# Project 8 --- Space Server Dataset.

## Space stuffs excites me a lot…

The Sloan Digital Sky Survey which offers public data of space observations. 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.

**Here with the given dataset we will try to predict whether the stellar object is star, galaxy or quasar.**

## Purpose:

Our universe consists of un-countable number of stellar objects. This model will help the astronomers to distinguish between the star galaxy and quasar automatically (releasing a big chunk of work load from their shoulders), so that they can focus on other valuable stuffs. This will help us to develop autonomous telescopes that scan our sky to find the answers of existence.

# Dataset taken from:

# <https://www.kaggle.com/lucidlenn/sloan-digital-sky-survey>

Skyserver.csv

## Feature details:

* ra, dec — right ascension and declination respectively
* u, g, r, i, z — filter bands (a.k.a. photometric system or astronomical magnitudes)
* run, rerun, camcol, field — descriptors of fields (i.e. 2048 x 1489 pixels) within image
* redshift — increase in wavelength due to motion of astronomical object
* plate — plate number
* mjd — modified Julian date of observation
* fiberid — optic fiber ID
* objid — Object Identifier
* specobjid — Object Identifier
* class — object class (galaxy, star or quasar object) (Target attribute)

This project is done on Jupyter notebook i.e. Python based.

At first we import some basic libraries like pandas to load analysis and manipulation of data, numpy to perform a number of mathematical operations on arrays, seaborn and matplot for EDA visualisations.

*#Import libraries*

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **seaborn** **as** **sns**

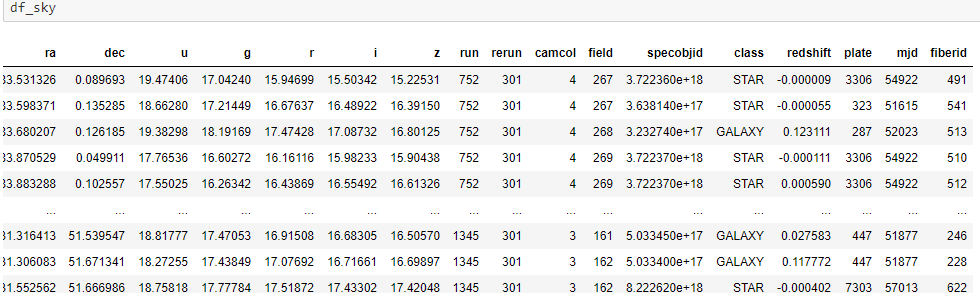
**import** **matplotlib.pyplot** **as** **plt**

Let’s load our dataset:

*#load datset*

df\_sky = pd.read\_csv("Skyserver.csv")

Import the **‘Skyserver.csv’**with the help of pandas.



After loading the dataset we get a glimpse of data. We come to know that our target attribute “class” is categorical i.e. to predict star, galaxy or quasar we need to perform classification i.e. we need to apply classification algorithms on our data..

Studying the dataset:

*#basic insights*

df\_sky.info()

This gives the basic insights of the dataset like the shape of dataset, number of rows and columns, attribute names , datatype of attributes. Count of non null values in each columns, count of each datatype present in dataset, and finaly the memory size consumed by the dataset.

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 10000 entries, 0 to 9999

Data columns (total 18 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 objid 10000 non-null float64

1 ra 10000 non-null float64

2 dec 10000 non-null float64

3 u 10000 non-null float64

4 g 10000 non-null float64

5 r 10000 non-null float64

6 i 10000 non-null float64

7 z 10000 non-null float64

8 run 10000 non-null int64

9 rerun 10000 non-null int64

10 camcol 10000 non-null int64

11 field 10000 non-null int64

12 specobjid 10000 non-null float64

13 class 10000 non-null object

14 redshift 10000 non-null float64

15 plate 10000 non-null int64

16 mjd 10000 non-null int64

17 fiberid 10000 non-null int64

dtypes: float64(10), int64(7), object(1)

memory usage: 1.4+ MB

We see our target attribute is of object type.

df\_sky.describe()

df\_sky.isnull().sum()

df\_sky.nunique()

After running the above lines we come to conclusion that we can drop some columns for better model accuracy.

objid and specobjid are just identifiers so remove them & the features 'run', 'rerun', 'camcol' and 'field' are values which describe parts of the camera at the moment when making the observation, so remove them also.

df\_sky.drop(['specobjid','objid','rerun','field','camcol','run'],axis=1,inplace=**True**)

Let’s check our final dataset for further process i.e. EDA and Model training.



