STELLAR CLASSIFICATION

BY

G.Manasa

BACKGROUND

Stellar Classification uses the spectral data of stars to categorize them into different categories. The modern stellar classification system is known as the Morgan–Keenan (MK) classification system. It uses the old HR classification system to categorize stars with their chromaticity and uses Roman numerals to categorize the star's size.

OBJECTIVE

To predict the class (i.e Galaxy or Stars) in Stellar Classification. To fit various models and compare the results.

THE PATH

Team followed standard Machine Learning algorithm development process to predict the class.

ABOUT THE DATA

The data set has 18 attributes and 1,00,000 rows.

Categorical Variables	Continuous Variables
class	Alpha, delta, u, g, r
obj_ID	i, z, run_ID, field_ID
rerun_ID	spec_obj_ID, red shift
	plate, MJD, fibre_ID

DESCRIPTION OF ATTRIBUTES

- Obj_id : object Identifier, the unique value that identifies the object in the image catalog used by the CAS.
- Alpha : Right Ascension angle (at J2000 epoch).
- Delta : Declinition angle(at J2000 epoch).
- u : Ultraviolent filter in the photometric system.
- g : Green filter in the photometric system.
- r : Red filter in the photometric system.
- i : Near Infrared filter in the phometric system.
- z : Infrared filter in the photometric system.
- run_iD : Run Number used to identify the specific scan .

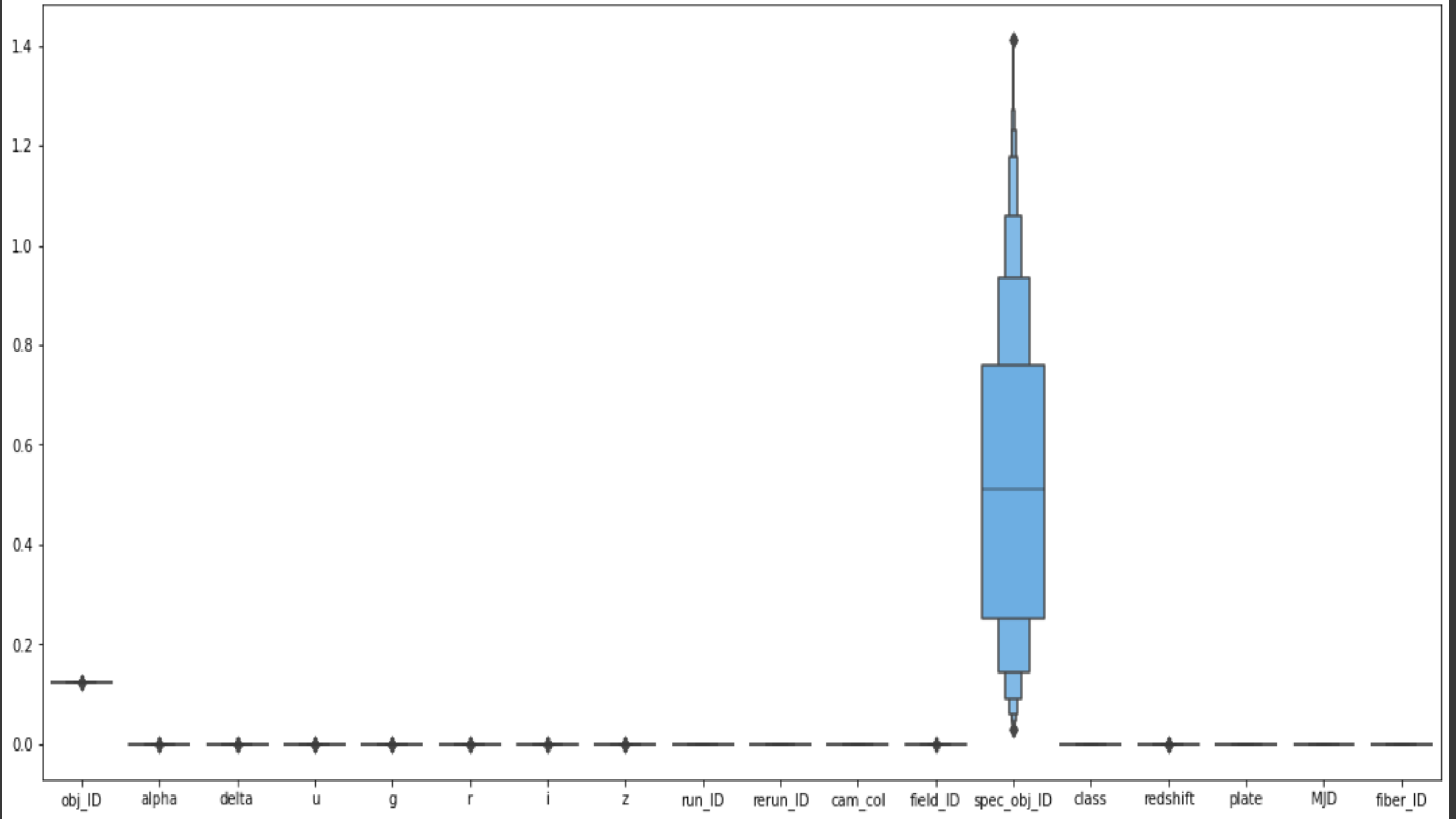
- rereun_ID : Rerun Number to specify how
  the image was processed .
- cam_col : Camera column to identify the scanline within the run .
- field_ID : Field number to identify each field.
- Spec_obj_ID: Unique ID used for optical spectroscopic objects.
- Class : Object class (galaxy,staror quasar object).
- redshift : redshift value based on their increase in wavelength.
- plate : plate ID,identifies each plate in SDSS.
- MJD : Modified Julian Date , used to indicate when a given piece of SDSS data was
  taken.
- Fiber_ID : fiber ID that identifies the fiber that pointed the light at the focal plane in
  each observation.

