ANALYSIS ON STELLAR CLASSIFICATION

Presented by Group 10 :

- J. VANI SUMEDHA
- N. HARIKA
- PAKKI VENKATA TEJA

Bhavan's Vivekananda College | B.Sc Hons Data Science

CONTENTS

- INTRODUCTION
- OBJECTIVE
- CLEANING OF DATA
- EDA
- ANALYSIS
- CONCLUSION

DATA DESCRIPTION

- The data consists of 81,039 observations of space taken by the SDSS (Sloan Digital Sky Survey).
- Every observation is described by 18 feature columns and 1 class column which identifies it to be either a star or galaxy.

OBJECTIVE

Application of different machine learning (ML) algorithms and find the best ML model to classify galaxy or a star.

THE PATH

The team has followed various ML algorithms to find the best model to predict the accuracy for our dataset.

ABOUT THE DATA

The dataset consists of 17 feature columns and 1 target column.
Class is the target variable.

Discrete columns

- run_ID
- rerun_ID
- cam_col
- field_ID
- fiber_ID
- plate
- MJD

Continuous columns

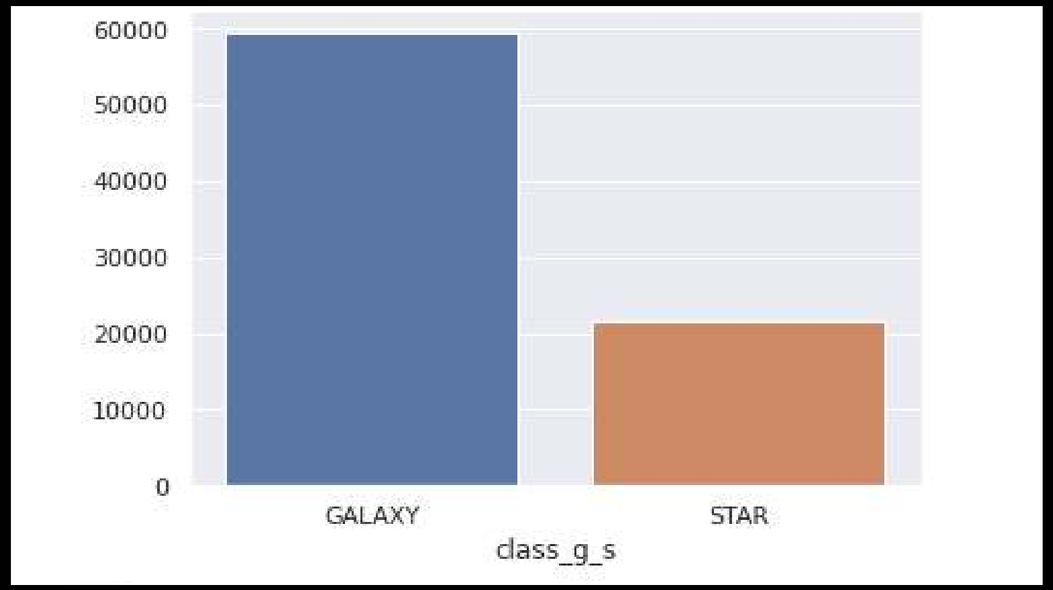
- obj_ID
- alpha
- delta
- u
- g
- r
- i
- z
- spec_obj_ID
- redshift

*Refer to appendix 1

```
obj_ID          0
alpha           0
delta           0
u               0
g               0
r               0
i               0
z               0
run_ID          0
rerun_ID        0
cam_col         0
field_ID        0
spec_obj_ID     0
class_g_s       0
redshift        0
plate           0
MJD             0
fiber_ID        0
dtype: int64
```