Let’s do some EDA & visualize some inter feature relation in the dataset.

df\_sky["class"].value\_counts().sort\_index()

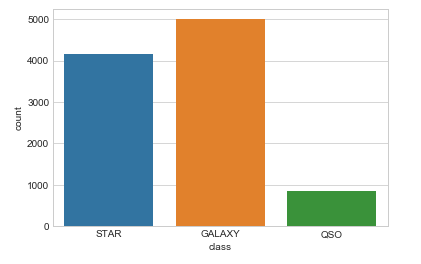
Out[337]:

GALAXY 4998

QSO 850

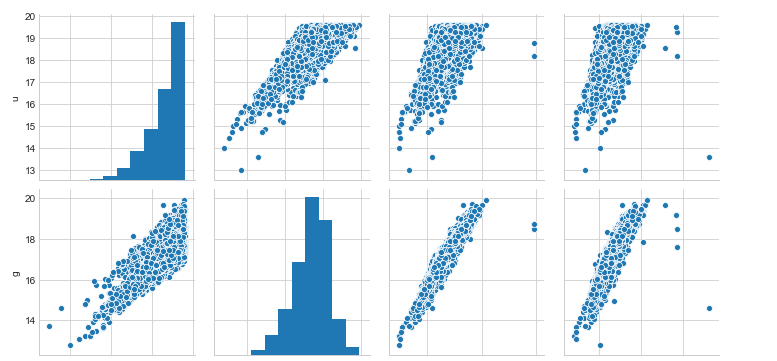
STAR 4152

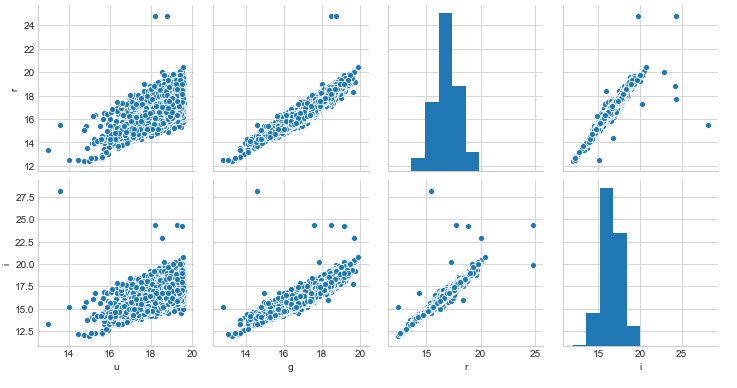
We see the most abundant objects (50%) are galaxies, a little less (40%) are stars and only around (10%) of the rows are classified as QSOs.



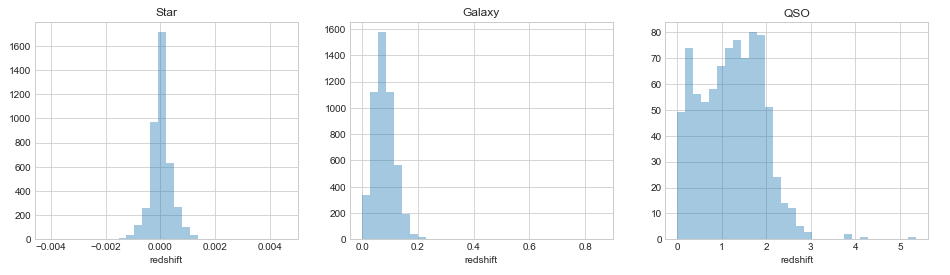
Below we plot the inter-relation between the spectrum data received.

sns.pairplot(df\_sky[['u','g','r','i','class']])

