 non-null  int64  
 8   rerun     10000 non-null  int64  
 9   camcol    10000 non-null  int64  
 10  field     10000 non-null  int64  
 11  class     10000 non-null  object 
 12  redshift  10000 non-null  float64
 13  plate     10000 non-null  int64  
 14  mjd       10000 non-null  int64  
 15  fiberid   10000 non-null  int64  
dtypes: float64(8), int64(7), object(1)
memory usage: 1.2+ MB
```

No missing data so the dataset is complete with no voids and missing cells

The Target from data is Data classification to Star Galaxy or Quasar, so the class column has 3 Categories and in this case we need to convert them into numeric data.

Sklearn provides a very efficient tool for encoding the levels of categorical features into numeric values. `LabelEncoder().fit` encodes labels with a value between 0 and n_classes-1 where n is the number of distinct labels. If a label repeats it assigns the same value to as assigned earlier. You can read more about it here (https://stackoverflow.com/questions/66056695/what-does-labelencoder-fit-do)!

In [ ]:
```
le = LabelEncoder().fit(data['class'])
data['class'] = le.transform(data['class'])
```

The Final Dataset

```
In [ ]: #Printing the dataset after all the changes to check the dataset
        data.head(10)
```

Out[15]:

	ra	dec	u	g	r	i	z	run	rerun	camcol
0	183.531326	0.089693	19.47406	17.04240	15.94699	15.50342	15.22531	752	301	4
1	183.598370	0.135285	18.66280	17.21449	16.67637	16.48922	16.39150	752	301	4
2	183.680207	0.126185	19.38298	18.19169	17.47428	17.08732	16.80125	752	301	4
3	183.870529	0.049911	17.76536	16.60272	16.16116	15.98233	15.90438	752	301	4
4	183.883288	0.102557	17.55025	16.26342	16.43869	16.55492	16.61326	752	301	4
5	183.847174	0.173694	19.43133	18.46779	18.16451	18.01475	18.04155	752	301	4
6	183.864379	0.019201	19.38322	17.88995	17.10537	16.66393	16.36955	752	301	4
7	183.900081	0.187473	18.97993	17.84496	17.38022	17.20673	17.07071	752	301	4
8	183.924588	0.097246	17.90616	16.97172	16.67541	16.53776	16.47596	752	301	4
9	183.973498	0.081626	18.67249	17.71375	17.49362	17.28284	17.22644	752	301	4

After lable encoding, Galaxies have been replaced by number 0, Quasars by number 1 and Stars by number 2

```
In [ ]: #Checking the information about the data
        data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   ra        10000 non-null  float64
 1   dec       10000 non-null  float64
 2   u         10000 non-null  float64
 3   g         10000 non-null  float64
 4   r         10000 non-null  float64
 5   i         10000 non-null  float64
 6   z         10000 non-null  float64
 7   run       10000 non-null  int64  
 8   rerun     10000 non-null  int64  
 9   camcol    10000 non-null  int64  
 10  field     10000 non-null  int64  
 11  class     10000 non-null  int64  
 12  redshift  10000 non-null  float64
 13  plate     10000 non-null  int64  
 14  mjd       10000 non-null  int64  
 15  fiberid   10000 non-null  int64  
dtypes: float64(8), int64(8)
memory usage: 1.2 MB
```

We can understand that the dataset is complete with no null values. And since the dataset has only numerical values now, there's no need of further encoding and type conversions.

Perform train and test split

```
In [ ]: X = data.drop('class', axis=1)
        y = data['class']
```

Data Scaling

Scaling means converting floating-point feature values from their natural range (for example, 100 to 900) into a standard range—usually 0 and 1 (or sometimes -1 to +1). `StandardScaler` follows Standard Normal Distribution (SND). Therefore, it makes mean = 0 and scales the data to unit variance. You can read more about other types of scalers [here (https://www.geeksforgeeks.org/standardscaler-minmaxscaler-and-robustscaler-techniques-ml/)](https://www.geeksforgeeks.org/standardscaler-minmaxscaler-and-robustscaler-techniques-ml/) and scaling in general [here (https://developers.google.com/machine-learning/data-prep/transform/normalization)](https://developers.google.com/machine-learning/data-prep/transform/normalization).