obj_ID	alpha	delta	u	g	r	i	z	run_ID	rerun_ID	cam_col	field_ID	spec_obj_ID	class	redshift
1.237661e+18	135.689107	32.494632	23.87882	22.27530	20.39501	19.16573	18.79371	3606	301	2	79	6.543777e+18	GALAXY	0.634794
1.237665e+18	144.826101	31.274185	24.77759	22.83188	22.58444	21.16812	21.61427	4518	301	5	119	1.176014e+19	GALAXY	0.779136
1.237661e+18	142.188790	35.582444	25.26307	22.66389	20.60976	19.34857	18.94827	3606	301	2	120	5.152200e+18	GALAXY	0.644195
1.237663e+18	338.741038	-0.402828	22.13682	23.77656	21.61162	20.50454	19.25010	4192	301	3	214	1.030107e+19	GALAXY	0.932346
1.237680e+18	345.282593	21.183866	19.43718	17.58028	16.49747	15.97711	15.54461	8102	301	3	137	6.891865e+18	GALAXY	0.116123
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1.237679e+18	39.620709	-2.594074	22.16759	22.97586	21.90404	21.30548	20.73569	7778	301	2	581	1.055431e+19	GALAXY	0.000000

Data Quality Check

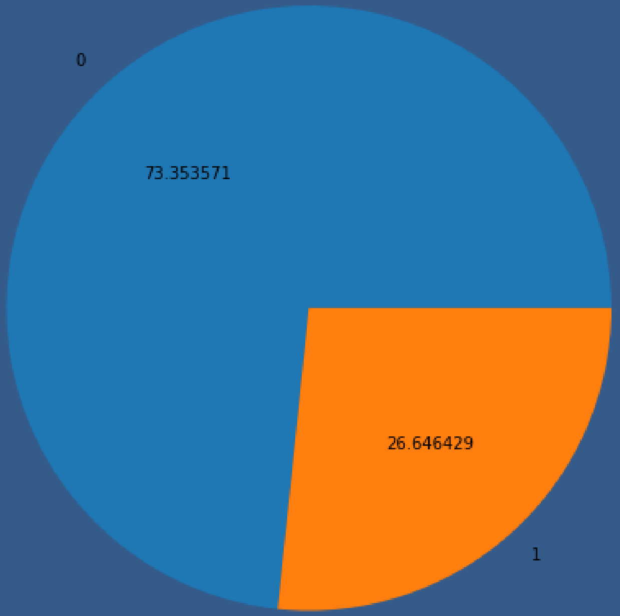
There are no missing values in our stellar classification dataset.