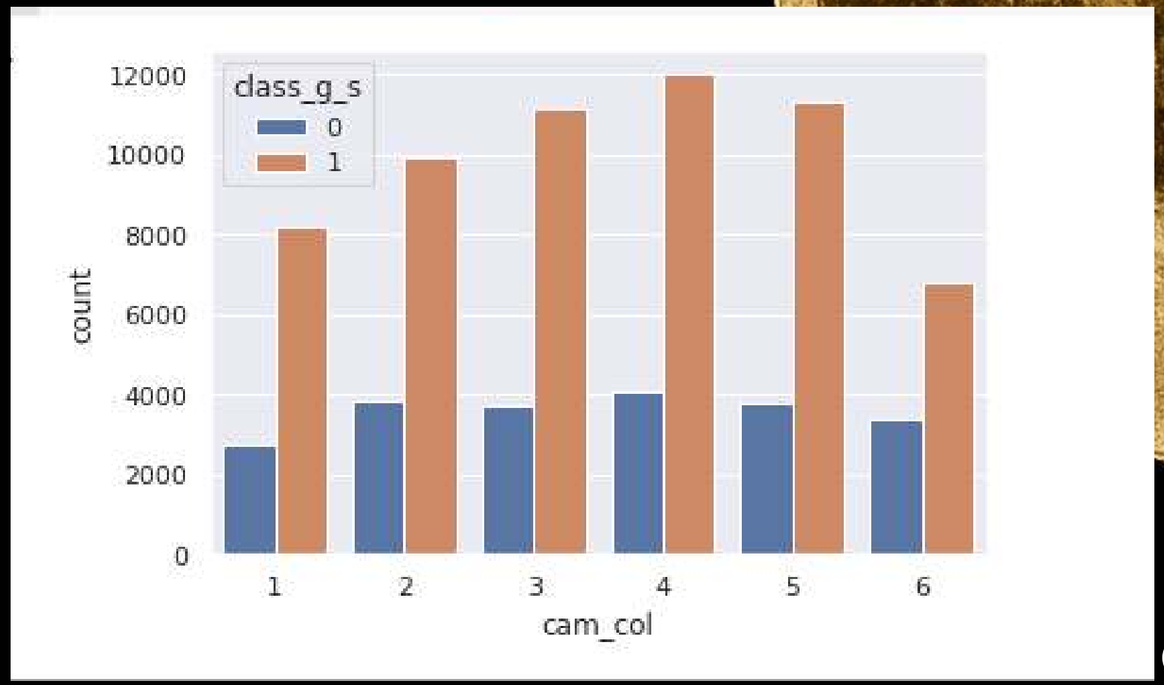
From the original data, there are no missing values or null values.

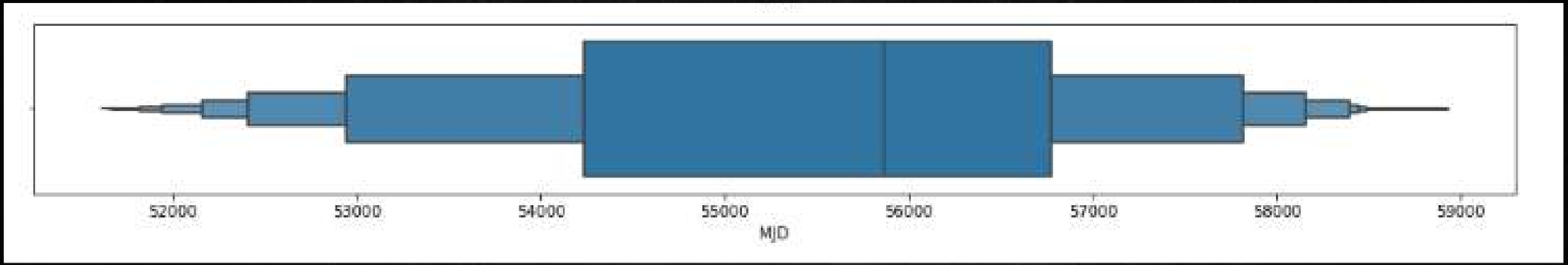
EXPLORATORY DATA ANALYSIS (EDA)



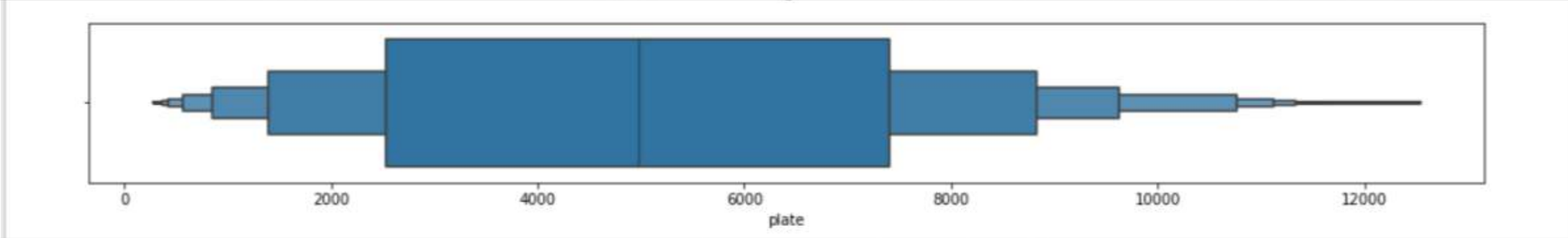
The count of galaxies and stars in our dataset.