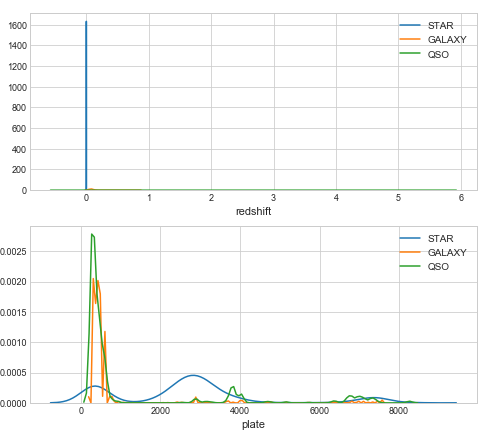
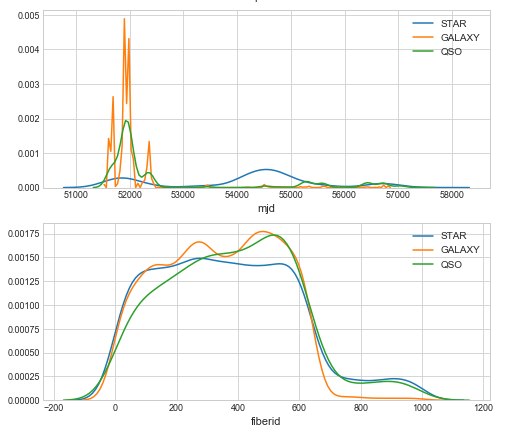




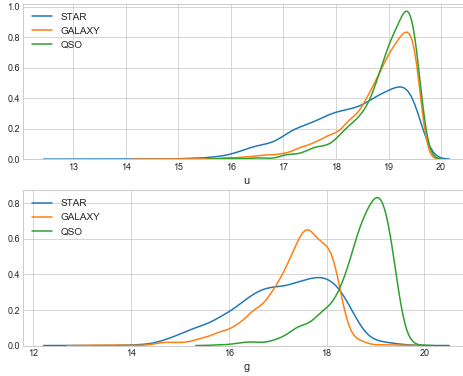
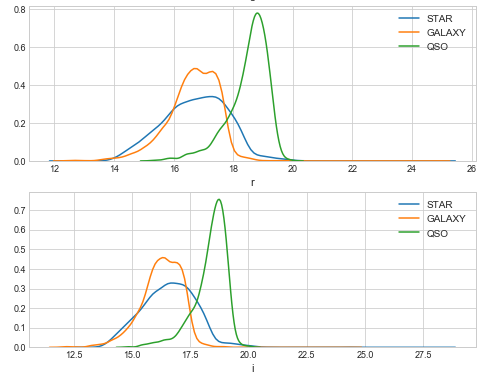
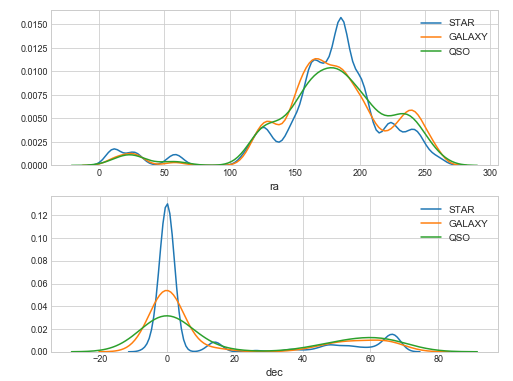
We see a strong relation between the various spectrum data received. With the help of these spectrum we identify stellar objects as galaxy, star or quasar.

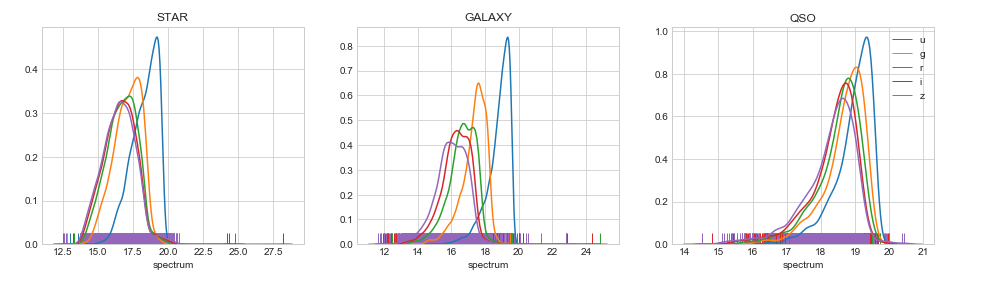


The above plot shows the redshift comparison of star galaxy and quasar. Lower redshifts are stars and higher redshift readings are quasars.

