Exploring Machine Learning Model for Forecasting Asteroid Probability - Nearby or Away from Earth

RISHIKESH ANIL WAGHELA
Masters in Data Analytics
Dublin Business School
Dublin, Ireland
10628916@dbs.ie

SHREYA SURESH PADTE
Masters in Data Analytics
Dublin Business School
Dublin, Ireland
10634936@dbs.ie

SAURABH SANTOSH GUPTA
Masters in Data Analytics
Dublin Business School
Dublin, Ireland
10628050@dbs.ie

Abstract

The amount of data throughout space exploration is growing daily and is getting increasingly comprehensive than those in past years. In consideration of the numerous asteroids throughout the universe. The proximity to several of them surprises us. Hence, such asteroids may end up being destructive. As a necessary consequence, it is reasonable to understand which asteroids are close to Earth and which are not. The accessible data on the attributes of asteroids allows this analysis to examine the connectivity of asteroids to Earth and implies a Supervised Machine Learning Technique to determine if an asteroid stands nearby or far away. Applying Logistic Regression with UnderSampling technique in collaboration with hyperparameter tuning led to a significant accuracy of 100%.

Introduction

Asteroids are stone leftovers from the beginning phases of the solar system's creation, which occurred around 4.6 billion years ago. They are also known as minute planets. There are 1,267,131 identified asteroids as of right present according to **("In Depth | Asteroids," n.d.)**. Machine learning is the process of attempting to learn through information in order to develop insights and conclusions about it. After that, this evaluation can be utilized to forecast. The capacity for processing large datasets and the rate of computing have increased exponentially with in past decade. Such innovation gives academics a broader possibility to apply Machine Learning Techniques among a wide range of fields. Inside the expanding subject of space exploration, this study describes one well implementation of Machine Learning. Such an area which produces an enormous quantity of information includes cosmology. Experts estimate to be huge amount of asteroids, with sizes ranging between thousands of meters to less than 1 meter **(Hefele et al., 2020)**. Even though only a small minority of relatively close to Earth asteroids are nearly sufficient to the Mankind to inflict harm, whereas if asteroid is large sufficient, this could wipe out a big territory.

Research Question

1]. What cosmic variables can boost or enhance the likelihood of accurately forecasting whether an asteroid is nearby or far away?

2]. How effectively can Logistic Regression or SGD Classifier forecast the possibility of asteroid being nearby or far away?

3]. How efficiently the model performs while using sampling techniques?

Data Sources

The dataset named "Asteroid" was utilized throughout this analysis. This dataset was formally administered by the California Institute of Technology's Jet Propulsion Laboratory, a unit of NASA. Through Kaggle, the specific dataset was spotted and downloaded. **("Asteroid Dataset," n.d.)**. This dataset provides over 900,000 rows and 45 columns.

Literature Review

(Malakouti, 2023), this report concentrates on categorizing threatening asteroids using multiple methods of machine learning. The study examined the record of NASA-certified asteroids classified as NEO (Near Earth Object). Machine Learning models such as – Random Forest, Gradient, Extra Tree, Light Gradient Boosting and Ada Boost were used to anticipate the risks posed by asteroids throughout the Earth's surroundings. The 90,836 asteroids investigated under this study were 70,000 kilometers away. The study ended up getting conclusion the best resilient efficiency was associated to the Random Forest Method while the worst efficiency was associated towards the Ada boost method.

The author **(Hossain and Zabed, 2023)**, deployed Machine Learning Techniques towards Asteroid Identification and Diameter Estimation. In the words of the author, there seem to be millions of asteroids, which makes it important to classify them and calculate overall diameter in order to understand various features. This will assist them in identifying asteroids that may be dangerous to Mankind. K-nearest neighbors classifier, logistic regression classifier, SGD classifier, and XG - Boost classifier techniques were used to accomplish the identification assignment. K-nearest neighbors, Linear, Logistic Regression, Random Forest and models involving neural networks were used to estimate diameter. Through utilizing these techniques, the study was able to identify asteroids with a 99.99 percentage prediction performance.

Methodology

Crafting decisions based on data gathered across diverse sources has recently led to advancements and significant financial gains. The CRISP-DM (Cross Industry Standard Procedure for Data Mining), which had been designed around 1996, is used in the large portion of data analysis initiatives. We employed the CRISP-DM mechanism to carry out the development procedure. CRISP-DM includes Six steps, which is illustrated in Fig 1. This is a description of how each level of the CRISP-DM methodology's project implementation progressed.

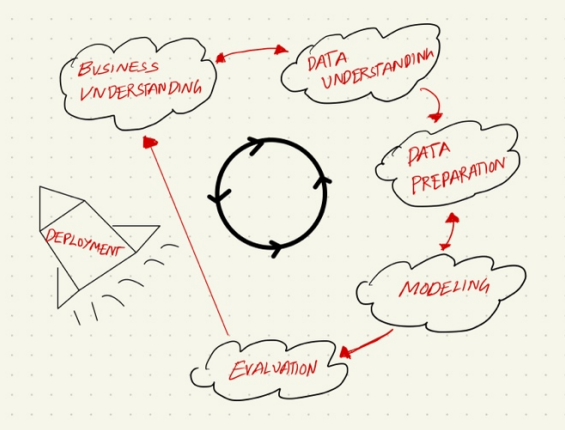


Fig 1: CRISP-DM

BUSINESS UNDERSTANDING:

Only by having a thorough understanding of the business will you be able to pinpoint exactly what its current challenges are, how to evaluate them and find a solution (or solutions), and what tactics to use to achieve the objectives. Because of the potential harm that asteroids could cause if they hit Earth, there has been a lot of study into predicting their distance. However, the field of predicting asteroids' distance is at best rudimentary due to a lack of adequate data, the inability to approach the actual asteroids, and the lack of a precise formula for doing so.

Any sufficiently large asteroid or impact would result in numerous firestorms or massive tsunamis, depending on where it makes contact. A robust Supervised Machine Learning Models that can determine whether an asteroid is close could help humanity overcome these limitations and avert catastrophic destruction.

DATA UNDERSTANDING:

Firstly, this are some of features of asteroid dataset with the actual meaning of the parameters which we will be considering knowing the prediction.

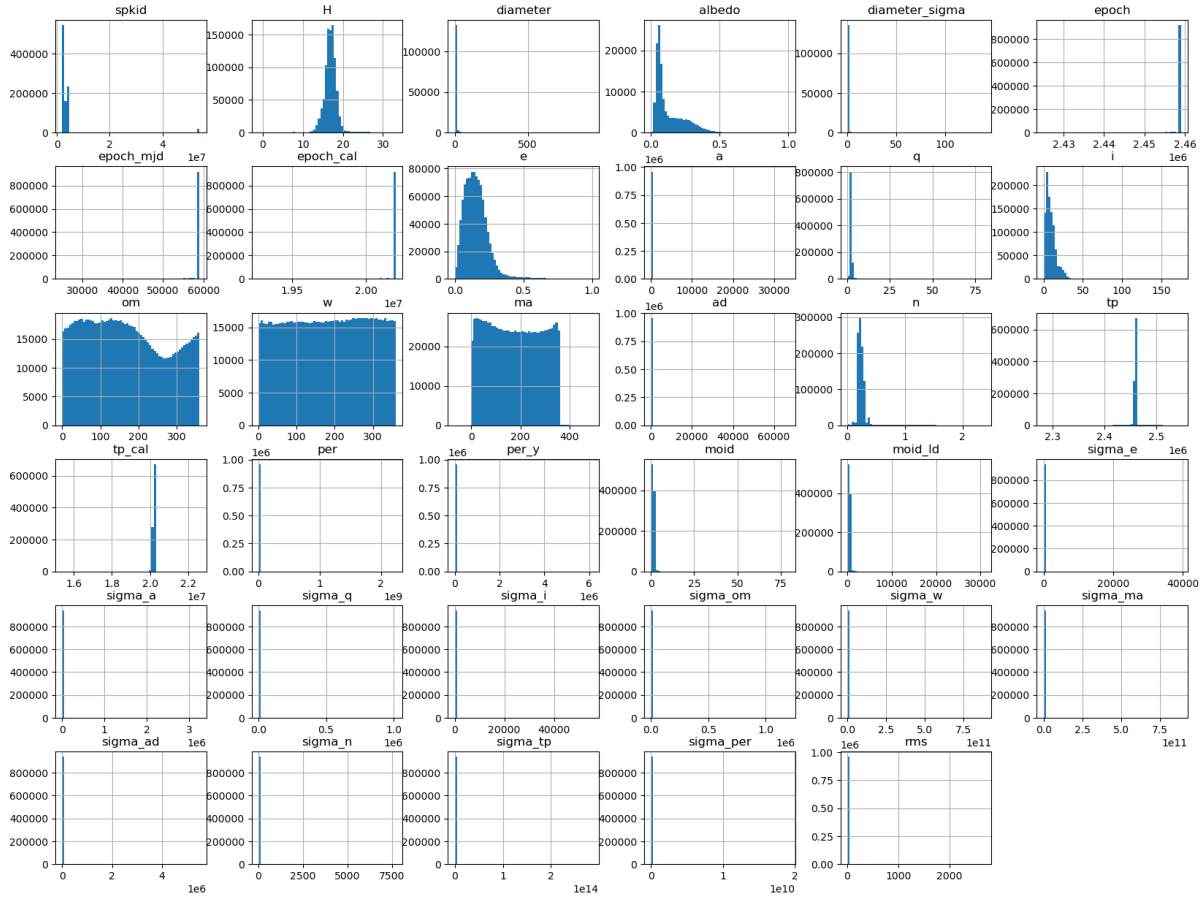
- **SPK-ID**: Object primary SPK-ID
- **Object ID**: Object internal database ID
- **fullname**: Object full name/designation
- **pdes**: Object primary designation
- **name**: Object IAU name
- **NEO**: Near-Earth Object (NEO) flag
- **PHA**: Potentially Hazardous Asteroid (PHA) flag
- **H**: Absolute magnitude parameter
- **Diameter**: object diameter (from equivalent sphere) km Unit
- **Albedo**: Geometric albedo
- **Diameter_sigma**: 1-sigma uncertainty in object diameter km Unit
- **Orbit_id**: Orbit solution ID
- **Epoch**: Epoch of osculation in modified Julian day form
- **Equinox**: Equinox of reference frame
- **e**: Eccentricity
- **a**: Semi-major axis au Unit
- **q**: perihelion distance au Unit
- **i**: inclination; angle with respect to x-y ecliptic plane
- **tp**: Time of perihelion passage TDB Unit
- **moid_ld**: Earth Minimum Orbit Intersection Distance au Unit