The cam col is higher for the galaxy class than star.

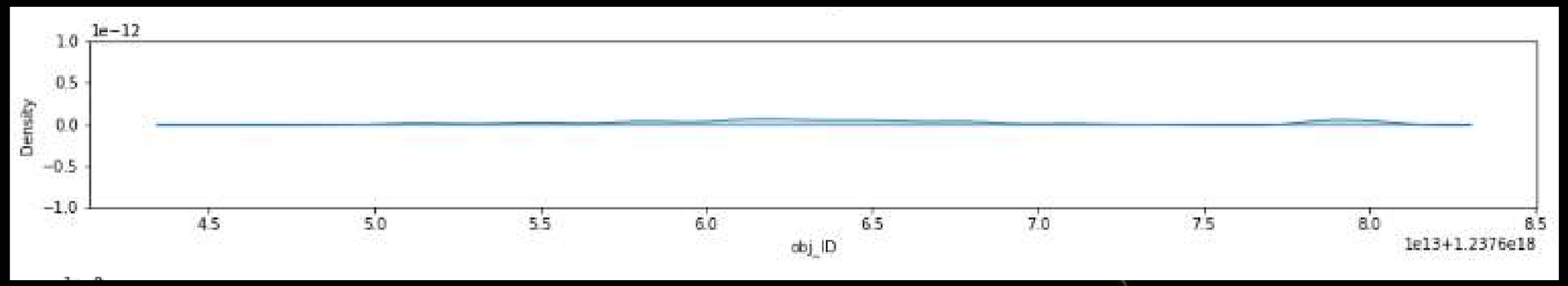




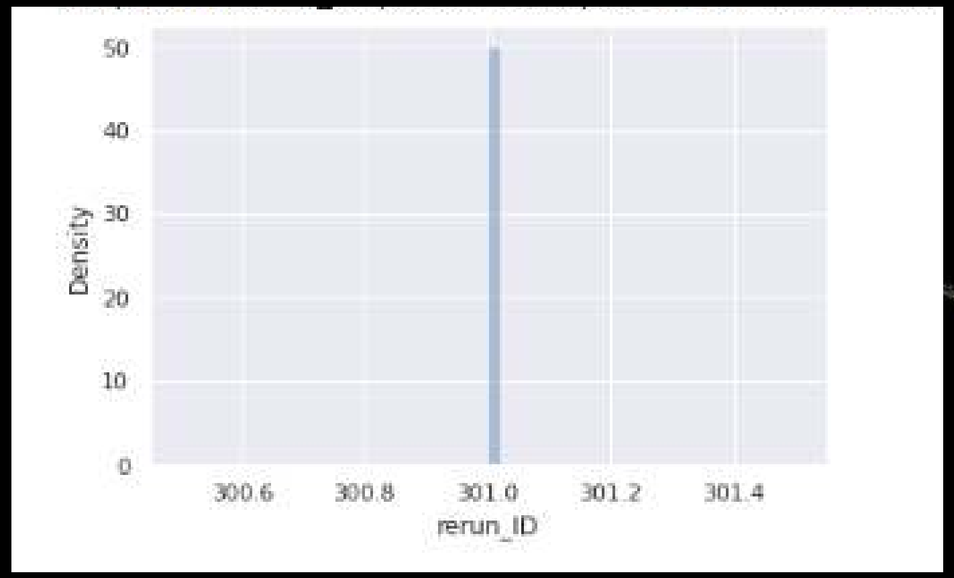
Nearly 54500 - 56800 data points was taken in fiber_id.



Nearly 3000 - 7000 data points are from same spectrum.



The values of object ID is almost zero to the complete data set.



There is only one value in this attribute and by this we cannot distinguish the class.