We check for the outliers in the data which might effect our analysis.

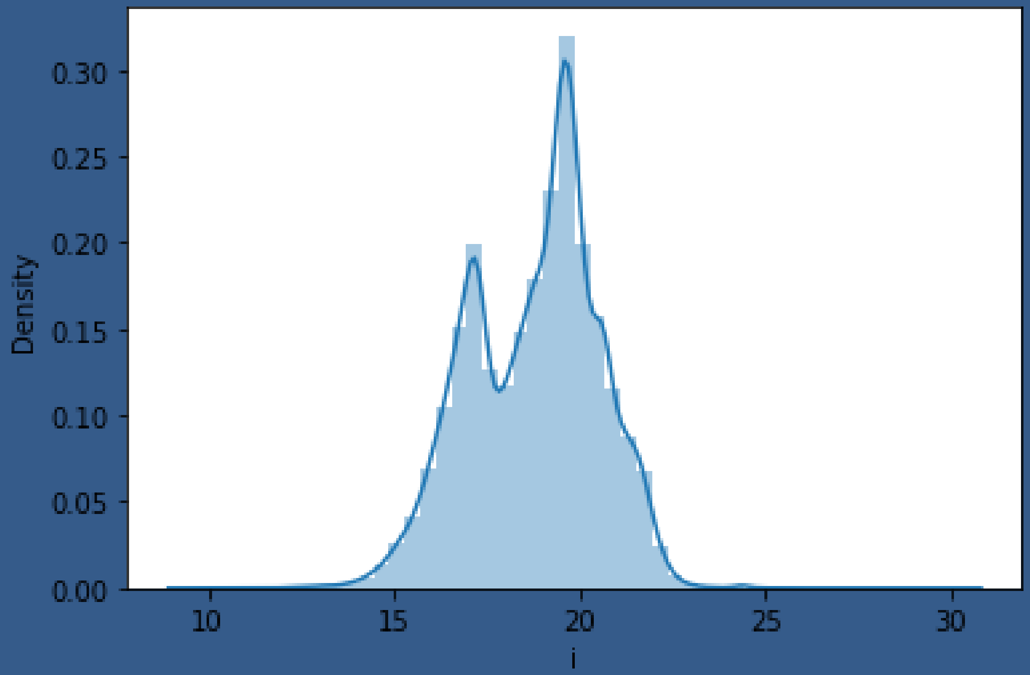


1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
obj_ID
alpha
delta
u
g
r
i
z
run_ID
rerun_ID
cam_col
field_ID
spec_obj_ID
class
redshift
plate
MJD
fiber_ID

