Satellite Orbit Classification

4AI3 Project Final Report

Premanshu Bhagat
Bachelor of Technology – Software Engineering Tech.
McMaster University
Hamilton, ON Canada
Email: bhagatp@mcmaster.ca

Abstract*—The report presents an algorithm that can accurately classify satellites into respective orbital categories like Low Earth Orbit (LEO), Medium Earth Orbit (MEO), Geostationary Orbit (GEO) and Elliptical. The algorithm takes as input key orbital parameters which include inclination (degrees), eccentricity, apogee(km) & perigee(km) which is used to internally compute semi-major axis(km) and the orbital period(minutes). Given these inputs a Decision Tree Classifier predicts the most probable orbital class. This automated classification aims to support mission planning, reduce manual errors and ensure orbital safety by integrating with collision avoidance systems

Keywords*—Decision Tree classifier; LEO; MEO; GEO; orbital class, satellites*

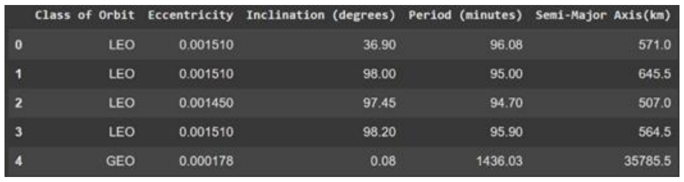
I. Introduction

Traditionally, orbit classification is done manually or via simulations, however due to the surge in launch cadences and increased involvement of private launch agencies like SpaceX, Blue Origin, Rocket Lab etc., orbits have become more clustered and there is a large chunk of data to analyze pre-labelling which is prone to human error and fatigue resulting in misclassifications. The goal of the project is to automate this process of orbit classification using machine learning, it sources data from the historical *UCS Satellite database* which includes orbital features for over 7500 satellites launched globally. The goal is to train a *Decision Tree Classifier* on a set of test data (20%), test it on the remaining chunk of test data (80%), evaluate it and ultimately deploy it to accurately and efficiently categorize orbits into LEO, MEO, GEO and Elliptical classes for user provided satellite data.

II. ALGORITHM OVERVIEW

A. Data Collection

The dataset used for training and testing is sourced from a historical dataset which contains over 28 feature columns with over 7500 satellite entries. The data, though rich, is very impure in context of model training. There are only 6 columns that are relevant for the purposes of orbital classification, those features include 'Class of Orbit', 'Perigee (km)', 'Apogee (km)', 'Eccentricity', 'Inclination (degrees)', 'Period (minutes)'. These features are then condensed to 5 relevant columns that purge the apogee and perigee values into a 'Semi-Major Axis(km)' parameter which dominates the decision process as will be seen ahead in the report. A snippet of the data frame has been shown in Fig 1.



	Class of Orbit	Eccentricity	Inclination (degrees)	Period (minutes)	Semi-Major Axis(km)
0	LEO	0.001510	36.90	96.08	571.0
1	LEO	0.001510	98.00	95.00	645.5
2	LEO	0.001450	97.45	94.70	507.0
3	LEO	0.001510	98.20	95.90	564.5
4	GEO	0.000178	0.08	1436.03	35785.5

Figure 1:Cleaned up data snippet for 5 entries out of over 7500 entries.

B. Algorithm

The algorithm implemented here is the Decision Tree Classifier from the scikit learn library. It is a recursive algorithm that keeps splitting the dataset based on feature thresholds. The splitting continues until a max depth is reached. The result of the algorithm is non-linear decision boundaries based on the training data which can successfully classify satellites into 4 different orbital classes assigned to different orbit classes during data encoding in the data preprocessing stage of algorithm implementation.

Once deployed the model can use the decision boundaries established by the training process to classify user inputs which are detailed in the '*Results'* section of the report.

III. ALGORITHM IMPLEMENTATION

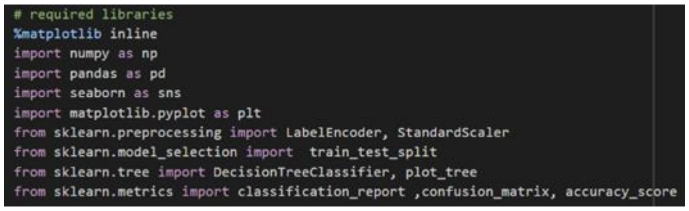
The implementation of this algorithm follows the following workflow:

A. Importing libraries & the data set

As mentioned in the previous section data was sourced from the UCS satellite database which was in a .xlsx format, the *pandas* library was used to extract the data from the excel file to create a structured data frame used for preprocessing purposes.

In terms of the algorithm, it was taken from the *sklearn* library and other data cleanup functions, and model evaluation functions were also sourced from different *sklearn* modules. For visualization *matplotlib* and *seaborn* were used. The *numpy* module was used for vectorization and mathematical purposes.

A list of all libraries & dependencies for the project is shown below in Fig 2.