LOGISTIC REGRESSION

x = independent attributes

y = it contains the target attribute i.e., class

Train/test split	Accuracy
70:30	0.8111
75:25	0.7904
80:20	0.7576
60:40	0.8128

In conclusion, we get the best accuracy with 60:40 ratio for this dataset.

*Refer to appendix 2 & 3

NEURAL NETWORK

• EXPERIMENTATION

layers	epochs	ratio	learning rate	Accuracy	loss
10-7-5-1	100	60 40	0.05	0.7324	0.581
10-7-5-1	200	60 40	0.05	0.7324	0.581
10-7-5-1	300	60 40	0.05	0.7324	0.5811
10-7-5-1	400	60 40	0.05	0.7324	0.5823
10-7-5-1	500	60 40	0.05	0.7324	0.5823
10-8-6-1	100	60 40	0.05	0.7324	0.581
10-8-6-1	200	60 40	0.05	0.7324	0.581
10-8-6-1	300	60 40	0.05	0.7324	0.5811
10-8-6-1	400	60 40	0.05	0.7324	0.5823
10-8-6-1	500	60 40	0.05	0.7324	0.5823

layers	epochs	ratio	learning rate	Accuracy	loss
10-7-5-1	100	75 25	0.05	0.7346	0.5804
10-7-5-1	200	75 25	0.05	0.7346	0.5799
10-7-5-1	300	75 25	0.05	0.7346	0.5799
10-7-5-1	400	75 25	0.05	0.7346	0.5802
10-7-5-1	500	75 25	0.05	0.7346	0.5804
10-8-6-1	100	75 25	0.05	0.7346	0.5804
10-8-6-1	200	75 25	0.05	0.7346	0.5804
10-8-6-1	300	75 25	0.05	0.7346	0.5794
10-8-6-1	400	75 25	0.05	0.7346	0.5798
10-8-6-1	500	75 25	0.05	0.7346	0.5802

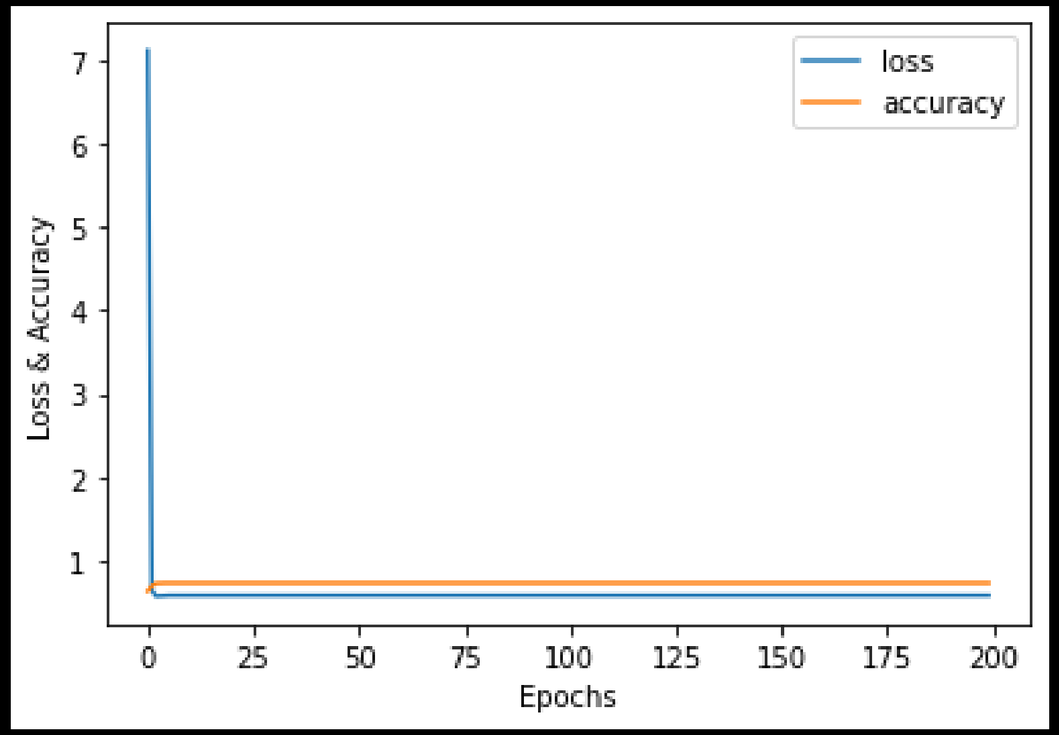
The accuracy and loss values does not change much even with different epochs, layers and learning rate .