EXPLORATORY DATA ANALYSIS

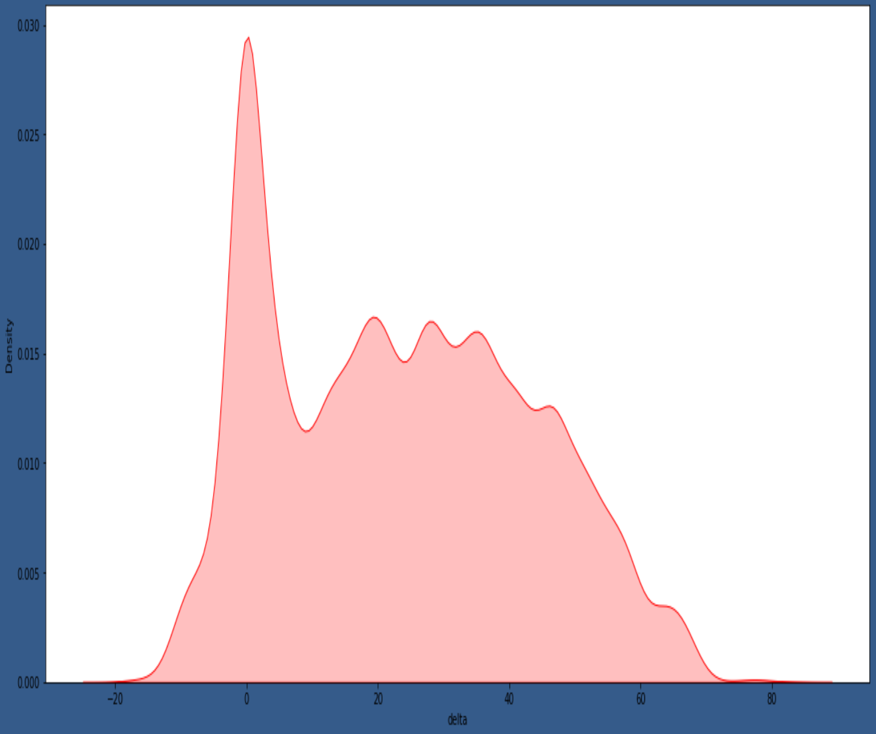


Pie Plot of the Class

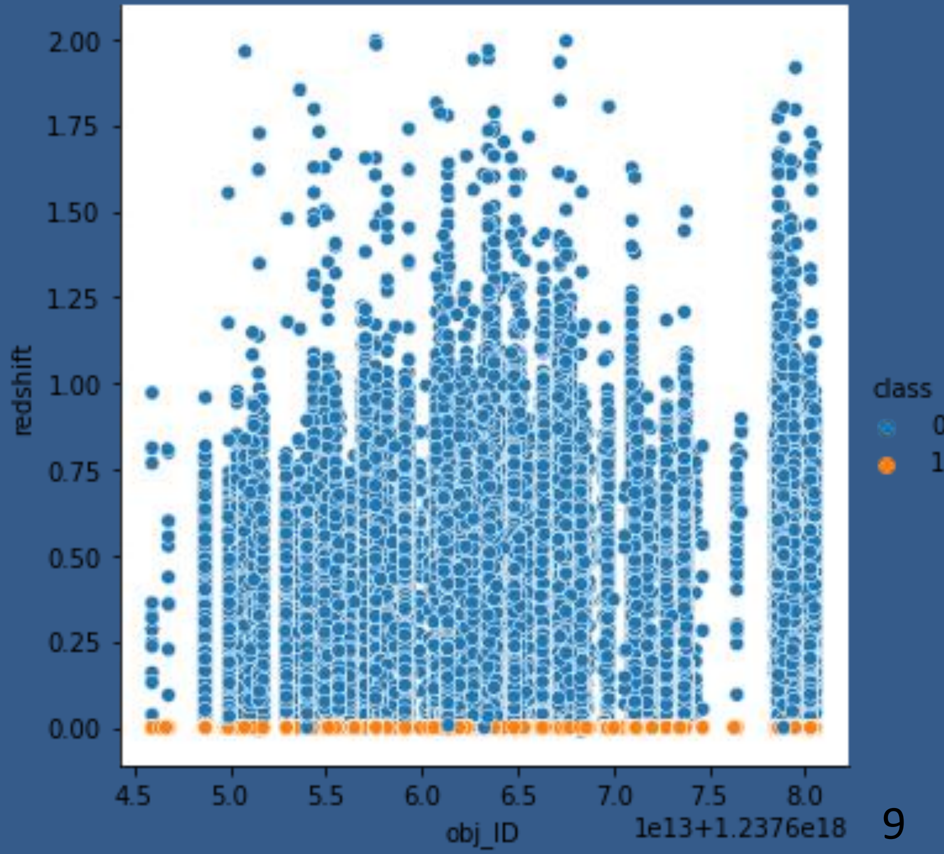


Distplot of i

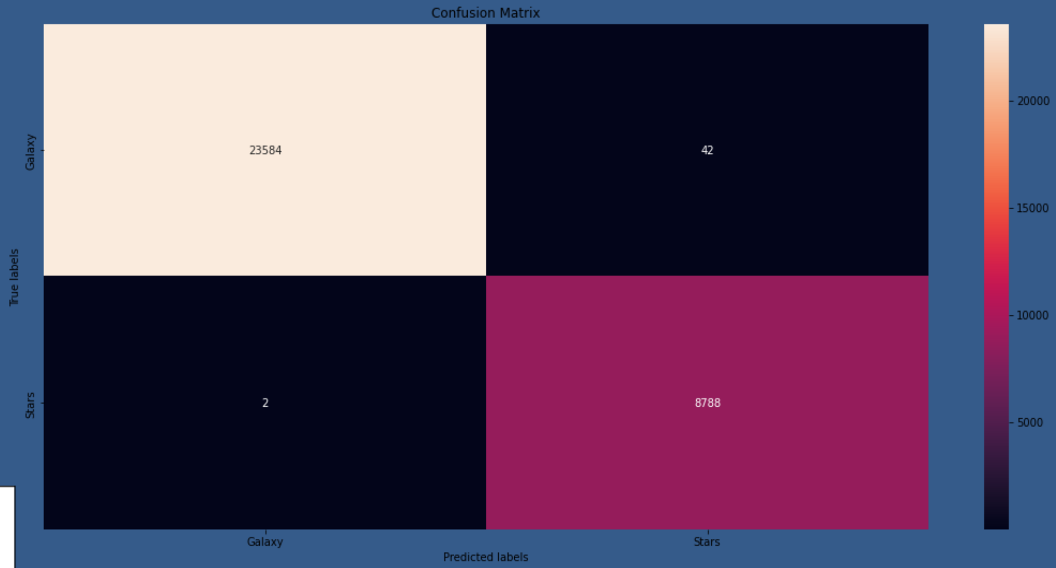
Kdeplot of delta

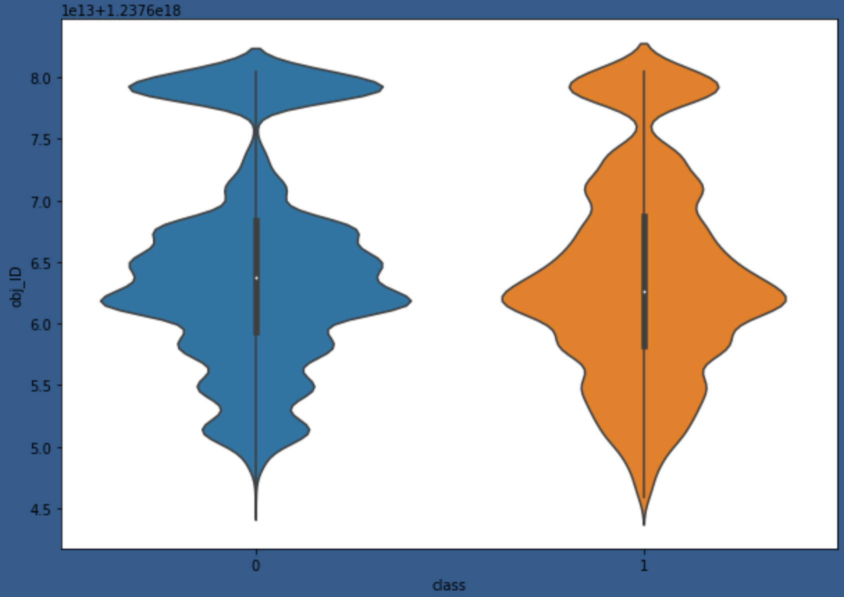


Relational Plot



Confusion Matrix





Violin Plot of Obj_ID

LOGISTIC REGRESSION

Train-Test	Accuracy	MAE
80-20	0.7380	0.2619
70-30	0.7352	0.2648
60-40	0.7340	0.2659
65-35	0.7349	0.2650

Least MAE: 0.2619 for 80-20 ratio

K-NEAREST NEIGHBOR(KNN)

Train-Test	Accuracy	MAE
80-20	0.8346	0.1654
70-30	0.8310	0.1691
60-40	0.8278	0.1722
65-35	0.8297	0.1702

Least
MAE:0.1654
For 80-20
ratio.

SUPPORT VECTOR MACHINE(SVM)

Least MAE:
0.2619 for 80-20
ratio.