DATA PREPARATION:

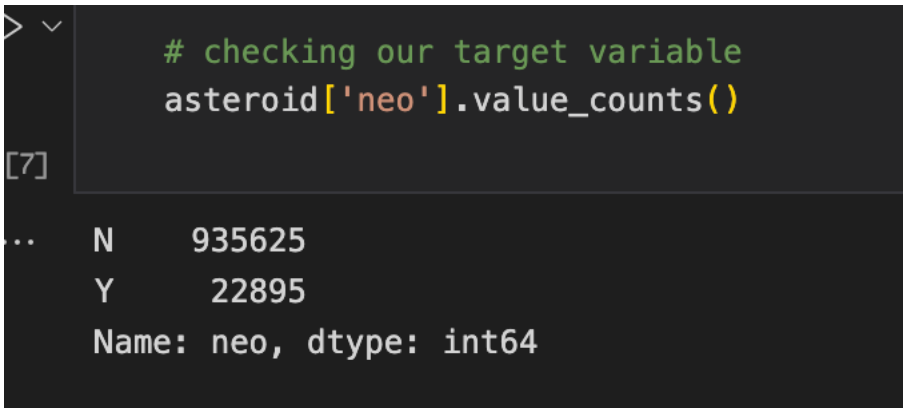
Visualizing the Data

In order to get a broad idea of the type of data we are manipulating, let's quickly review the data.

We can see most of the data points are right skewed.Hence, we can say that most of the numerical attribute's mean greater than median.



Defining our Target Variable



The column 'NEO' represents NEAR EARTH OBJECT and the values are label in YES and NO This is our target variable and our prediction is base on this two labels **N- 935625, Y-22895**.

Handling Caterogical Attributes

Many significant real-world features are categorical rather than numeric, despite the fact that the majority of machine learning algorithms only operate with numeric values. They take on levels or numbers when used as categorical features. These can be categorized in a number of ways, such as by age, state, or category of customer. As an alternative, these could be produced by binning underlying numerical characteristics, like categorizing people based on age ranges (e.g., 0–10, 11–18, 19–30, 30–50, etc.). Finally, these could be numerical identifiers where there is no significance in the connection between the values.. **(Engel, n.d.)**.
So , we have various encoding techniques in machine learning like one-hot encoding, label/ordinal encoding, Target encoding, Frequency Encoding, Binary encoding, Feature Hashing. This whole process is known as **feature encoding.** By applying these techniques it allow model to understand more efficiently. But in some cases of the machine learning, algorithm like decision tree directly takes the categorical value and educated about the labels. In other words, categorical data must be transformed into a numerical shape. The model's forecasts may also need to be transformed back into a categorical form if the categorical variable is an output variable so that they can be presented or applied. **(Brownlee, 2017).**
In our dataset, to transform the feature from categorical to numerical we have use two encoding techniques **one-hot encoder** and **Frequency Encoding.**

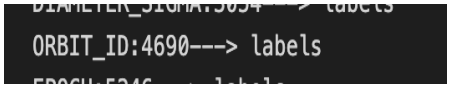
Firstly, why **one-hot encoder**, OHE is widely used for converting categorical features to numerical features. OHE transforms a single variable with n observations and d distinct values, to d binary variables with n observations each. Each observation indicates the presence 1 or absence 0 of dth binary variable. However, data becomes sparse after this transformation. **(TY - CHAP AU - Ul Haq, 2018).**
For, example if we use technique of pandas get_dummies that will create columns which will represent our labels class. And convert our categorical feature into numeric. Surprisingly, the challenged is what if there are more than 100 of labels?.

This method will make 100 of columns and divide our labels into them as 0 and 1. A data analyst will find it difficult to visualize and understand the data as a result. Additionally, these techniques will lengthen my runtime and storage. As Author believes that other than the accuracy, due to growing memory and storage consumption, compactness of machine learning models will become equally important in the future. **(Anderberg, 1973).**

So, we have applied the One Hot Encoder on **'PHA'** and **'CLASS'** to convert data of each row of a categorical column into numerical data.

The second encoding method we use is **Frequency Encoding**. This method is widely used by the professionals and in the Kaggle competition. Basically, this encoding is based on label of high cardinality. We may say that the frequency of categorical labels may be connected to target variable. In fact, it Helps the model to understand how to weigh data according to direct and inverse proportions, based on the type of data.

We have used this method for 'ORBIT_ID' column which have 4690 labels. In this case, if we apply OHE, as the cardinality is will higher the encoder will generate the enormous dispersed matrix which impact on the model. This method is further used in finding the unique labels of the column 'ORBIT_ID' and then replacing these values with the frequency of times they occurred in the column.



```
# Coverting the lables of ORBIT_ID into their respective frequency count and then storing it into a dictionary
asteroid_count = asteroid['ORBIT_ID'].value_counts().to_dict()
```

```
#Mapping the values of 'astroid_count' dictionary into the ORBIT_ID column
asteroid['ORBIT_ID'] = asteroid['ORBIT_ID'].map(asteroid_count)
```

```
#checking the changed values of the ORBIT_ID column
asteroid['ORBIT_ID']
```

```
0           45
1          569
2            8
3         1083
4            9
          ...
958519    19150
958520    18703
958521    12301
958522    29931
958523    29931
Name: ORBIT_ID, Length: 958520, dtype: int64
```

Hence, this method will not expand the attribute space and easily implemented.

In addition to using all of these methods, one can also create their own function and convert a categorical attribute into a numerical one. However, this method has limitations and may lengthen user work time if there are more than 10 columns. Additionally, they lack the abilities necessary to address this type of issue.

Perhaps in our case, our target variable is binary class with **Y AND N** .
Therefore we have made a function to change our categorical attribute into numerical.
Y = 1 AND N = 0
Which means if **1** the asteroid in the cosmos is **near the earth**, if **0** the asteroid is **away from the earth**.

```
# altering the Prediction from object to float by making a function

def alter(column):
    if column == 'Y':
        return 0
    else:
        return 1
```

```
# applying the function on the PREDICTION column
asteroid['PREDICTION'] = asteroid['PREDICTION'].apply(alter)
```

DATA CLEANING

Data preparation and cleaning take up the majority time of a data scientist's work. The data cleaning process takes up 80% of time, which is important for the algorithm. Understanding data will improve the accuracy of data results, according to a proverb from long ago.
We have a huge number of unique IDs in our dataset, and dropping any of them will not have an effect on our model.
Therefore, attributes like 'EQUINOX' , 'PDES', 'ID', 'PREFIX','FULL_NAME', 'NAME' had been dropped for future analysis. By deleting the Nominal attributes doesn't affect the model score.
But, by keeping such attributes may impact model run time.