```
In [ ]: from sklearn.preprocessing import StandardScaler
        scaler = StandardScaler(copy=True , with_mean= True , with_std = True)
        X= scaler.fit_transform(X)
```

```
In [ ]: #Show data
        X[:20]
```

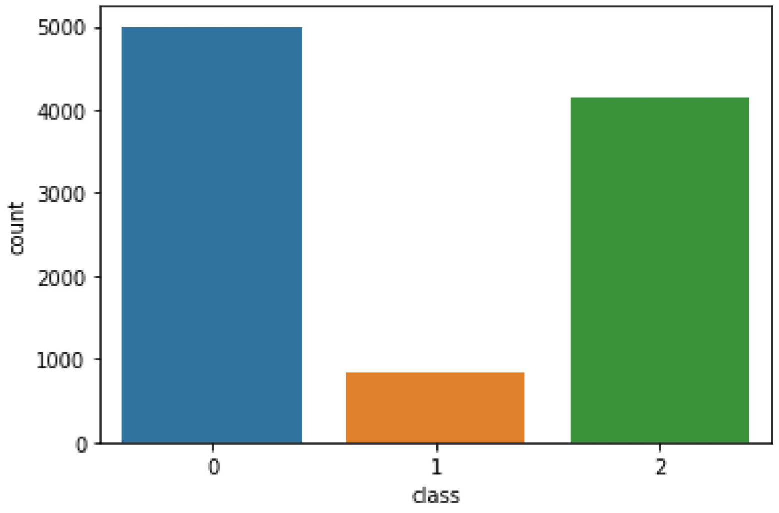
```
Out[19]: array([[ 0.16745842, -0.58492272,  1.03148637, -0.34855938, -0.83728027,
                 -0.94605772, -0.99534154, -0.83806089,  0.        ,  0.21085172,
                 -0.21763043, -0.36973112,  1.03148936,  1.30931064,  0.66863177],
                [ 0.16886159, -0.58311429,  0.05243046, -0.16653251, -0.15415531,
                 -0.08264457, -0.02604308, -0.83806089,  0.        ,  0.21085172,
                 -0.21763043, -0.36984929, -0.63621258, -0.87919741,  0.91101156],
                [ 0.17057433, -0.58347525,  0.92156796,  0.86709322,  0.59315368,
                  0.44120145,  0.31452753, -0.83806089,  0.        ,  0.21085172,
                 -0.21147922, -0.05302706, -0.65633905, -0.60919097,  0.77527888],
                [ 0.17455754, -0.58650069, -1.03063038, -0.81362749, -0.63669227,
                 -0.52660429, -0.43092107, -0.83806089,  0.        ,  0.21085172,
                 -0.20532801, -0.36999261,  1.03148936,  1.30931064,  0.76073609],
                [ 0.17482457, -0.58441247, -1.29023238, -1.17251944, -0.37676237,
                 -0.02510121,  0.15827647, -0.83806089,  0.        ,  0.21085172,
                 -0.20532801, -0.36818949,  1.03148936,  1.30931064,  0.77043128],
                [ 0.17406874, -0.58159078,  0.97991837,  1.15913585,  1.2396114 ,
                  1.25349122,  1.34542376, -0.83806089,  0.        ,  0.21085172,
                 -0.20532801, -0.36889881, -0.63565351, -0.8454466 ,  1.16793412],
                [ 0.17442882, -0.58771882,  0.9218576 ,  0.54793007,  0.24763882,
                  0.07037522, -0.0442872 , -0.83806089,  0.        ,  0.21085172,
                 -0.20532801, -0.11185311, -0.65633905, -0.60919097,  0.99826828],
                [ 0.17517603, -0.58104423,  0.43515363,  0.50034225,  0.50505868,
                  0.54578672,  0.53849374, -0.83806089,  0.        ,  0.21085172,
                 -0.20532801, -0.36889818,  1.03148936,  1.30931064,  0.78497407],
                [ 0.17568894, -0.58462313, -0.8607082 , -0.4233206 , -0.15505443,
                 -0.0401308 ,  0.04415727, -0.83806089,  0.        ,  0.21085172,
                 -0.1991768 , -0.36947888, -0.63621258, -0.87919741,  1.17278172],
                [ 0.17671255, -0.58524271,  0.06412468,  0.36155588,  0.61126719,
                  0.61244768,  0.66793136, -0.83806089,  0.        ,  0.21085172,
                 -0.1991768 , -0.26550833, -0.65577999, -0.62441192,  0.22750057],