Train-Test	Accuracy	MAE
80-20	0.7381	0.2619
70-30	0.7352	0.2648
60-40	0.7340	0.2660
65-35	0.7349	0.2651

NEURAL NETWORK

Architecture	Train-Test	Optimizer	epochs	Accuracy	MAE
17-8-7-2-1	70-30	Adam	100	0.7431	0.5702
17-8-7-2-1	70-30	Adam	300	0.7221	0.5221
17-8-7-2-1	70-30	Adam	500	0.6945	0.5207
17-10-7-1	75-25	Adam	700	0.7331	0.5815
17-10-7-1	75-25	Adam	1000	0.7322	0.5814
5-18-3-2-1	80-20	Adam	700	0.7337	0.5797
5-18-3-2-1	80-20	Adam	1000	0.7337	0.5798

Architecture	Train-Test	Optimizer	epochs	Accuracy	MAE
13-7-6-1	75-25	Adam	100	0.734	0.5794
13-7-6-1	75-25	Adam	500	0.7339	0.5796
10-9-3-6-1	80-20	Adam	700	0.7337	0.5797
10-9-3-6-1	80-20	Adam	1000	0.7337	0.5797
10-9-3-6-1	80-20	Adam	500	0.7337	0.5797

Architecture	Train-test	Optimizer	Epochs	Accuracy	MAE
10-9-3-6-1	80-20	SGD	20	0.7336	0.5797
10-9-3-6-1	70-30	SGD	50	0.7351	0.5782
10-9-3-6-1	60-40	SGD	100	0.7367	0.5765

BAGGING

BOOTSTRAP

Train-Test	Accuracy	MAE
80-20	0.9956	0.0022
70-30	0.9989	0.0010
60-40	0.9987	0.0013
65-35	0.9987	0.0013

Least MAE:0.0010 for 70-30 ratio

MAE for 80-20 ratio is 0.0022

BOOSTING

ADAPTIVE BOOSTING

Least MAE:
0.0056

Train-Test	Accuracy	MAE
80-20	0.9933	0.0067
70-30	0.9944	0.0056
60-40	0.9937	0.0063
65-35	0.9937	0.0063

GRADIENT BOOSTING

Train-Test	Accuracy	MAE
80-20	0.9987	0.0013
70-30	0.9984	0.0016
60-40	0.9986	0.0014
65-35	0.9986	0.0015

Least MAE:
0.0013

EXTREME GRADIENT BOOSTING

Train-Test	Accuracy	MAE
80-20	0.9990	0.0008
70-30	0.9990	0.0010
60-40	0.9990	0.0009
65-35	0.9989	0.0011

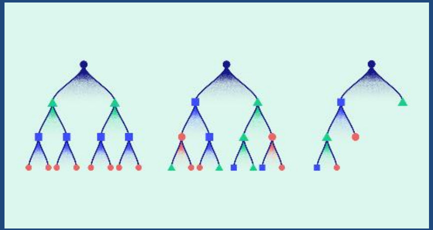
Least MAE: 0.0008 for 80-20 ratio

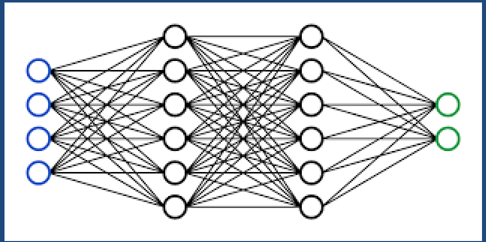
Which Model is Performing Better?

Algorithm	Accuracy	MAE
XGB Boosting	0.9990	0.0008
Gradient Boosting	0.9987	0.0013
Bagging	0.9956	0.0022
Adaptive Boosting	0.9933	0.0067
KNN	0.8346	0.1654
SVM	0.7381	0.2619
Logistic Regression	0.7380	0.2619
Neural Network	0.7337	0.5797

CONCLUSION

After fitting all these models, the best fit for this dataset is Extreme Gradient Boosting with Mean absolute error 0.0008 with an accuracy of 99%.





The highest Mean absolute error is 0.5797 with an accuracy of 73% i.e for Neural Networks.