Handling missing values in categorical attribute

The majority of the time, the data in a real-world contains duplicate or missing numbers. This could be the result of human error or a failing mechanism used to gather the data. Additionally, the majority of machine learning models fail when features are absent. In reality, if we attempt data analysis, it would be biased or inaccurate.
Perhaps there many ancient technique to deal with problem, one which is deleting the instance or filling the missing values with the estimated or high frequent value.

Unfortunately, in our categorical attribute **"PHA**" we have **19921** Nan values and two labels, **Y and N.**

```
#Checking the NA values in the PHA column
asteroid['PHA'].isna().sum()
```

```
19921
```

```
#Checking the unique values in the PHA column
asteroid['PHA'].unique()
```

```
array(['N', 'Y', nan], dtype=object)
```

We could discard the data in this circumstance, but we risk losing crucial information. The alternative might be to fill in the NA values with the values that show up most frequently, but this won't help us because the data is unbalanced.
To solve this issue, we create a new label called 'missing' and replace it with all the **NA** values in the **PHA** column. By using **simple imputer** from Scikit library.

Applying SimpleImputer

```
# Replacing the misssing values using SimpleImputer

imputer = SimpleImputer(missing_values= np.nan, strategy= 'constant', fill_value= 'missing')
asteroid['PHA']= imputer.fit_transform(asteroid[['PHA']])
```

```
#After applying SimpleImputer we have 3 labels
asteroid['PHA'].unique()
```

```
array(['N', 'Y', 'missing'], dtype=object)
```

Applying OneHotEncoder

```
# now applying onehotencoder using sklearn
# sparse matrix length is ambiguous; use getnnz() or shape[0]
ohe  = OneHotEncoder(sparse= False)
asteroid['PHA'] = ohe.fit_transform(asteroid[['PHA']])
```

Simple methods for filling in missing values using a univariate imputer. Use a descriptive statistic (such as mean, median, or most common) or a constant value to fill in any missing values along each column. **(Pedregosa et al., 2011)**.Subsequently, we have apply OHE and transform our categorical attribute into numerical attribute.

In the future application we could apply classifier algorithm to predict our missing values or use unsupervised machine learning methods like clustering to cluster the similarity of the missing values. Also, there was survey which techniques would be appropriate on handling missing values was done and to handle missing values, a number of conventional statistical and machine learning imputation methods, including mean, regression, K nearest neighbor, ensemble based, etc., have been suggested in the literature. **(Emmanuel, 2021)**

Thirdly, we also had **Nan values** in our **target variable**, which were quite low.
Hence, we have directly drop those values.

```
[8]  asteroid['neo'].unique()

..   array(['N', 'Y', nan], dtype=object)

[9]  asteroid['neo'].isnull().sum()

..   4

[10] # null values in target variable
     # asteroid['neo'] = asteroid['neo'].isnull().dropna(axis=0)
     asteroid = asteroid.dropna(subset=['neo'])

[11] asteroid['neo'].unique()

..   array(['N', 'Y'], dtype=object)
```

Handling missing values in numerical attribute

The below image interprets the null values of different attributes in the our dataset.

```
 1  SPKID                  0
 2  PREDICTION             0
 3  PHA                    0
 4  H                   6263
 5  DIAMETER          822311
 6  ALBEDO            823417
 7  DIAMETER_SIGMA    822439
 8  ORBIT_ID               0
 9  EPOCH                  0
10  EPOCH_MJD              0
11  EPOCH_CAL              0
12  E                      0
13  A                      0
14  Q                      0
15  I                      0
16  OM                     0
17  W                      0
18  MA                     1
19  AD                     0
20  N                      0
21  TP                     0
22  TP_CAL                 0
23  PER                    0
24  PER_Y                  0
25  MOID               19921
26  MOID_LD              127
27  SIGMA_E            19922
28  SIGMA_A            19922
29  SIGMA_Q            19922
30  SIGMA_I            19922
31  SIGMA_OM           19922
32  SIGMA_W            19922
33  SIGMA_MA           19922
34  SIGMA_AD           19922
35  SIGMA_N            19922
36  SIGMA_TP           19922
37  SIGMA_PER          19922
38  CLASS                  0
39  RMS                    2
40  dtype: int64
```

By the analysing the dataset, the three columns ‘DIAMETER’, ‘ ALBEDO’ ,’DIAMETER_SIGMA’ contains missing values more than 95%. Therefore, we are dropping this attributes.
Still this may impact our prediction model. To solve the problem one we can ask the organization to put the observation again and fill the data another way using different methods like MAR or multiple imputation to handle missing values.

Secondly, for other numeric features we implemented Simple Imputer from scikit learn to handle the missing values. We must employ a median strategy because, as we are aware, means are not robust against outliers.

```
# using Simpleimputer we can fill the missing values with median in each numerical column
imputer2  = SimpleImputer(strategy='median')
imputer2.fit(asteroid)
```

```
SimpleImputer(strategy='median')
```

Finally, here is our dataset after it has been thoroughly cleaned, with no missing values and all categorical features converted to numerical.

```
# Now we notice that the data is clean for further implementation
final_asteroid.isnull().sum()
```

```
Output exceeds the size limit. Open the full output data in a text editor
SPKID         0
PREDICTION    0
PHA           0
H             0
ORBIT_ID      0
EPOCH         0
EPOCH_MJD     0
EPOCH_CAL     0
E             0
A             0
Q             0
I             0
OM            0
W             0
MA            0
AD            0
N             0
TP            0
TP_CAL        0
PER           0
PER_Y         0
MOID          0
MOID_LD       0
SIGMA_E       0
SIGMA_A       0
...
SIGMA_TP      0
SIGMA_PER     0
CLASS         0
RMS           0
dtype: int64
```

```
asteroid.info()
```

```
[43]
...  Output exceeds the size limit. Open the full o
     <class 'pandas.core.frame.DataFrame'>
     Int64Index: 958520 entries, 0 to 958523
     Data columns (total 36 columns):
      #   Column       Non-Null Count   Dtype
     ---  ------       --------------   -----
      0   SPKID        958520 non-null  int64
      1   PREDICTION   958520 non-null  int64
      2   PHA          958520 non-null  float64
      3   H            952257 non-null  float64
      4   ORBIT_ID     958520 non-null  int64
      5   EPOCH        958520 non-null  float64
      6   EPOCH_MJD    958520 non-null  int64
      7   EPOCH_CAL    958520 non-null  float64
      8   E            958520 non-null  float64
      9   A            958520 non-null  float64
      10  Q            958520 non-null  float64
      11  I            958520 non-null  float64
      12  OM           958520 non-null  float64
      13  W            958520 non-null  float64
      14  MA           958519 non-null  float64
      15  AD           958520 non-null  float64
      16  N            958520 non-null  float64
      17  TP           958520 non-null  float64
      18  TP_CAL       958520 non-null  float64
      19  PER          958520 non-null  float64
     ...
      34  CLASS        958520 non-null  float64
      35  RMS          958518 non-null  float64
     dtypes: float64(32), int64(4)
     memory usage: 302.8 MB
```

CREATING A TRAIN AND TEST SET

So, as we predicting that asteroid Is closer to the planet or not we are making **"NEO**" name changed to **"PREDICTION**" and separating our X and Y variable in the dataset.
X are the features which are **independent** and **Y** is the **dependent** variable in dataset.
Y is the forecast in YES and NO which we have change to 1 and 0 respectively.