Here we see stars have 0 redshifts, galaxy have strong mjd readings, plate has high relation with quasars.

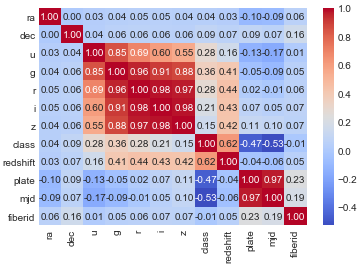


In the above plot we see how the spectrum showing pattern in different stellar objects.

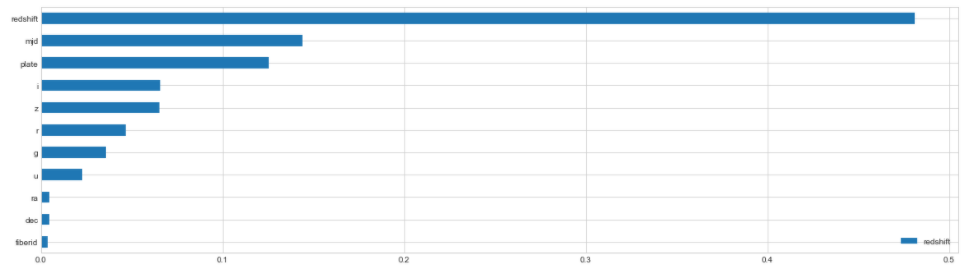
Now let’s convert our only object target feature into int data type for model training and further process.

df\_sky['class']=df\_sky['class'].map({'STAR':0,'GALAXY':1,'QSO':2}).astype(int)

Now we plot correlation heat map to find the intensity of relation between each feature..



Using Random Forest to gain an insight on Feature Importance



Data preprocessing:

Let’s standardize the data using standardscaler and segregate the data into input and output.

*# Apply Scaling*

**from** **sklearn** **import** preprocessing

std\_scale = preprocessing.StandardScaler().fit(df\_sky.drop('class', axis=1))

X = std\_scale.transform(df\_sky.drop('class', axis=1))

Y = df\_sky['class']

Now split the data into train and test sets.

*#train test split*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.20, random\_state=44, shuffle =**True**)

Apply classifiers:

* *K Nearest Neighbors*

knn = KNeighborsClassifier()

training\_start = time.perf\_counter()

knn.fit(X\_train, y\_train)

training\_end = time.perf\_counter()

prediction\_start = time.perf\_counter()

preds = knn.predict(X\_test)

prediction\_end = time.perf\_counter()

acc\_knn = (preds == y\_test).sum().astype(float) / len(preds)\*100

knn\_train\_time = training\_end-training\_start

knn\_prediction\_time = prediction\_end-prediction\_start

print("Scikit-Learn's K Nearest Neighbors Classifier's prediction accuracy is: **%3.2f**" % (acc\_knn))

print("Time consumed for training: **%4.3f** seconds" % (knn\_train\_time))

print("Time consumed for prediction: **%6.5f** seconds" % (knn\_prediction\_time))

Scikit-Learn's K Nearest Neighbors Classifier's prediction accuracy is: 92.60

Time consumed for training: 0.757 seconds

Time consumed for prediction: 0.33619 seconds

* *Naive Bayes*

gnb = GaussianNB()

training\_start = time.perf\_counter()

gnb.fit(X\_train, y\_train)

training\_end = time.perf\_counter()

prediction\_start = time.perf\_counter()

preds = gnb.predict(X\_test)

prediction\_end = time.perf\_counter()

acc\_gnb = (preds == y\_test).sum().astype(float) / len(preds)\*100

gnb\_train\_time = training\_end-training\_start

gnb\_prediction\_time = prediction\_end-prediction\_start

print("Scikit-Learn's Gaussian Naive Bayes Classifier's prediction accuracy is: **%3.2f**" % (acc\_gnb))

print("Time consumed for training: **%4.3f** seconds" % (gnb\_train\_time))

print("Time consumed for prediction: **%6.5f** seconds" % (gnb\_prediction\_time))