                [ 0.17683178, -0.58308602,  0.81867332,  0.45518723,  0.32002733,
                  0.29758829,  0.31289013, -0.83806089,  0.        ,  0.21085172,
                 -0.1991768 , -0.36979707,  1.03148936,  1.30931064,  0.74134571],
                [ 0.17905309, -0.58403354,  0.25791803,  0.15434418,  0.09531295,
                  0.1143867 ,  0.15158558, -0.83806089,  0.        ,  0.21085172,
                 -0.19302559, -0.3681061 ,  1.03148936,  1.30931064,  0.94009713],
                [ 0.179404  , -0.58088408,  1.13821874,  0.86650088,  0.76483872,
                  0.78139043,  0.80556427, -0.83806089,  0.        ,  0.21085172,
                 -0.19302559, -0.36956736,  1.03148936,  1.30931064,  0.92555434],
                [ 0.18062649, -0.58547992,  1.15921765,  0.37678737,  0.13715012,
                  0.08511578,  0.06767929, -0.83806089,  0.        ,  0.21085172,
                 -0.19302559, -0.36968672,  1.03148936,  1.30931064,  0.93524953],
                [ 0.18123476, -0.58453441,  0.76913279,  0.18696495, -0.19217122,
                 -0.38043364, -0.54561378, -0.83806089,  0.        ,  0.21085172,
                 -0.19302559, -0.18427654, -0.65577999, -0.62441192,  0.17417702],
                [ 0.18460582, -0.58026059,  0.14357054,  1.30915544,  1.45732014,
                  1.51360998,  1.2914644 , -0.83806089,  0.        ,  0.21085172,
                 -0.18687438,  0.3297998 , -0.65633905, -0.60919097,  1.13400095],
                [ 0.18190916, -0.58661793,  0.53127794,  0.76494705,  1.01778171,
                  1.1465937 ,  1.2341887 , -0.83806089,  0.        ,  0.21085172,
                 -0.18687438, -0.36916795, -0.63565351, -0.8454466 , -1.54187183],
                [ 0.18240867, -0.58061649,  0.72660397,  2.04202274,  2.15158106,
                  2.21584041,  2.33699748, -0.83806089,  0.        ,  0.21085172,
                 -0.18687438,  2.66073327, -0.65633905, -0.60919097,  1.11461057],
                [ 0.18540578, -0.58577178,  0.51243926,  0.14716211,  0.03033282,
                  0.02413909,  0.05003362, -0.83806089,  0.        ,  0.21085172,
                 -0.18072317, -0.36909046, -0.63621258, -0.87919741,  1.33275238],
        [ 0.18523939, -0.58156585, -0.96878016, -0.531168  , -0.28296358,
         -0.33967158, -0.27939936, -0.83806089,  0.        ,  0.21085172,
         -0.18072317, -0.18251527, -0.65633905, -0.60919097,  1.35214276]])
```

In [ ]:
```
#Performing the 30% test and 70% train split here
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
```

Density Distribution Plots

In [ ]:
```
#Checking the number of labels for each class of the dataset where 0 = Galaxy,
sns.countplot(x=data['class'])
```

Out[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb4fa16d700>
```