Secondly, we are creating a train set and test set with the split of 70% and 30% respectively. The process entails splitting the information into two subsets. The training dataset is the first subset, which is used to fit the algorithm. The model is not trained using the second subset; rather, it is given the input element of the dataset, and its predictions are then made and contrasted with the anticipated values. The test dataset is the second dataset in question. The splitting varies from dataset to dataset.

```
# splilting the dataset into train and test sets to avoid overfitting
X_train, X_test , Y_train, Y_test = train_test_split(X,Y, test_size= 0.30,random_state=42 )
```

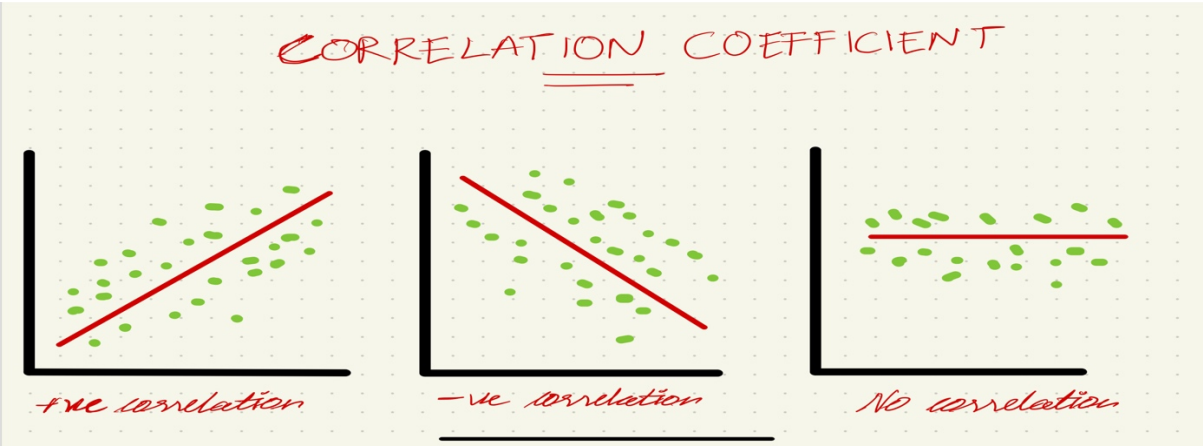
+ Code + Markdown

```
# checking the shape of our train and test dataset
print(f'TRAIN----> {X_train.shape},{Y_train.shape} 70% \nTEST----> {X_test.shape},{Y_test.shape} 30% ')
```

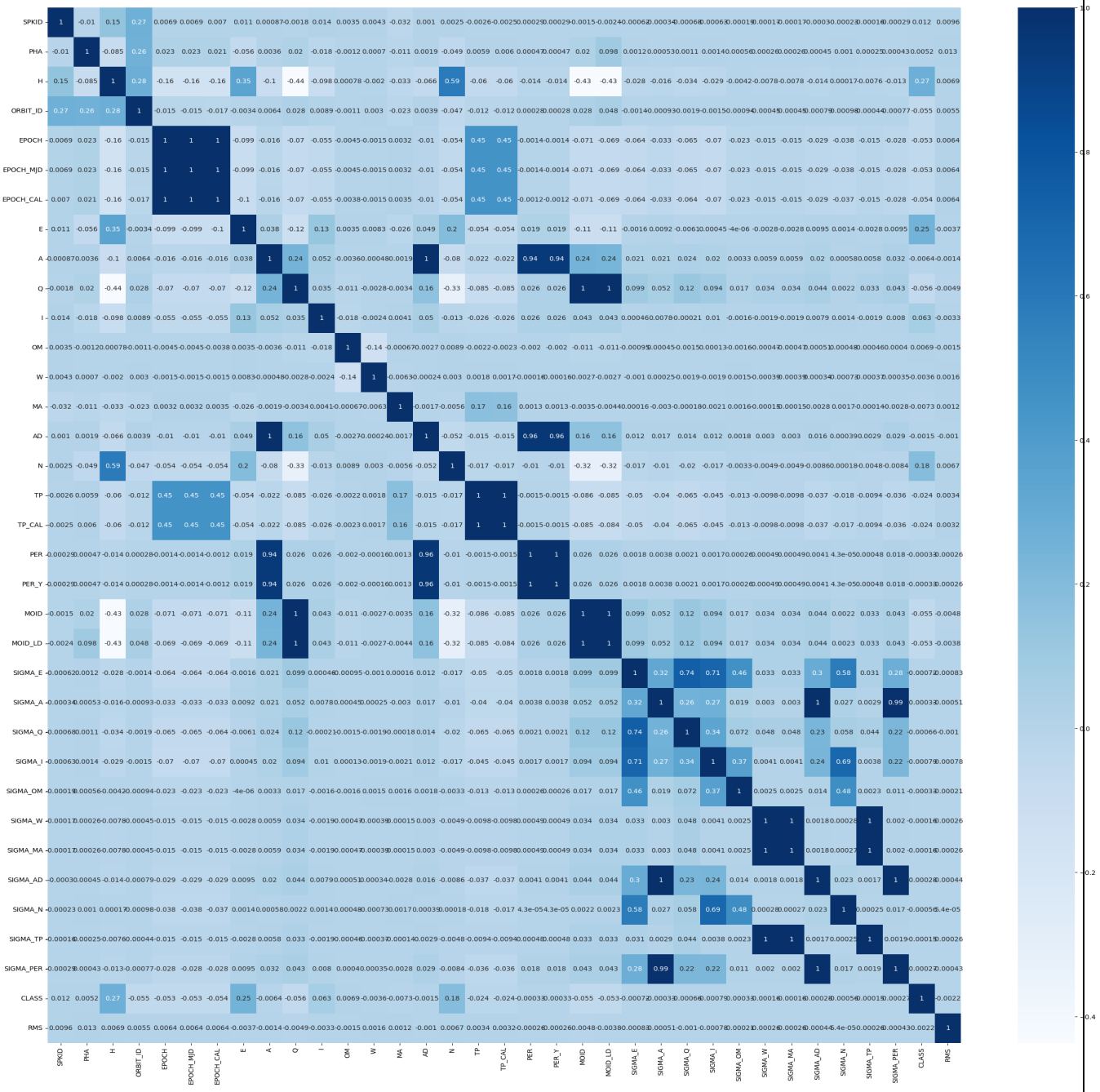
```
TRAIN----> (670964, 35),(670964,) 70%
TEST----> (287556, 35),(287556,) 30%
```

FEATURE ENGINEERING

Firstly, we will only work on independent features in our training set to avoid overfitting. We will look for correlation among the feature. The most popular method for determining a linear connection is the **Pearson correlation coefficient (r).** The strength and direction of the connection between two variables is expressed as a number between -1 and 1. **(Turney, 2022).** Below chart interpret the different types of correlation in **Pearson correlation**.



By using **seaborn heatmap** we have plot the correlation of the **35 numeric columns**



Manual correlation analysis would be difficult for a user to perform. What if the information contains hundreds of columns?

Therefore, it would be simple for us to create a function that compares all the features to one another and eliminates the first feature that is highly correlated with any other feature.

```
# with the following function we can select highly correlated features
# it will remove the first feature that is correlated with anything other feature
# abs method converts -ve correlation into +ve one
# Note: we should avoid removing -ve correlated as it would be helpful in traning models for better prediction

def correlation(dataset, threshold):
    col_corr = set() # set of all the names of correlared columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            # if (corr_matrix.iloc[i,j])> threshold: # we are interesred in +ve coeff value
            if  abs(corr_matrix.iloc[i,j]) > threshold:
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr
```

[56]

```
#Assigning 0.8 threshold value to get highly positive and negative correlation
corr_features = correlation(X_train,0.8)
len(set(corr_features)) #Displaying the number of highly correlated columns
```

[57]

```
12
```

This method iterates through the features, comparing each one against the others one at a time, and removing any that are **significantly positive or negative in comparison** to the others. We can also determine the level of connection between each other.
For example, the threshold for our feature selection **is 0.8** positive and negative correlation. As a result, we will get both a highly positive and a highly negative coefficient using the **absolute function**.

Hence, the function **has removed 12 features** that are highly correlated.

```
#Assigning 0.8 threshold value to get highly positive and negative correlation
corr_features = correlation(X_train,0.8)
len(set(corr_features)) #Displaying the number of highly correlated columns
```

[57]

```
12
```

```
# these columns are highly correlated with other columns
corr_features
```

[58]

```
{'AD',
 'EPOCH_CAL',
 'EPOCH_MJD',
 'MOID',
 'MOID_LD',
 'PER',
 'PER_Y',
 'SIGMA_AD',
 'SIGMA_MA',
 'SIGMA_PER',
 'SIGMA_TP',
 'TP_CAL'}
```

```
# droping these columns
X_train = X_train.drop(corr_features,axis=1)
X_test = X_test.drop(corr_features,axis=1) # Dropping these highly correlated columns from the test dataset as well
```

[59]

Finally, we have removed the features from the independent train dataset and the Attributes from the independent test dataset, respectively (X train and X test).