THANK YOU

Presented By
G.Manasa
K.Akanksha
M.Swarna Lakshmi
T.Varsha

APPENDIX

```
from sklearn.linear_model import LogisticRegression
import statsmodels.formula.api as smf
from sklearn.model_selection import train_test_split
from sklearn import metrics

logreg = LogisticRegression()
logreg.fit(X, y)
```

```
[42] X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0,train_size=0.65)
```

```
[43] lm2 = LogisticRegression()
     lm2.fit(X_train, y_train)
     y_pred = lm2.predict(X_test)

     skl.mean_absolute_error(y_test,y_pred)
```

```
y_pred=logreg.predict(X)
```


```
[42] X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0,train_size=0.65)
```

KNN

```
[45] from sklearn.neighbors import KNeighborsClassifier
     knn = KNeighborsClassifier()
     knn.fit(X_train, y_train)
     y_pred=knn.predict(X_test)
     print(metrics.accuracy_score(y_test,y_pred))
     print(skl.mean_absolute_error(y_test,y_pred))
```

```
0.8297137216189536
0.1702862783810464
```

SVM

```
from sklearn import svm
clf = svm.SVC()
clf.fit(X_train, y_train)
y_pred=clf.predict(X_test)
print(metrics.accuracy_score(y_test,y_pred))
print(skl.mean_absolute_error(y_test,y_pred))
```

3m

```
0.7349457058242843
0.2650542941757157
```

BAGGING

```
from sklearn.ensemble import BaggingClassifier
import sklearn.metrics as metrics
bag_model = BaggingClassifier(
base_estimator=BaggingClassifier(),
n_estimators=100,
max_samples=0.8,
bootstrap=True,
oob_score=True,
random_state=42
)
l=bag_model.fit(X_train, y_train)
```

```
[56] mae = metrics.mean_absolute_error(y_test, l.predict(X_test))

     print("The mean abs error (MAE) on test set: {:.4f}".format(mae))
```

```
The mean abs error (MAE) on test set: 0.0010
```

```
[57] l.score(X_test,y_test)
```

```
0.9989775772105486
```

Adaptive Boosting

[47]
```
#Adaboosting
from sklearn.ensemble import AdaBoostClassifier

adaclf = AdaBoostClassifier(
                            n_estimators=100,
                            learning_rate=0.1,
                            random_state=42)
adaclf.fit(X_train,y_train)
y_pred_1 = adaclf.predict(X_test)
ab=mean_absolute_error(y_test, y_pred_1)
print(ab)
```

```
0.006346072486250176
```

[48]
```
adaclf.score(X_test,y_test)
```

```
0.993653927513749
```

Gradient Boosting

```
[49] from sklearn import datasets, ensemble
     from sklearn.inspection import permutation_importance
     from sklearn.ensemble import GradientBoostingClassifier
     from sklearn.metrics import mean_absolute_error
```

```
[50] #Gradient boosting
     classifier1 = GradientBoostingClassifier(max_depth=4,n_estimators=15,learning_rate=0.1,random_state=0)
     classifier1.fit(X_train, y_train)
```

```
GradientBoostingClassifier(max_depth=4, n_estimators=15, random_state=0)
```

```
[51] y_pred = classifier1.predict(X_test)
     mean_absolute_error(y_test, y_pred)
```

```
0.0014807502467917078
```

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

```
0.9985192497532083
```

XGB

[53]
```
#ExtremeGradient Boosting
from xgboost import XGBClassifier
clf = XGBClassifier(n_estimators=100,
                    learning_rate=0.1,
                    random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
eg=mean_absolute_error(y_test, y_pred)
print(eg)
```

```
0.0010576787477083627
```

[54]
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.9989423212522917
```

```python
import tensorflow as tf
tf.random.set_seed(42)

# STEP1: Creating the model

model= tf.keras.Sequential([
                            tf.keras.layers.Dense(17, activation='relu'),
                            tf.keras.layers.Dense(8, activation='relu'),
                            tf.keras.layers.Dense(7, activation='relu'),
                            tf.keras.layers.Dense(2, 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.SGD(lr=0.01),
              metrics= [tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                        tf.keras.metrics.Precision(name='precision'),
                        tf.keras.metrics.Recall(name='a=recall')
              ]
              )

# STEP1: Fit the model
history= model.fit(X_train, y_train, epochs=20)
```