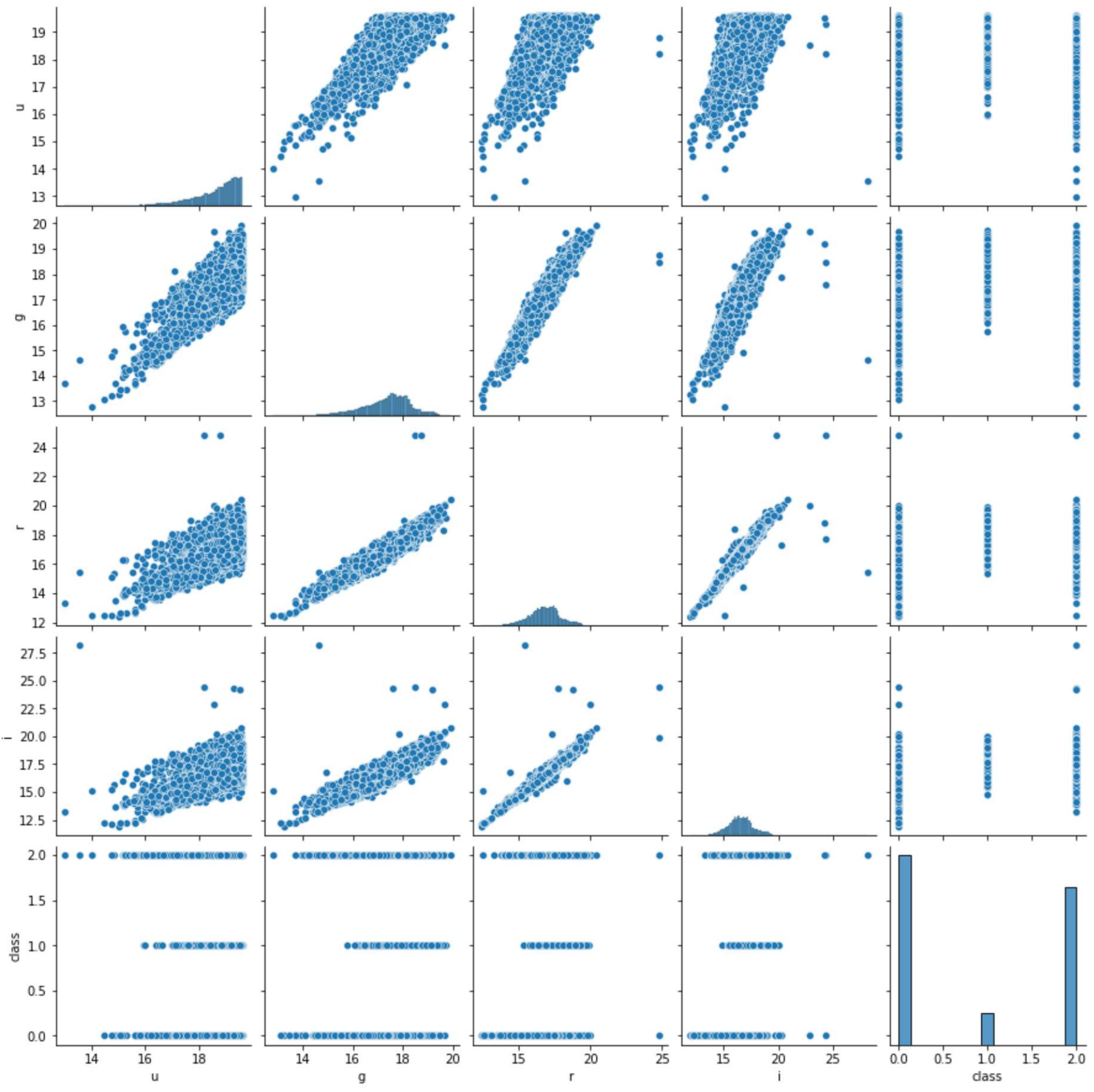


Some information about the filters used while gathering the data:

"U" stands for ultraviolet. "G" stands for green. "R" stands for red. "I" stands for infrared.

```
In [ ]: #Using pairplots to establish and understand interdependancy of train features
        sns.pairplot(data[['u','g','r','i','class']])
```

```
Out[22]: <seaborn.axisgrid.PairGrid at 0x7fb4fa0691f0>
```

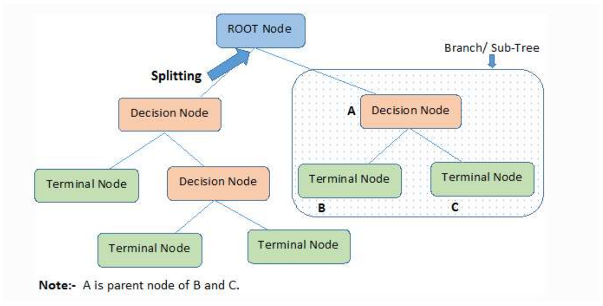


Machine Learning models (Classification models)

Decision tree classifier -

The decision of making strategic splits heavily affects a tree’s accuracy. The decision criteria are different for classification and regression trees.

Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that the purity of the node increases with respect to the target variable. The decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.



The algorithm selection is also based on the type of target variables. Let us look at some algorithms used in Decision Trees:

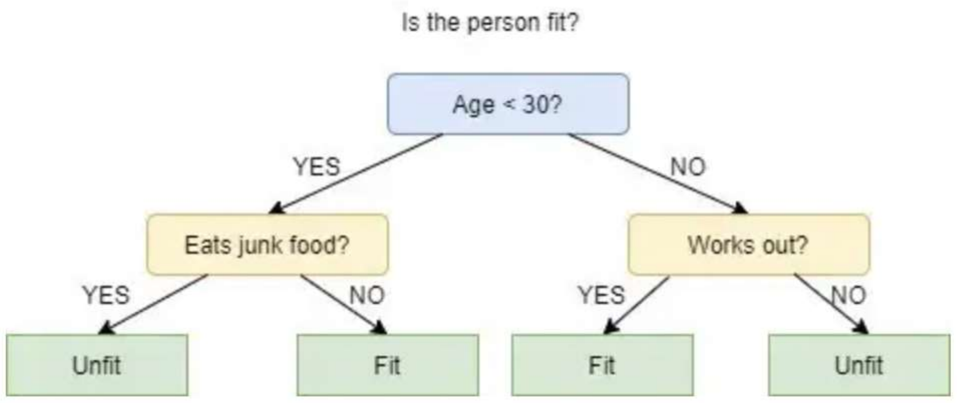
1. ID3 → (extension of D3)
2. C4.5 → (successor of ID3)
3. CART → (Classification And Regression Tree)
4. CHAID → (Chi-square automatic interaction detection Performs multi-level splits when computing classification trees)
5. MARS → (multivariate adaptive regression splines)

To understand these, let's have a look at the most primitive of these - ID3.

[Link for the example. (https://towardsdatascience.com/decision-trees-for-classification-id3-algorithm-explained-89df76e72df1)](https://towardsdatascience.com/decision-trees-for-classification-id3-algorithm-explained-89df76e72df1)

You can read further about the methods [here (https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html)](https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html).

Let's consider this example -



The initial node is called the **root** node (colored in blue), the final nodes are called the **leaf** nodes (colored in green) and the rest of the nodes are called **intermediate** or **internal** nodes. The root and intermediate nodes represent the decisions while the leaf nodes represent the outcomes.