optimizer	train test	layers	epochs	accuracy	loss	lr
Adam	60-40	10--7--5--1	100	0.7323	0.5812	0.1
Adam	60-40	10--7--5--1	200	0.7323	0.5812	
Adam	60-40	10--7--5--1	300	0.7323	0.5809	
Adam	60-40	10--7--5--1	400	0.7323	0.5809	
Adam	60-40	10--7--5--1	500	0.7323	0.5809	
Adam	60-40	10--7--5--1	100	0.7323	0.5816	0.01
Adam	60-40	10--7--5--1	200	0.7323	0.5816	
Adam	60-40	10--7--5--1	300	0.7323	0.5808	
Adam	60-40	10--7--5--1	400	0.7323	0.5808	
Adam	60-40	10--7--5--1	500	0.7323	0.5808	
Adam	60-40	10--8--6--1	100	0.7323	0.5816	0.01
Adam	60-40	10--8--6--1	200	0.7323	0.5816	
Adam	60-40	10--8--6--1	300	0.7323	0.5808	
Adam	60-40	10--8--6--1	400	0.7323	0.5808	
Adam	60-40	10--8--6--1	500	0.7323	0.5808	
Adam	60-40	10--8--6--1	100	0.7323	0.5812	0.1
Adam	60-40	10--8--6--1	200	0.7323	0.5812	
Adam	60-40	10--8--6--1	300	0.7323	0.5809	
Adam	60-40	10--8--6--1	400	0.7323	0.5809	
Adam	60-40	10--8--6--1	500	0.7323	0.5809	

optimizer	train test	layers	epochs	accuracy	loss	lr
Adam	70-30	10--7--5--1	100	0.7346	0.5785	0.01
Adam	70-30	10--7--5--1	200	0.7347	0.5785	
Adam	70-30	10--7--5--1	300	0.7347	0.5793	
Adam	70-30	10--7--5--1	400	0.7347	0.5785	
Adam	70-30	10--7--5--1	500	0.7347	0.5785	
Adam	80-20	10--7--5--1	100	0.7334	0.5799	0.01
Adam	80-20	10--7--5--1	200	0.7334	0.5799	
Adam	80-20	10--7--5--1	300	0.7334	0.5799	
Adam	80-20	10--7--5--1	400	0.7334	0.5799	
Adam	80-20	10--7--5--1	500	0.7334	0.5799	
Adam	70-30	10--8--6--1	100	0.7346	0.5785	0.01
Adam	70-30	10--8--6--1	200	0.7347	0.5785	
Adam	70-30	10--8--6--1	300	0.7347	0.5785	
Adam	70-30	10--8--6--1	400	0.7347	0.5785	
Adam	70-30	10--8--6--1	500	0.7347	0.5785	
Adam	80-20	10--8--6--1	100	0.7334	0.5799	0.01
Adam	80-20	10--8--6--1	200	0.7334	0.5799	
Adam	80-20	10--8--6--1	300	0.7334	0.5799	
Adam	80-20	10--8--6--1	400	0.7334	0.5799	
Adam	80-20	10--8--6--1	500	0.7334	0.5799	

optimizer	train test	layers	epochs	accuracy	loss	lr
Adam	70-30	10--7--5--1	100	0.7346	0.5785	0.05
Adam	70-30	10--7--5--1	200	0.7346	0.5785	
Adam	70-30	10--7--5--1	300	0.7346	0.5793	
Adam	70-30	10--7--5--1	400	0.7346	0.5785	
Adam	70-30	10--7--5--1	500	0.7346	0.5785	
Adam	80-20	10--7--5--1	100	0.7334	0.5799	0.05
Adam	80-20	10--7--5--1	200	0.7334	0.5799	
Adam	80-20	10--7--5--1	300	0.7334	0.5799	
Adam	80-20	10--7--5--1	400	0.7351	0.578	
Adam	80-20	10--7--5--1	500	0.7351	0.578	
Adam	70-30	10--8--6--1	100	0.7346	0.5785	0.05
Adam	70-30	10--8--6--1	200	0.7346	0.5785	
Adam	70-30	10--8--6--1	300	0.7346	0.5785	
Adam	70-30	10--8--6--1	400	0.7346	0.5785	
Adam	70-30	10--8--6--1	500	0.7346	0.5785	
Adam	80-20	10--8--6--1	100	0.7334	0.5799	0.05
Adam	80-20	10--8--6--1	200	0.7334	0.5799	
Adam	80-20	10--8--6--1	300	0.7334	0.5799	
Adam	80-20	10--8--6--1	400	0.7334	0.5799	
Adam	80-20	10--8--6--1	500	0.7334	0.5799	