Our dataset in almost ready for **model evaluation and implementation.**
After all data cleaning and feature engineering, we have reduced **the 44 features to just 23.**

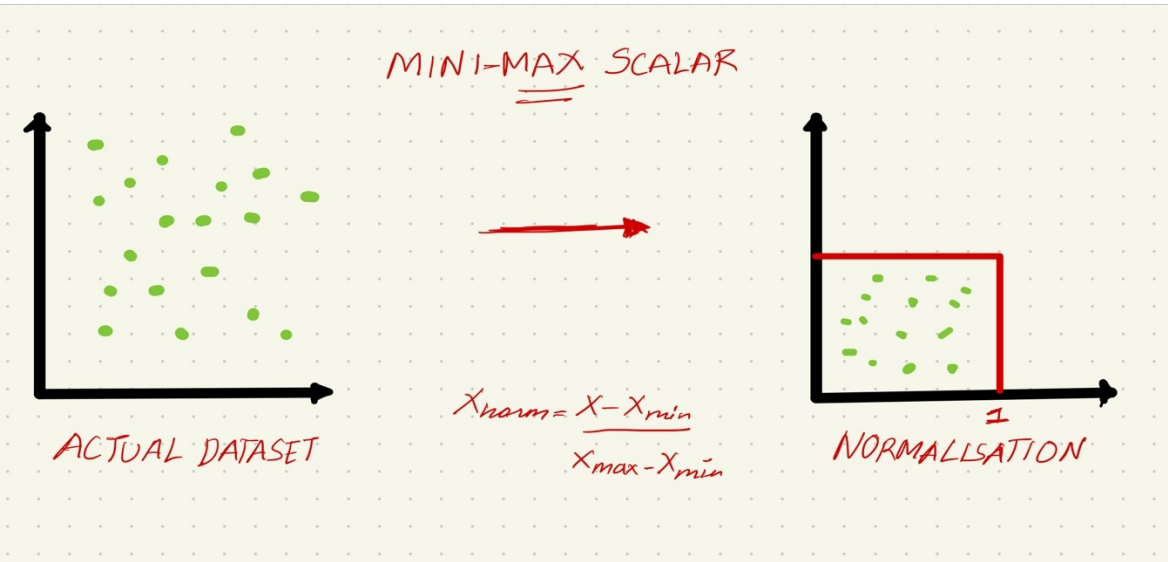
```
#Notice the remaining number of columns in the train and test dataset
X_train.shape, X_test.shape
```

```
((670964, 23), (287556, 23))
```

MODELING:

FEATURE SCALING

You must scale your features in your data, which is one of the most crucial changes. In most cases, when the input numerical attributes have very diverse scales, machine learning algorithms don't work well. **(Géron, 2019)**
Take a business dataset as an example, and compare the salary feature with the rating, which would be expressed in thousands and 1 to 10, respectively. Due to the various scales of the two features, it is possible that features with larger magnitudes will be given more weight. The machine learning algorithm's performance will be impacted; clearly, we are not interested in our algorithm to be biased towards one trait.
We have used **Min-max scaling.** (normalization). It scales between 0 and 1.

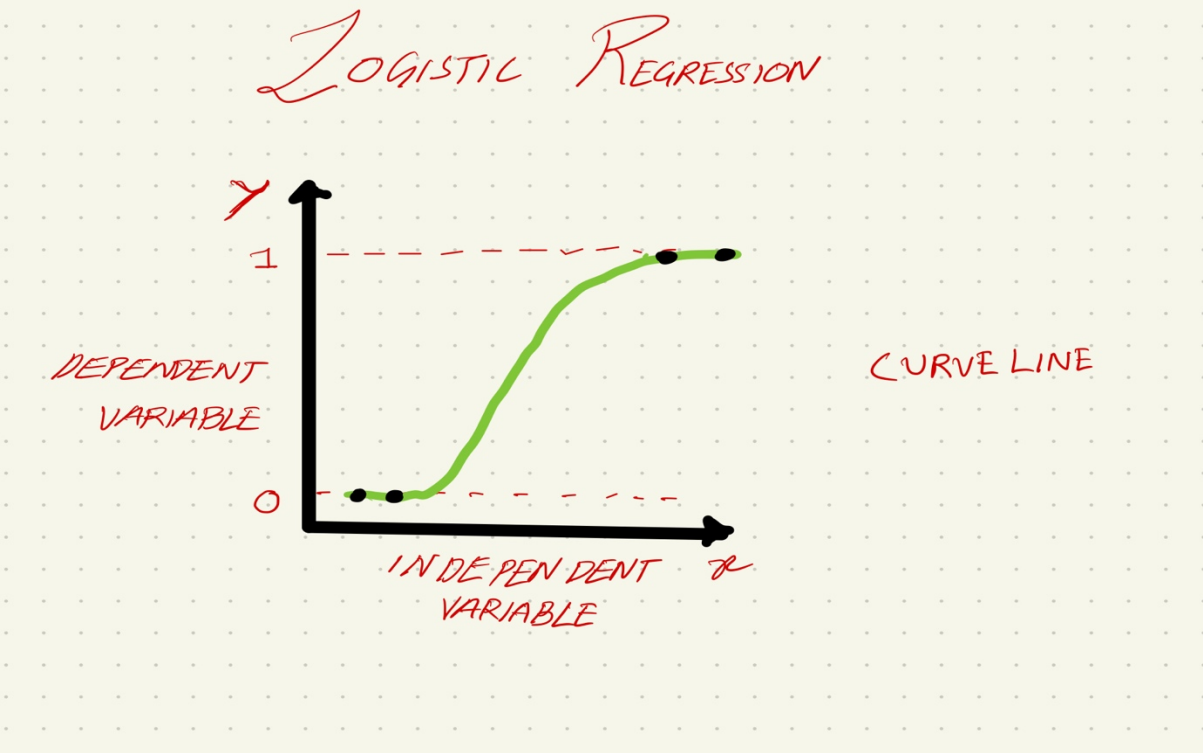


MODEL EVALUATION WITHOUT OPTIMISATION

The process of selecting a model to get the solution for the problem is known as model selection. Model selection for machine learning is an important aspect for predictions.

The term **"classifier**" also applies to **Logistic Regression**. When the goal variable is categorical, this model is employed. If the 'multi-class' option is set to 'ovr' in the multiclass case, the training algorithm employs the one-vs-rest (OvR) scheme; otherwise, it uses the cross-entropy loss. (At this time, only the solvers "lbfgs," "sag," "saga," and "newton-cg" allow the "multinomial" option.) **(Pedregosa et al., 2011)**

For instance, in our situation, we must forecast whether the asteroid will approach or flee from the planet. Every situation must be categorized as **YES** or **NO**.



For classification problems, cyber security, and image processing, this model is used in a variety of corporate domains.

```
[66]  log = LogisticRegression()
      ✓ 0.0s

[67]  #fit the model on the training data
      log.fit(X_train, Y_train)
      ✓ 2.7s

...   /Users/raw/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: 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
        n_iter_i = _check_optimize_result(

      LogisticRegression()

[68]  #Using trained model to predict for test set
      y_pred = log.predict(X_test)
      ✓ 0.0s

[69]  #Accuracy
      log.score(X_test, Y_test) * 100
      ✓ 0.0s

...   99.9777434656206

[70]  #Comparing the score of the trained data with test data
      log.score(X_train,Y_train)*100
      ✓ 0.1s

...   99.98211528487371
```

We have adjusted the model to our train split, and it will now begin training on the train set. We will only adjust the model on the train set and watch how it forecasts on the test set.