ID3 stands for Iterative Dichotomiser 3 and is named such because the algorithm iteratively (repeatedly) dichotomizes(divides) features into two or more groups at each step. Invented by Ross Quinlan, ID3 uses a top-down greedy approach to build a decision tree. In simple words, the top-down approach means that we start building the tree from the top and the greedy approach means that at each iteration we select the best feature at the present moment to create a node. **Most generally ID3 is only used for classification problems with nominal features only.**

ID3 uses Information Gain or just Gain to find the best feature. Information Gain calculates the reduction in the entropy and measures how well a given feature separates or classifies the target classes. The feature with the highest Information Gain is selected as the best one.

In simple words, Entropy is the measure of disorder and the Entropy of a dataset is the measure of disorder in the target feature of the dataset. In the case of binary classification (where the target column has only two types of classes) entropy is 0 if all values in the target column are homogenous(similar) and will be 1 if the target column has equal number values for both the classes.

ID3 Steps -

1. Calculate the Information Gain of each feature.
2. Considering that all rows don’t belong to the same class, split the dataset S into subsets using the feature for which the Information Gain is maximum.
3. Make a decision tree node using the feature with the maximum Information gain.
4. If all rows belong to the same class, make the current node as a leaf node with the class as its label.
5. Repeat for the remaining features until we run out of all features, or the decision tree has all leaf nodes.

We denote our dataset as S, entropy is calculated as -

$$\text{Entropy}(S) = - \sum p_i * \log_2(p_i) \; ; \; i = 1 \text{ to } n$$

Where n is the total number of classes in the target column; p_i is the probability of class ‘i’ or the ratio of “number of rows with class i in the target column” to the “total number of rows” in the dataset.

Information Gain for a feature column A is calculated as -

$$IG(S, A) = \text{Entropy}(S) - \sum((|S_v| \; / \; |S|) * \text{Entropy}(S_v))$$

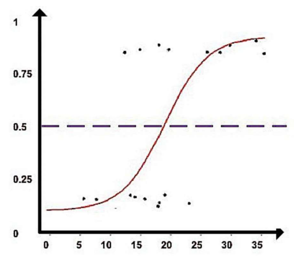
where S_v is the set of rows in S for which the feature column A has value v, $|S_v|$ is the number of

In [ ]:
```
# Decision Tree Classifier
dtClassifer = DecisionTreeClassifier(max_leaf_nodes=15,max_depth=3)
```

Linear regression classifier -

Machine learning generally involves predicting a quantitative outcome or a qualitative class. The former is commonly referred to as a regression problem. In the scenario of linear regression, the input is a continuous variable, and the prediction is a numerical value. When predicting a qualitative outcome (class), the task is considered a classification problem. Examples of classification problems include predicting what products a user will buy or if a target user will click on an online advertisement.

Not all algorithms fit cleanly into this simple dichotomy, though, and logistic regression is a notable example. Logistic regression is part of the regression family as it involves predicting outcomes based on quantitative relationships between variables. However, unlike linear regression, it accepts both continuous and discrete variables as input and its output is qualitative. In addition, it predicts a discrete class such as “Yes/No” or “Customer/Non-customer”.



In practice, the logistic regression algorithm analyzes relationships between variables. It assigns probabilities to discrete outcomes using the Sigmoid function, which converts numerical results into an expression of probability between 0 and 1.0. Probability is either 0 or 1, depending on whether the event happens or not. For binary predictions, you can divide the population into two groups with a cut-off of 0.5. Everything above 0.5 is considered to belong to group A, and everything below is considered to belong to group B.

You can read further about them [here (https://www.kdnuggets.com/2022/07/logistic-regression-work.html#:~:text=is%20Logistic%20Regression%3F-,Logistic%20regression%20is%20a%20Mac](https://www.kdnuggets.com/2022/07/logistic-regression-work.html#:~:text=is%20Logistic%20Regression%3F-,Logistic%20regression%20is%20a%20Mac)

In [ ]:
```
#Linear Classifier - Logistic Regression
LRClassifer = LogisticRegression()
```

KNN classifier -

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

“Birds of a feather flock together.”