Scikit-Learn's Gaussian Naive Bayes Classifier's prediction accuracy is: 97.05

Time consumed for training: 0.479 seconds

Time consumed for prediction: 0.13537 seconds

* *XGBoost*

xgb = XGBClassifier(n\_estimators=100)

training\_start = time.perf\_counter()

xgb.fit(X\_train, y\_train)

training\_end = time.perf\_counter()

prediction\_start = time.perf\_counter()

preds = xgb.predict(X\_test)

prediction\_end = time.perf\_counter()

acc\_xgb = (preds == y\_test).sum().astype(float) / len(preds)\*100

xgb\_train\_time = training\_end-training\_start

xgb\_prediction\_time = prediction\_end-prediction\_start

print("XGBoost's prediction accuracy is: **%3.2f**" % (acc\_xgb))

print("Time consumed for training: **%4.3f**" % (xgb\_train\_time))

print("Time consumed for prediction: **%6.5f** seconds" % (xgb\_prediction\_time))

XGBoost's prediction accuracy is: 99.35

Time consumed for training: 2.687

Time consumed for prediction: 0.03142 seconds

* *Scitkit-Learn's Random Forest Classifier*

rfc = RandomForestClassifier(n\_estimators=10)

training\_start = time.perf\_counter()

rfc.fit(X\_train, y\_train)

training\_end = time.perf\_counter()

prediction\_start = time.perf\_counter()

preds = rfc.predict(X\_test)

prediction\_end = time.perf\_counter()

acc\_rfc = (preds == y\_test).sum().astype(float) / len(preds)\*100

rfc\_train\_time = training\_end-training\_start

rfc\_prediction\_time = prediction\_end-prediction\_start

print("Scikit-Learn's Random Forest Classifier's prediction accuracy is: **%3.2f**" % (acc\_rfc))

print("Time consumed for training: **%4.3f** seconds" % (rfc\_train\_time))

print("Time consumed for prediction: **%6.5f** seconds" % (rfc\_prediction\_time))

Scikit-Learn's Random Forest Classifier's prediction accuracy is: 99.10

Time consumed for training: 0.442 seconds

Time consumed for prediction: 0.00416 seconds

* *Support Vector Machine Classifier*

svc = SVC()

training\_start = time.perf\_counter()

svc.fit(X\_train, y\_train)

training\_end = time.perf\_counter()

prediction\_start = time.perf\_counter()

preds = svc.predict(X\_test)

prediction\_end = time.perf\_counter()

acc\_svc = (preds == y\_test).sum().astype(float) / len(preds)\*100

svc\_train\_time = training\_end-training\_start

svc\_prediction\_time = prediction\_end-prediction\_start

print("Scikit-Learn's Support Vector Machine Classifier's prediction accuracy is: **%3.2f**" % (acc\_svc))

print("Time consumed for training: **%4.3f** seconds" % (svc\_train\_time))

print("Time consumed for prediction: **%6.5f** seconds" % (svc\_prediction\_time))

Scikit-Learn's Support Vector Machine Classifier's prediction accuracy is: 96.75

Time consumed for training: 1.150 seconds

Time consumed for prediction: 0.10702 seconds

* *LGBMClassifier*

lgb = LGBMClassifier()

training\_start = time.perf\_counter()

lgb.fit(X\_train, y\_train)

training\_end = time.perf\_counter()

prediction\_start = time.perf\_counter()

preds = lgb.predict(X\_test)

prediction\_end = time.perf\_counter()

acc\_lgb = (preds == y\_test).sum().astype(float) / len(preds)\*100

lgb\_train\_time = training\_end-training\_start

lgb\_prediction\_time = prediction\_end-prediction\_start

print("Scikit-Learn's LGBMClassifier's prediction accuracy is: **%3.2f**" % (acc\_lgb))

print("Time consumed for training: **%4.3f** seconds" % (lgb\_train\_time))

print("Time consumed for prediction: **%6.5f** seconds" % (lgb\_prediction\_time))