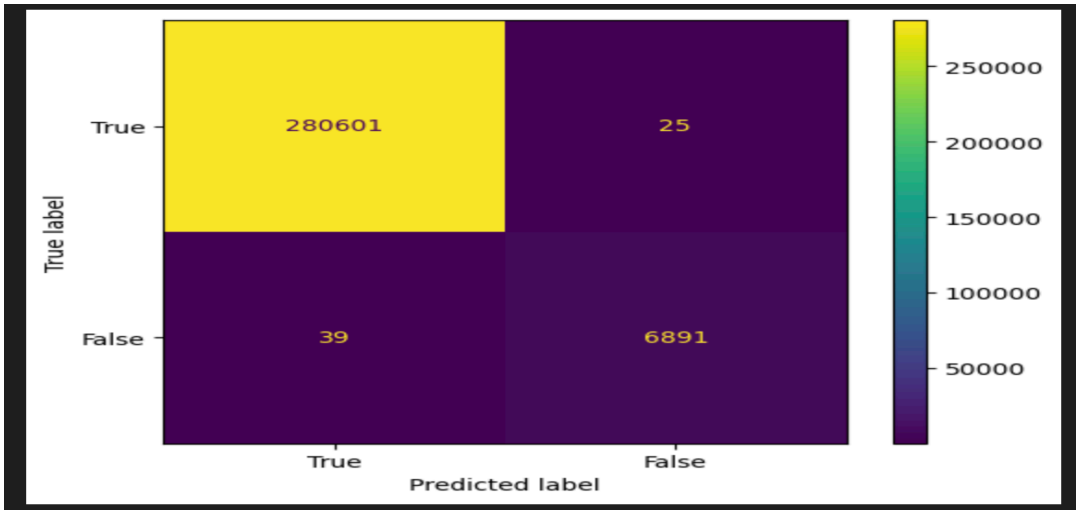
ACCURACY

For this model we achieved a prediction of **99.97%** with the train data and have achieved **99.98%** with the test data.

Confusion Matrix: It is a table layout which is used for getting different outcomes of the prediction and Output for classification problem. The table is divided into 4 parts (**True-Negative, False-Negative, True-Positive, False-Positive**) each part has its own significant value.

True-Positive (TP): The number of positive outcomes predicted as Positive.
True-Negative (TN): The number of negative outcomes predicted as negative.
False-Positive (FP): The number of negative outcomes predicted as positive.
False-Negative (FN): The number of positive outcomes predicted as negative.

For this Model we have achieved a score of **TP-6891, TN-280601, FP-39, FN-25.**



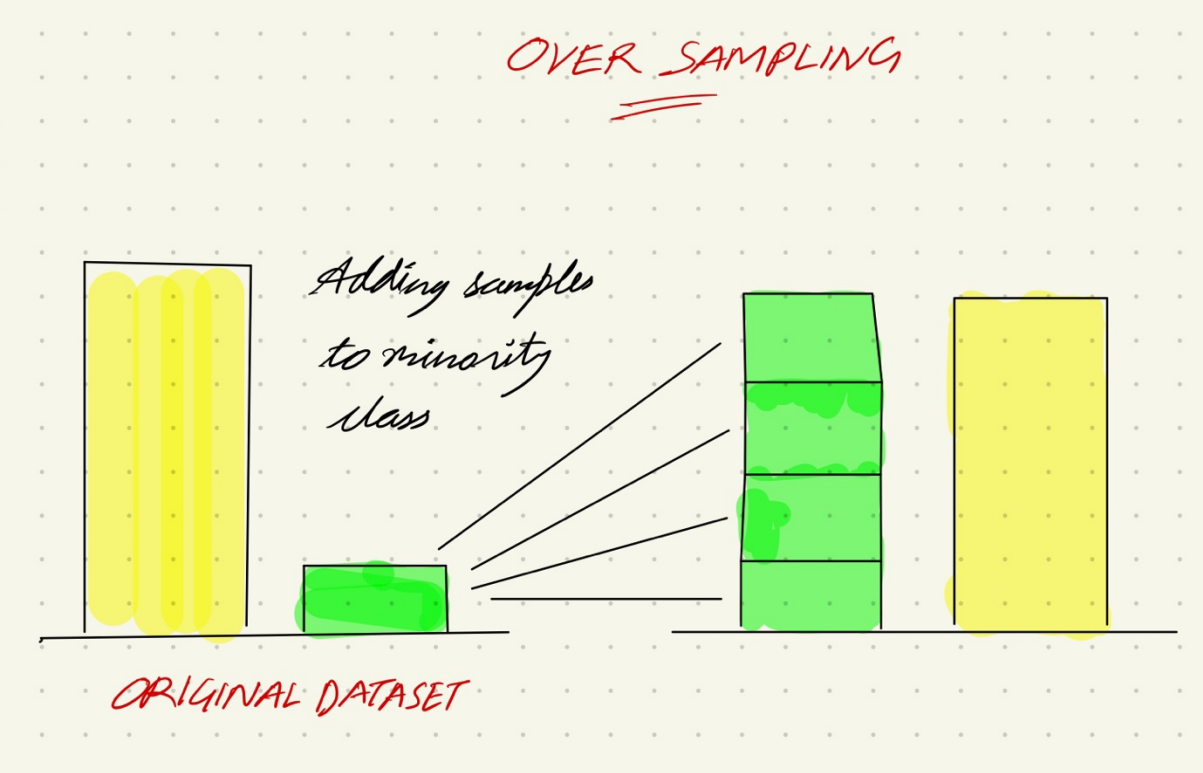
Classification report: It provides us with multiple attributes such as accuracy, micro average, weighted average, precision, recall, F1 score and support all in single table form.

```
print(classification_report(Y_test , y_pred))
```

✓ 0.5s

```
              precision    recall  f1-score   support

         0.0       1.00      0.99      1.00      6930
         1.0       1.00      1.00      1.00    280626

    accuracy                           1.00    287556
   macro avg       1.00      1.00      1.00    287556
weighted avg       1.00      1.00      1.00    287556
```

OPTIMIZATION USING OVERSAMPLING METHOD FOR IMBALANCE DATA



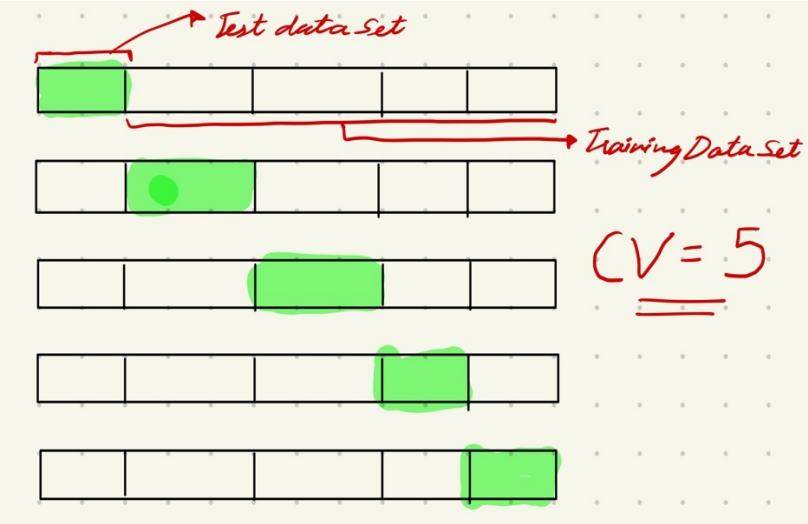
In the past, sampling techniques like the synthetic minority oversampling technique **(SMOTE)** have been applied to such issues in order to artificially balance the data collection prior to classifier training.
Other real-world classification domains, from those found in civilian applications like medical diagnosis to those found in military applications like surveillance, also exhibit imbalance in class distribution. This is not only true for fault detection. **(. Mathew, 2018).** Reducing the misclassification of the minority class instances is crucial in most of these real-world problems

We have used **SGD classifier (STOCHASTIC GRADIENT DESCENT).**
SGD classifier performs well on these issues rather than logistic regression when the code is sluggish, and the dataset is large.
if (**loss = log)** is basically equivalent to logistic regression.
Though, we have set the learning rate to **'optimal'**, still it will learn on **0.001 and 1.**
The initial learning rate is also 0.001, 1.
A multiplier for the regularization word. Stronger regularization results from larger values. When learning rate is set to "ideal," it is also used to calculate learning rate. **(Pedregosa et al., 2011).** All of this is considered hyperparameter tuning. Because it would take a lot of time to manually identify each parameter, we have used the **GridSearchCV** method to find the best parameter.

Cross-validation

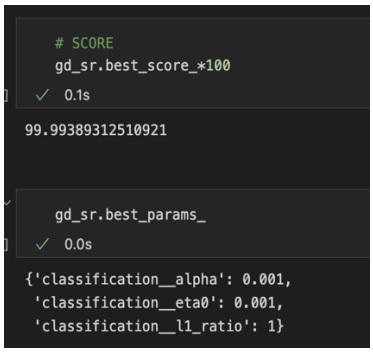
A methodological error is learning the parameters of a prediction function and evaluating it on the same set of data. A model that simply repeats the labels of the samples it has just seen would score perfectly but be unable to make any predictions about data that has not yet been seen**. Overfitting** is the term for this circumstance. It is customary to reserve a portion of the available data as a test set (X test, y test) when conducting a (supervised) machine learning experiment in order to avoid this problem. It is possible to choose the ideal values by **GRIDSEARCHCV. (Pedregosa et al., 2011)**

Hence CV = 5



Scoring is set to **recall** for binary classifier.