INSIGHTS



The features that makes the neural network model the best :

- Optimizer : Adam
- The architecture 10-7-5-1, 10-8-6-1 is the optimum layer.
- Train and test split ratio - 70 : 30.
- Epochs = 100,200,300,400,500.
- The best accuracy is 0.7347

*Refer to appendix 4, 5 & 6

BAGGING AND BOOSTING

BAGGING

Train / test split	Accuracy
60:40	0.993
70:30	0.993
75:25	0.992
80:20	0.992

*Refer to appendix 7

BOOSTING

ADAPTIVE BOOSTING :

Train/Test Split	Model score (Accuracy)	Learning Rate
70:30	0.9980	0.6
75:25	0.9985	0.6
80:20	0.9985	0.6
60:40	0.9983	0.6

*Refer to appendix 8

GRADIENT BOOSTING :

Train/Test Split	Model score (Accuracy)	Learning Rate
70:30	0.9983	0.6
75:25	0.9987	0.6
80:20	0.9988	0.6
60:40	0.9983	0.6

XGBOOSTING :

Train/Test Split	Model score (Accuracy)	Learning Rate
70:30	0.9990	0.6
75:25	0.9990	0.6
80:20	0.9991	0.6
60:40	0.9989	0.6

*Refer to appendix 9

COMPARISON OF THE MODELS

TRAIN/TEST SPLIT RATIO
60 : 40

Logistic Regression	0.8128
Neural Networks	0.7323
Bagging	0.9930
Adaptive Boosting	0.9983
Gradient Boosting	0.9983
XGBoosting	0.9989

CONCLUSION

- After application of different ML algorithms to the dataset, the best accuracy is given by XG boosting.
- In conclusion XG Boosting is the best fit for the dataset.

Thank You

TEAM MEMBERS :

- J. VANI SUMEDHA
- N. HARIKA
- PAKKI VENKATA TEJA

Colab Link : https://colab.research.google.com/drive/11-oguoGTVzWiav1e5jfg1_dZ_EVxEO2h?usp=sharing

Github Link : https://github.com/harika-unp/UNP-Assignments

APPENDIX

ATTRIBUTES OF THE DATASET

1. obj_ID = Object Identifier, the unique value that identifies the object in the image catalog used by the CAS
2. alpha = Right Ascension angle (at J2000 epoch)
3. delta = Declination angle (at J2000 epoch)
4. u = Ultraviolet filter in the photometric system
5. g = Green filter in the photometric system
6. r = Red filter in the photometric system
7. i = Near Infrared filter in the photometric system
8. z = Infrared filter in the photometric system
9. run_ID = Run Number used to identify the specific scan
10. rereun_ID = Rerun Number to specify how the image was processed
11. cam_col = Camera column to identify the scanline within the run
12. field_ID = Field number to identify each field
13. spec_obj_ID = Unique ID used for optical spectroscopic objects (this means that 2 different observations with the same spec_obj_ID must share the output class)
14. class = object class galaxy, star
15. redshift = redshift value based on the increase in wavelength
16. plate = plate ID, identifies each plate in SDSS
17. MJD = Modified Julian Date, used to indicate when a given piece of SDSS data was taken
18. fiber_ID = fiber ID that identifies the fiber that pointed the light at the focal plane in each observation