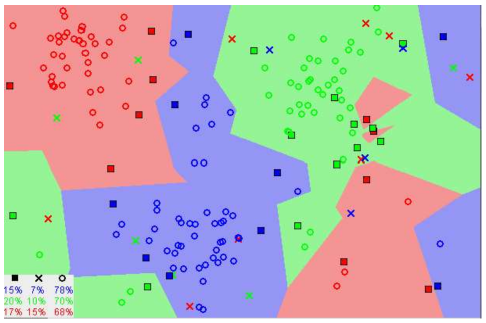


Image showing how similar data points typically exist close to each other Notice in the image above that most of the time, similar data points are close to each other. The KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood— calculating the distance between points on a graph.

We have explained KNN in much more detail in our previous project on Hunting for Exoplanets (https://www.youtube.com/watch?v=y1k2jc3YTeg).

In [ ]:
```
#Nearest Neighbor Classifier
NeNeClassifier = KNeighborsClassifier(n_neighbors=3)
```

In [ ]:
```
#Fitting the models to the dataset
dtClassifer.fit(X_train, y_train)

LRClassifer.fit(X_train, y_train)

NeNeClassifier.fit(X_train, y_train)
```

Out[26]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [ ]:
```
#Getting the prediction set of the models
y_preds = dtClassifer.predict(X_test)
y_predsLR = LRClassifer.predict(X_test)
y_predsNeNe = NeNeClassifier.predict(X_test)
```

```
In [ ]: #Displaying the last 10 predictions on the test split as output by all the mod

        print(y_preds[:10],'\n',y_test[:10])

        print("\n*******************************************************")
        print(y_predsLR[:10],'\n',y_test[:10])

        print("\n*******************************************************")
        print(y_predsNeNe[:10],'\n',y_test[:10])
```

```
[2 0 2 2 0 0 2 1 0 0]
 7054    2
4692    0
907     2
4498    2
9957    0
3341    0
7152    2
1152    1
7105    0
2066    0
Name: class, dtype: int64

*******************************************************
[2 0 2 2 0 0 2 1 0 0]
 7054    2
4692    0
907     2
4498    2
9957    0
3341    0
7152    2
1152    1
7105    0
2066    0
Name: class, dtype: int64

*******************************************************
[2 0 2 2 0 0 2 1 0 0]
 7054    2
4692    0
907     2
4498    2
9957    0
3341    0
7152    2
1152    1
7105    0
2066    0
Name: class, dtype: int64
```

Classification Report

```
In [ ]: from sklearn.metrics import classification_report
        target_names = ['0', '1', '2']

        print('\033[1m  Decision Tree -\n  \033[0m',classification_report(y_preds,y_te
        print('\033[1m  Linear Regression -\n  \033[0m',classification_report(y_predsL
        print("\033[1m  KNN Classifier -\n  \033[0m",classification_report(y_predsNeNe
```

```
  Decision Tree -
               precision    recall  f1-score   support

           0       0.99      0.99      0.99      1504
           1       0.94      0.98      0.96       255
           2       1.00      1.00      1.00      1241

    accuracy                           0.99      3000
   macro avg       0.98      0.99      0.98      3000
weighted avg       0.99      0.99      0.99      3000


  Linear Regression -
               precision    recall  f1-score   support

           0       0.97      0.99      0.98      1469
           1       0.94      0.97      0.96       258
           2       0.99      0.96      0.98      1273

    accuracy                           0.98      3000
   macro avg       0.97      0.97      0.97      3000
weighted avg       0.98      0.98      0.98      3000


  KNN Classifier -
               precision    recall  f1-score   support

           0       0.94      0.89      0.91      1575
           1       0.88      0.97      0.92       240
           2       0.88      0.92      0.90      1185

    accuracy                           0.91      3000
   macro avg       0.90      0.93      0.91      3000
weighted avg       0.91      0.91      0.91      3000
```