Due to the size of our dataset, we put it to default for **iteration**. What would happen if we increased the iteration? **The higher the iteration, the slower the training** would be. Therefore, in order to save time and have it search for the optimal parameters, we set it to default.
The train set data will now be fitted into the model, and the model will search for the finest parameter that would be most beneficial for our learning.

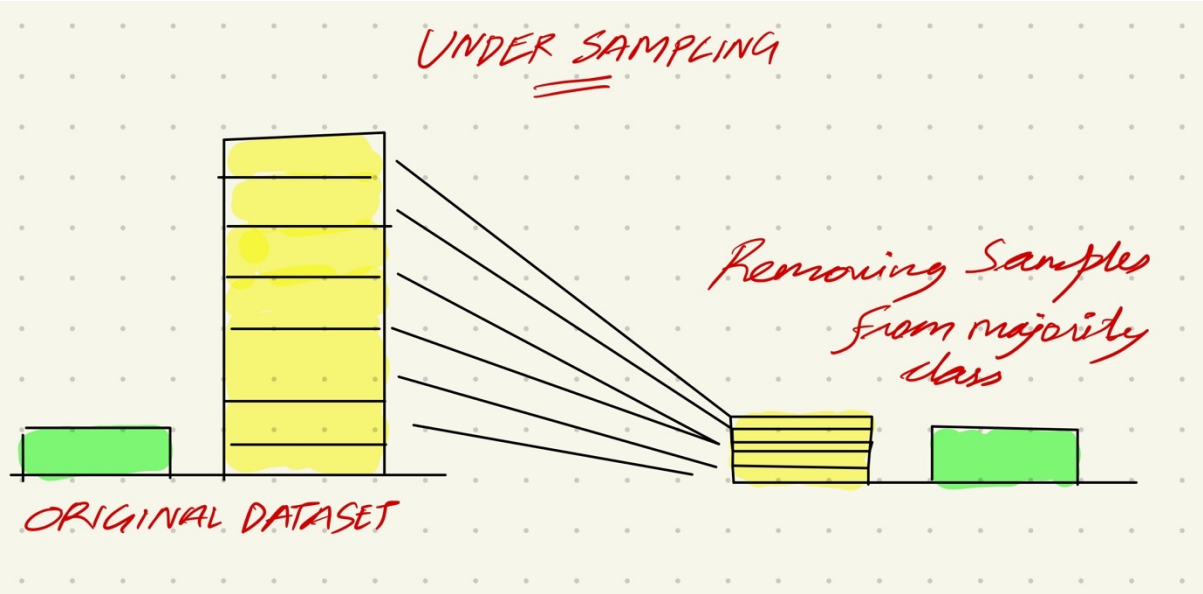


Here are our best parameters for **SGD classifier** to predict our Target variable.

ACCURACY

99.99%

OPTIMISATION USING UNDERSAMPLING METHOD FOR IMBALANCE DATA



When there is sufficient data for a thorough analysis**, Undersampling** is suitable. To establish two classes of equal size, the data scientist utilizes all the rare events while reducing the number of abundant events.
Since the majority class is more essential than the minority class, many machines learning algorithms, including decision trees, k-near neighbors, and neural networks, will learn this and focus more on and perform better on the majority class.

We have used **Near-miss** algorithm for **Undersampling** the data.

In our case it will work like,

Let **NO** be the samples belonging to the targeted class to be under-sampled. **YES** refers to the samples from the minority class (i.e., the most under-represented class).
Near-Miss selects the **NO** for which the average distance to the N closest samples of the **YES** is the smallest. **(developers., 2014)**

```
from imblearn.under_sampling import NearMiss
```
✓ 0.0s

```
# implementing undersampling for imbalance data
nm = NearMiss()
#Fitting X and Y in the model
X_near, Y_near = nm.fit_resample(X,Y)
X_near.shape, Y_near.shape
```
✓ 5m 15.4s

```
((45790, 35), (45790,))
```

```
# Reapplying train and test split on the balanced data

X_train,X_test, Y_train, Y_test = train_test_split(X_near,Y_near, test_size=0.20, random_state=42)
```
✓ 0.1s

```
X_train = X_train.drop(corr_features,axis=1)
X_test = X_test.drop(corr_features,axis=1)
```
✓ 0.0s

HYPERPARAMTER TUNING

As you can see, there are a lot of stages involved in data transformation that must be carried out correctly. Fortunately, Scikit-Learn offers the **Pipeline** class to assist with such transformational processes.

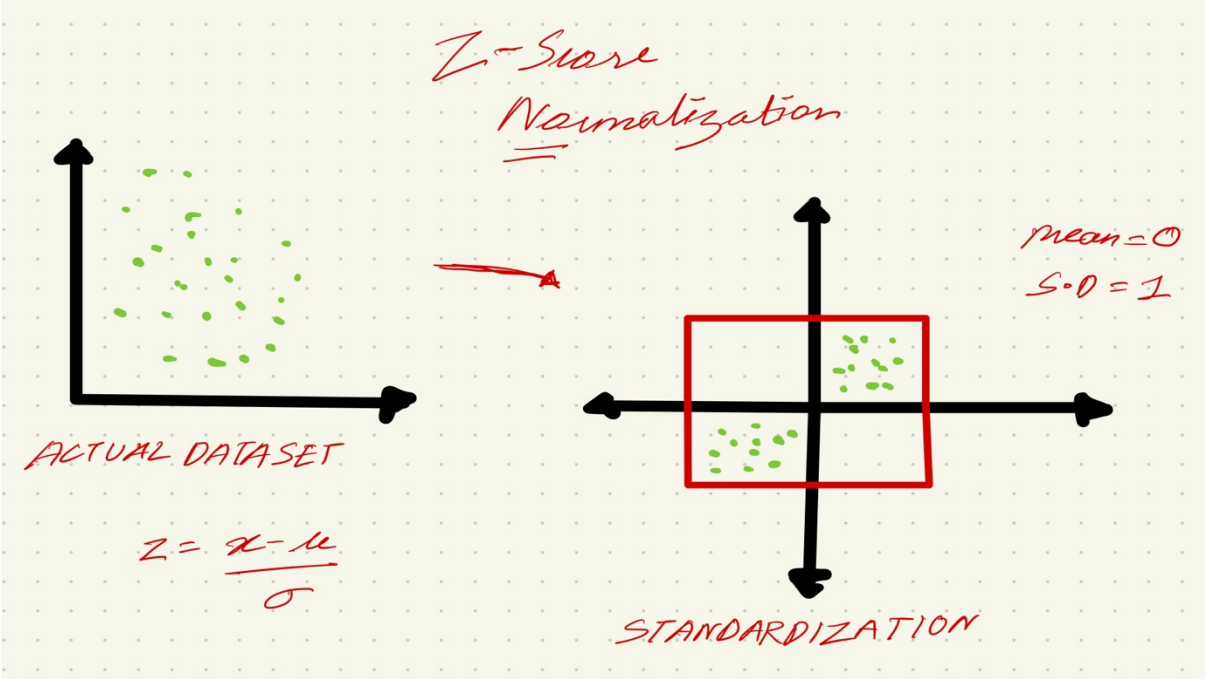
```
model1 = Pipeline([('scaling',StandardScaler()),('model',LogisticRegression(max_iter=1000))])


param_grid = {'model__penalty':["l1","l2"],
              'model__solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
               'model__max_iter':[100,1000]
                 }


clf = GridSearchCV(estimator=model1,param_grid=param_grid,cv=2)
```
[89] ✓ 0.0s

```
clf.fit(X_train,Y_train)
```
[90] ✓ 23.1s

The collection of name/estimator pairs that define the steps in a pipeline are provided to the Pipeline constructor. Every estimator except the final one needs to be a transformer (i.e., have a fit transform() function). You can choose any name you want, if it is original and doesn't contain a double underscore ("__"). The names will be used afterward for hyperparameter tuning.
When you use the pipeline's fit() method, fit transform() is called successively on each transformer, passing the results of each call as a parameter to the subsequent call, until the final estimator, for which it simply uses fit(). **(Géron, 2019)**

we are scaling using **Z-Score Normalization (STANDARDIZATON).**

Standardization is based on Normal distribution, where **mean is zero** and **standard deviation is one.**