PREDICTIVE ANALYSIS OF LOGISTIC REGRESSION

```
x= data_wo_id.drop("class_g_s", axis=1)
y=data_wo_id["class_g_s"]
x.head
```

```
<bound method NDFrame.head of              u         r  redshift      alpha    MJD         i         g  \
0      23.87882  20.39501  0.634794  135.689107  56354  19.16573  22.27530
1      24.77759  22.58444  0.779136  144.826101  58158  21.16812  22.83188
2      25.26307  20.60976  0.644195  142.188790  55592  19.34857  22.66389
3      22.13682  21.61162  0.932346  338.741038  58039  20.50454  23.77656
4      19.43718  16.49747  0.116123  345.282593  56187  15.97711  17.58028
...         ...       ...       ...         ...    ...       ...       ...
81034  22.16759  21.90404  0.000000   39.620709  57749  21.30548  22.97586
81035  22.69118  20.45003  0.404895   29.493819  56934  19.75759  22.38628
81036  21.16916  18.20428  0.143366  224.587407  54535  17.69034  19.26997
81037  25.35039  19.91386  0.455040  212.268621  56368  19.07254  21.63757
81038  22.62171  20.60115  0.542944  196.896053  57104  20.00959  21.79745

              z      delta  plate
0      18.79371  32.494632   5812
1      21.61427  31.274185  10445
2      18.94827  35.582444   4576
3      19.25010  -0.402828   9149
4      15.54461  21.183866   6121
...         ...        ...    ...
81034  20.73569  -2.594074   9374
81035  19.41526  19.798874   7626
81036  17.35221  15.700707   2764
81037  18.62482  46.660365   6751
81038  19.28075  49.464643   7410

[81039 rows x 10 columns]>
```

```
[17] x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0, train_size = .70)
```

```
[18] from sklearn.linear_model import LogisticRegression     #applying logistic regression

     logreg= LogisticRegression(C=1e9)
     logreg.fit(x_train,y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
LogisticRegression(C=1000000000.0)
```

```
[19] y_pred = logreg.predict(x_test)
     y_pred
```

```
array([1, 0, 1, ..., 1, 1, 1])
```

EVALUATION OF THE MODEL
&
CONFUSION MATRIX

```
[20] from sklearn.metrics import r2_score
     r2_score(y_test,y_pred)
```

```
0.02982553341965799
```

```
[21] from sklearn.metrics import mean_squared_error
     mean_squared_error(y_test,y_pred)
```

```
0.18887792036854228
```

```
[22] mean_squared_error(y_test,y_pred,squared=False)
```

```
0.43460087479035553
```

```
[23] from sklearn.metrics import accuracy_score
     accuracy_score(y_test,y_pred)
```

```
0.8111220796314578
```

CONFUSION MATRIX

```
[24] from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test,y_pred)
     cm
```

```
array([[ 3365,  3073],
       [ 1519, 16355]])
```

NEURAL NETWORKS

Analysis

```
[16] x= data_wo_id.drop("class_g_s", axis=1)
     y=data_wo_id["class_g_s"]
     x.head
```

```
<bound method NDFrame.head of              u         z  plate    MJD         i  redshift
0      23.87882  18.79371   5812  56354  19.16573  0.634794  20.39501
1      24.77759  21.61427  10445  58158  21.16812  0.779136  22.58444
2      25.26307  18.94827   4576  55592  19.34857  0.644195  20.60976
3      22.13682  19.25010   9149  58039  20.50454  0.932346  21.61162
4      19.43718  15.54461   6121  56187  15.97711  0.116123  16.49747
...         ...       ...    ...    ...       ...       ...       ...
81034  22.16759  20.73569   9374  57749  21.30548  0.000000  21.90404
81035  22.69118  19.41526   7626  56934  19.75759  0.404895  20.45003
81036  21.16916  17.35221   2764  54535  17.69034  0.143366  18.20428
81037  25.35039  18.62482   6751  56368  19.07254  0.455040  19.91386
81038  22.62171  19.28075   7410  57104  20.00959  0.542944  20.60115

            alpha      delta         g
0      135.689107  32.494632  22.27530
1      144.826101  31.274185  22.83188
2      142.188790  35.582444  22.66389
3      338.741038  -0.402828  23.77656
4      345.282593  21.183866  17.58028
...           ...        ...       ...
81034   39.620709  -2.594074  22.97586
81035   29.493819  19.798874  22.38628
81036  224.587407  15.700707  19.26997
81037  212.268621  46.660365  21.63757
81038  196.896053  49.464643  21.79745

[81039 rows x 10 columns]>
```

```
[24] from sklearn.model_selection import train_test_split
     x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=0.3, random_state=42)
     y_test
```