Scikit-Learn's LGBMClassifier's prediction accuracy is: 99.15

Time consumed for training: 1.057 seconds

Time consumed for prediction: 0.02211 seconds

* *DecisionTreeClassifier*

dtc = DecisionTreeClassifier()

training\_start = time.perf\_counter()

dtc.fit(X\_train, y\_train)

training\_end = time.perf\_counter()

prediction\_start = time.perf\_counter()

preds = dtc.predict(X\_test)

prediction\_end = time.perf\_counter()

acc\_dtc = (preds == y\_test).sum().astype(float) / len(preds)\*100

dtc\_train\_time = training\_end-training\_start

dtc\_prediction\_time = prediction\_end-prediction\_start

print("Scikit-Learn's DecisionTreeClassifier's prediction accuracy is: **%3.2f**" % (acc\_dtc))

print("Time consumed for training: **%4.3f** seconds" % (dtc\_train\_time))

print("Time consumed for prediction: **%6.5f** seconds" % (dtc\_prediction\_time))

Scikit-Learn's DecisionTreeClassifier's prediction accuracy is: 98.70

Time consumed for training: 0.123 seconds

Time consumed for prediction: 0.00058 seconds

* *LogisticRegression*

lr = LogisticRegression()

training\_start = time.perf\_counter()

lr.fit(X\_train, y\_train)

training\_end = time.perf\_counter()

prediction\_start = time.perf\_counter()

preds = lr.predict(X\_test)

prediction\_end = time.perf\_counter()

acc\_lr = (preds == y\_test).sum().astype(float) / len(preds)\*100

lr\_train\_time = training\_end-training\_start

lr\_prediction\_time = prediction\_end-prediction\_start

print("Scikit-Learn's LogisticRegression's prediction accuracy is: **%3.2f**" % (acc\_lr))

print("Time consumed for training: **%4.3f** seconds" % (lr\_train\_time))

print("Time consumed for prediction: **%6.5f** seconds" % (lr\_prediction\_time))

Scikit-Learn's LogisticRegression's prediction accuracy is: 97.80

Time consumed for training: 0.756 seconds

Time consumed for prediction: 0.00046 seconds

* *MLPClassifier*

MLP = MLPClassifier()

training\_start = time.perf\_counter()

MLP.fit(X\_train, y\_train)

training\_end = time.perf\_counter()

prediction\_start = time.perf\_counter()

preds = MLP.predict(X\_test)

prediction\_end = time.perf\_counter()

acc\_MLP = (preds == y\_test).sum().astype(float) / len(preds)\*100

MLP\_train\_time = training\_end-training\_start

MLP\_prediction\_time = prediction\_end-prediction\_start

print("Scikit-Learn's MLPClassifier's prediction accuracy is: **%3.2f**" % (acc\_MLP))

print("Time consumed for training: **%4.3f** seconds" % (MLP\_train\_time))

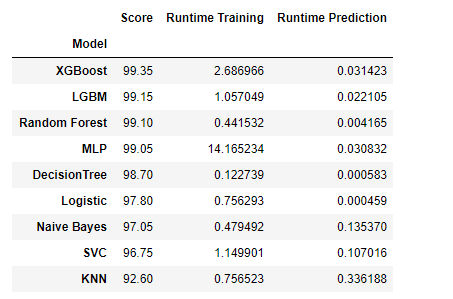
print("Time consumed for prediction: **%6.5f** seconds" % (MLP\_prediction\_time))

Scikit-Learn's MLPClassifier's prediction accuracy is: 99.05

Time consumed for training: 14.165 seconds

Time consumed for prediction: 0.03083 seconds

Let's compare the results:



We can see that both XGBoost and Scikit-Learn's LGBM Classifier could achieve very high accuracy.