Here, we have cross- validation in GridsearchCV as **2.**
Additionally, we are adjusting to the train set for training and determining the best logistic regression parameters.

```
# SCORE
clf.best_score_*100
```

```
100.0
```

```
clf.best_params_
```

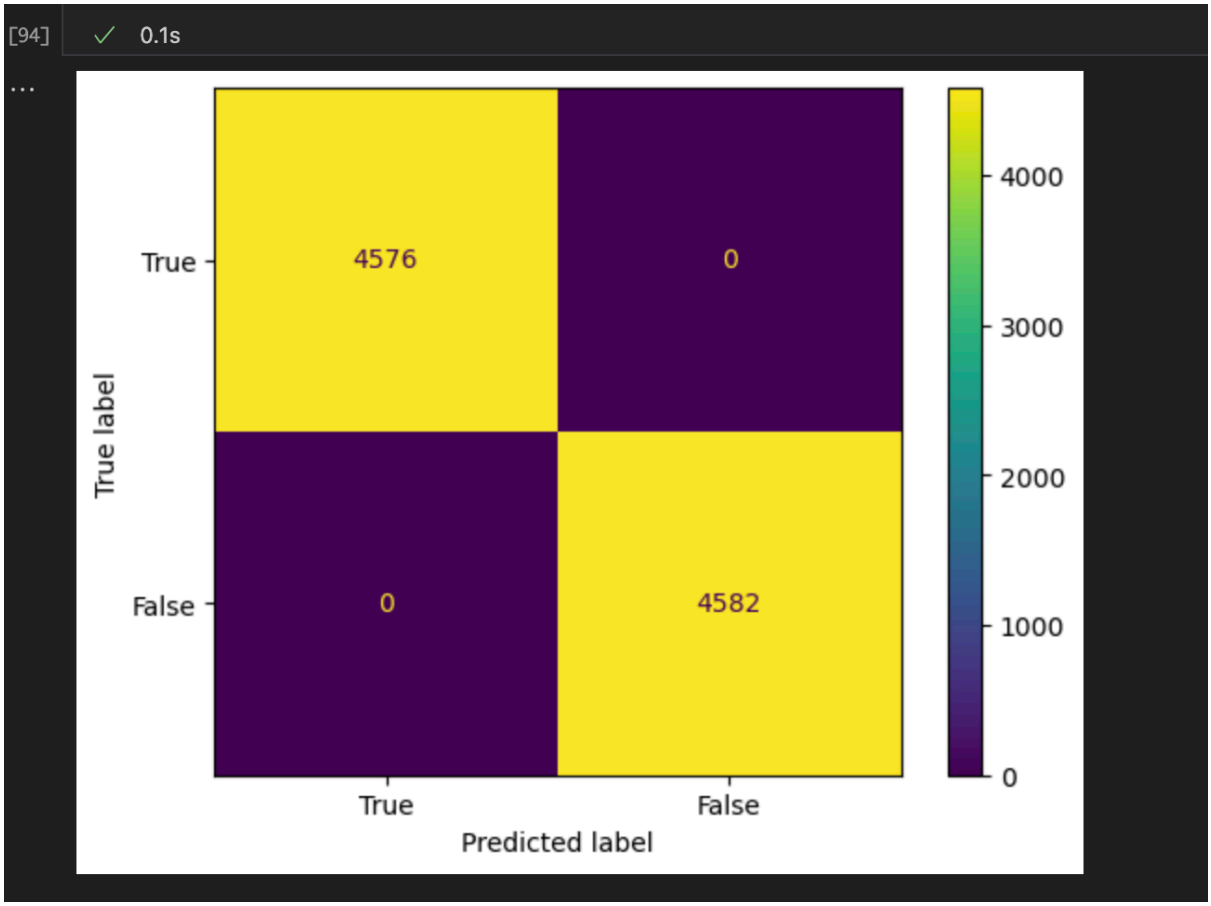
```
{'model__max_iter': 1000, 'model__penalty': 'l1', 'model__solver': 'liblinear'}
```

This are best parameters for our logistic model using undersampling method.
Where the solver should be **"liblinear,"** the penalty **"l1,"** and the iteration be **1000**.

ACCURACY

100%

Confusion Matrix: As we can see the confusion matrix for our model has been improved. With



EVALUATION:

So, here are comparing all the three techniques and their accuracy.

TECHNIQUE	MODEL	ACCURACY
IMBALANCE DATA	LOGISTIC REGRESSION	99%
OVER SAMPLING	SGD CLASSIFIER (WITH OPTIMISATION)	99%
UNDER SAMPLING	LOGISTIC REGRESSION (WITH OPTIMISATION)	100%

CONCULSION:

As a result, we used two techniques—one with optimization and the other without—with one model and one classification.
In the initial learning, imbalance dataset was used to train logistic regression with default settings, and we obtained an accuracy of **99.993%.**
As part of the second learning, we used an SGD classifier with hyperparameter tuning and the oversampling technique to balance the data, and as a result, we saw a slight improvement in accuracy, reaching **99.998%.**
In the third lesson, we used undersampling to balance the data and logistic regression with optimization to achieve a **100% accuracy.**

Reference

Asteroid Dataset [WWW Document], n.d. URL https://www.kaggle.com/datasets/sakhawat18/asteroid-dataset (accessed 3.3.23).

Hefele, J.D., Bortolussi, F., Zwart, S.P., 2020. Identifying Earth-impacting asteroids using an artificial neural network. Astron. Astrophys. 634, A45. https://doi.org/10.1051/0004-6361/201935983

Hossain, M.S., Zabed, Md.A., 2023. Machine Learning Approaches for Classification and Diameter Prediction of Asteroids, in: Ahmad, M., Uddin, M.S., Jang, Y.M. (Eds.), Proceedings of International Conference on Information and Communication Technology for Development, Studies in Autonomic, Data-Driven and Industrial Computing. Springer Nature, Singapore, pp. 43–55. https://doi.org/10.1007/978-981-19-7528-8_4

In Depth | Asteroids [WWW Document], n.d. . NASA Sol. Syst. Explor. URL https://solarsystem.nasa.gov/asteroids-comets-and-meteors/asteroids/in-depth (accessed 3.3.23).

Malakouti, S. matin, 2023. Ml Techniques: Classifying Dangerous Asteroids. https://doi.org/10.2139/ssrn.4328181

Engel, A., n.d. towardsdatascience.com. [Online]
Available at: https://towardsdatascience.com/categorical-variables-for-machine-learning-algorithms-d2768d587ab6

Brownlee, J., 2017. Why One-Hot Encode Data in Machine Learning?. [Online]
Available at: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

TY - CHAP AU - Ul Haq, I. A. -. G. I. A. -. V. P. A. -. B. S. P. -. 2. S. -. 6. E. -. 8. S. -. 9.-9.-1.-6.-4. T. -. C. F. T. w. C. O.-H. E. f. F. D. i. D., 2018. Categorical Features Transformation with Compact One-Hot Encoder for Fraud Detection in Distributed Environment: 16th Australasian Conference, AusDM 2018, Bahrurst, NSW, Australia, November 28–30, 2018, Revised Selected Papers.

Anderberg, M., 1973. Investigating Road Safety Management Systems in the European Countries: Patterns and Particularities. https://www.scirp.org/(S(351jmbntvnsjt1aadkposzje))/journal/home.aspx?journalid=357.

Pargent, F., 2019. A Benchmark Experiment on How to Encode Categorical Features in Predictive Modeling. Master Thesis in Statistics Ludwig-Maximilians-Universität München .

Emmanuel, T. M. T. M. D. e. a., 2021. A survey on missing data in machine learning. https://doi.org/10.1186/s40537-021-00516-9.

Pedregosa et al., J. 1., 2011. Scikit-learn: Machine Learning in Python. [Online]
Available at: https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html

Turney, S., 2022. Pearson Correlation Coefficient (r) | Guide & Examples. [Online]
Available at: https://www.scribbr.com/statistics/pearson-correlation-coefficient/

Géron, A., 2019. O'REILLY® Hands-on Machine Learning with Scikit-Learn, Keras &TensorFlow. fifth ed. s.l.:O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472..

. Mathew, C. K. P. M. L. a. W. H. L., 2018. Classification of Imbalanced Data by Oversampling in Kernel Space of Support Vector Machines,. IEEE Transactions on Neural Networks and Learning Systems, Volume 29.

developers., T. i.-l., 2014. imbalance learn. [Online]
Available at: https://imbalanced-learn.org/dev/references/generated/imblearn.under_sampling.NearMiss.html