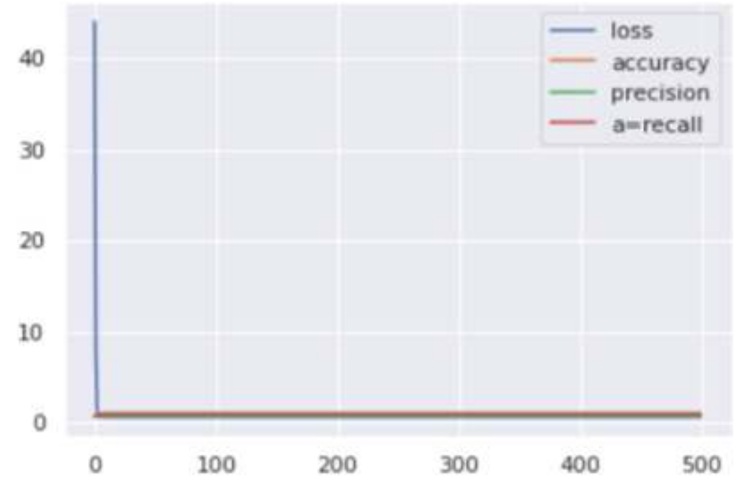
Application of neural network to the complete data
where x contain independent attributes and y contain target attribute.

Neural Network Code

```
tf.random.set_seed(42)
# STEP1: Creating the model

model= tf.keras.Sequential([
                            tf.keras.layers.Dense(10, activation='relu'),
                            tf.keras.layers.Dense(7, activation='relu'),
                            tf.keras.layers.Dense(5, activation='relu'),
                            tf.keras.layers.Dense(1, activation='sigmoid')
])

# STEP2: Compiling the model

model.compile(loss= tf.keras.losses.binary_crossentropy,
              optimizer= tf.keras.optimizers.Adam(lr=0.01),
              metrics= [tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                        tf.keras.metrics.Precision(name='precision'),
                        tf.keras.metrics.Recall(name='a=recall')
              ]
              )

# STEP3: Fit the model

history = model.fit(x_test, y_test, epochs = 500, verbose=0)
```

```
pd.DataFrame(history.history).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd93980d090>
```



The graph of epochs plotted against loss , precision, accuracy and recall

The output of the model for the ratio 70:30

```
[39] model.evaluate(x_test,y_test)

760/760 [==============================] - 1s 1ms/step - loss: 0.5785 - accuracy: 0.7347 - precision: 0.7347 - a=recall: 1.0000
[0.5785295963287354, 0.7346988916397095, 0.7346988916397095, 1.0]
```

BAGGING

```
[ ] from sklearn import datasets
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score
    from sklearn.model_selection import cross_val_score
    from sklearn.ensemble import BaggingClassifier
    from sklearn.tree import DecisionTreeClassifier
```

```
[ ] bgclassifier = BaggingClassifier(base_estimator=None, n_estimators=100,
                                     max_features=10,
                                     max_samples=100,
                                     random_state=1, n_jobs=5)
    bgclassifier.fit(x_train, y_train)
```

```
    BaggingClassifier(max_features=10, max_samples=100, n_estimators=100, n_jobs=5,
                      random_state=1)
```

```
[ ] print('Model test Score: %.3f, ' %bgclassifier.score(x_test, y_test), 'Model training Score: %.3f' %bgclassifier.score(x_train, y_train))
```

```
    Model test Score: 0.993,  Model training Score: 0.992
```

BOOSTING

ADABOOSTING

```
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
```

```
[43] # Create adaboost classifer object
     abc = AdaBoostClassifier(n_estimators=100,
                              learning_rate=0.6)
     # Train Adaboost Classifer
     model = abc.fit(x_train, y_train)

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

```
[44] print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.998025666337611
```

GRADIENT BOOSTING

[15]
```
from sklearn.ensemble import GradientBoostingClassifier
```

[45]
```
clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.6,max_depth=1,
 random_state=0).fit(x_train, y_train)
clf.score(x_test, y_test)
```

```
0.9983135899967095
```

EXTREME GRADIENT BOOSTING

[46]
```
from xgboost import XGBClassifier
```

```
xgb_clf = XGBClassifier()
xgb_clf.fit(x_train, y_train)
```

```
XGBClassifier()
```

```
score = xgb_clf.score(x_test, y_test)
print(score)
```

```
0.9990128331688055
```