We will now perform k fold cross valdiation for the top 2 classifiers, i.e. XGBoost & LGBM.

We do this to get a more realistic result by testing the performance for 10 different train and test datasets and averaging the results. Cross validation ensures that the above result is not arbitary and gives a more reliable performance check.

* *LGBM*

lgb\_cv = LGBMClassifier(n\_estimators=100)

scores = cross\_val\_score(lgb\_cv, X\_train, y\_train, cv=10, scoring = "accuracy")

print("Scores:", scores)

print("Mean:", scores.mean())

print("Standard Deviation:", scores.std())

Scores: [0.98375 0.9825 0.99 0.9925 0.99375 0.9875 0.9925 0.99375 0.98125 0.99375]

Mean: 0.9891249999999999

Standard Deviation: 0.004745063224025589

* *XGBoost*

xgb\_cv = XGBClassifier(n\_estimators=100)

scores = cross\_val\_score(xgb\_cv, X\_train, y\_train, cv=10, scoring = "accuracy")

print("Scores:", scores)

print("Mean:", scores.mean())

print("Standard Deviation:", scores.std())

Scores: [0.98625 0.98375 0.99 0.9925 0.9925 0.985 0.9925 0.99375 0.9875 0.9925 ]

Mean: 0.989625

Standard Deviation: 0.0034932971531205495

Finally XGBoost showed a higher mean and lower standard deviation than the Scikit-Learn LGBM. A high mean corresponds to a more stable performance and a low standard deviation corresponds to smaller range of results.

Let’s try parameter tuning for higher accuracy.

XGBoost - Testing optimal hyperparameters

xgboost = XGBClassifier(max\_depth=5, learning\_rate=0.01, n\_estimators=100, gamma=0, min\_child\_weight=1, subsample=0.8, colsample\_bytree=0.8, reg\_alpha=0.005)

xgboost.fit(X\_train, y\_train)

preds = xgboost.predict(X\_test)

accuracy = (preds == y\_test).sum().astype(float) / len(preds)\*100

print("XGBoost's prediction accuracy WITH optimal hyperparameters is: **%3.2f**" % (accuracy))

XGBoost's prediction accuracy WITH optimal hyperparameters is: 99.05

The parameter tuning did not improve the accuracy as excpected.

Let’s plot Confusion Matrix:

*#'STAR':0,'GALAXY':1,'QSO':2*

unique, counts = np.unique(df\_sky['class'], return\_counts=**True**)

dict(zip(unique, counts))

{0: 4152, 1: 4998, 2: 850}

predictions = cross\_val\_predict(xgb, df\_sky.drop('class', axis=1), df\_sky['class'], cv=3)

confusion\_matrix(df\_sky['class'], predictions)

array([[4148, 4, 0],

[ 8, 4964, 26],

[ 1, 48, 801]], dtype=int64)

The first row shows that out of 4152 stars, 4148 were classified correctly as stars. 4 stars were classified incorrectly as galaxies.

The second row shows out of 4998 galaxies 4964 were classified correctly. 26 galaxies were classified incorrectly as qsos and 8 galaxies were classified as stars.

The last row tells us that out of 850 quasars 801 were classified correctly.. 1 qsos was classified incorrectly as star and 48 qsos were classified as galaxies.

In total: We have only 87 objects which were classified incorrectly. Most of the objects were recognized as what they are.

We see XGB to perform the best among other models.

Need to save the model which we have prepared so far.

To do that we need to pickle the model.

*#save the best model.*

**import** **pickle**

filename='space.pkl'

M=open(filename,'wb')

pickle.dump(xgb, M)

M.close()

**To save your .Ipnyb file in form of executable, save the same as .py file**.

### **To check model performance on totally new data set with same features.**

Now we have a totally new data set which has same feature as per previous data set but contain different values.

**Note –** To do that your executable file ‘**model’ and ‘.py’ file should be in same folder.**

**Check out my work @**

<https://nbviewer.jupyter.org/github/HemantPatar/Project-DYnamics-M20/blob/main/Project%208%20%28Space%20Server%20Dataset%29.ipynb>

**Thank you ☺**