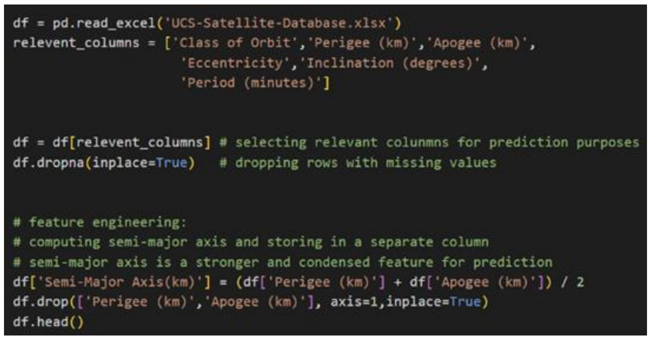


```
# required libraries
%matplotlib inline
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import  train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import classification_report ,confusion_matrix, accuracy_score
```

Figure 2: Required imports for algorithm implementation.

B. *Data Preprocessing*

- After reading the raw data using the *pandas read_excel()* command, relevant columns were extracted from the dataset to be sent further down the preprocessing pipeline as seen in Fig 3, rows with missing columns were dropped using the *dropna*() function with the *inplace* flag set to true, thus ensuring data integrity.



```
df = pd.read_excel('UCS-Satellite-Database.xlsx')
relevent_columns = ['Class of Orbit','Perigee (km)','Apogee (km)',
                    'Eccentricity','Inclination (degrees)',
                    'Period (minutes)']


df = df[relevent_columns] # selecting relevant colunmns for prediction purposes
df.dropna(inplace=True)   # dropping rows with missing values


# feature engineering:
# computing semi-major axis and storing in a separate column
# semi-major axis is a stronger and condensed feature for prediction
df['Semi-Major Axis(km)'] = (df['Perigee (km)'] + df['Apogee (km)']) / 2
df.drop(['Perigee (km)','Apogee (km)'], axis=1,inplace=True)
df.head()
```

Figure 3: Data Preprocessing and & Feature Engineering

- Feature engineering was performed at this stage by computing the value for the semi-major axis and saving it to a separate column and dropping off the less relevant apogee and perigee column which ensures that the most dominant and relevant features are used for training and testing the algorithm. The cleaned-up dataset has been shown in Fig. 1 in *section 2.A* of the report.

- At the end of the preprocessing pipeline was the target label encoding phase and feature scaling step in which the raw string labels assigned for different orbital classes were encoded to cardinal values between [0,3] using the *LabelEncoder()* provided by *sklearn preprocessing* module, the encoded class labels are *0* for *Elliptical*, *1* for *GEO*, *2* for *LEO* and *3* for *MEO.* Standardization was the scaling method applied to the feature vector to standardize the mean to 0 and the standard deviation to 1 by using the *StandardScaler()* provided by *sklearn preprocessing* module, the distribution pre and post scaling can be seen in Fig 4 and Fig 5, respectively.

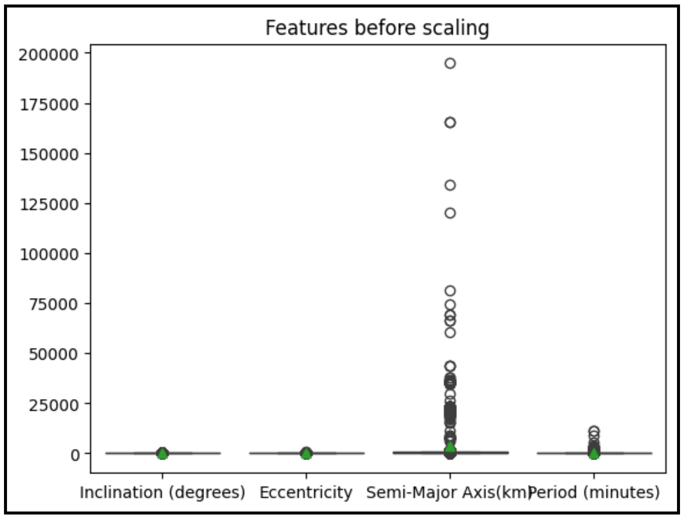


Figure 4: Features before Standardization.

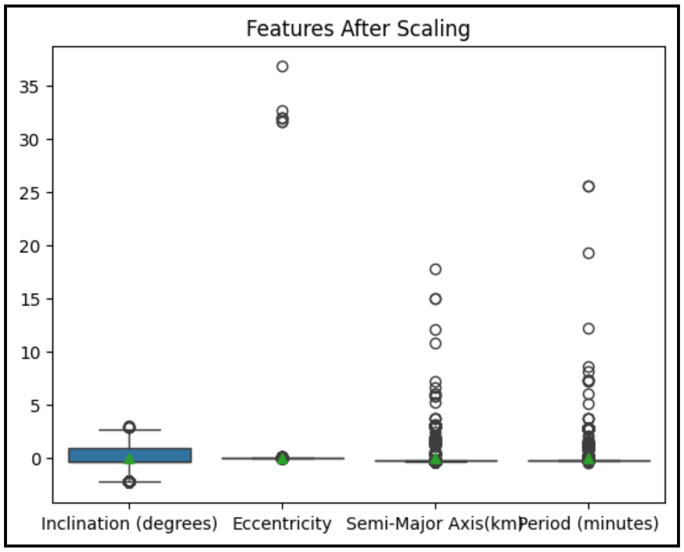
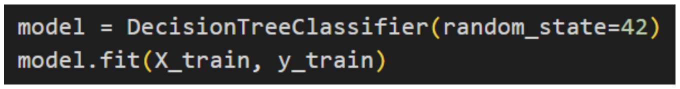


Figure 5: Features after Standardization

- After preprocessing of the data is complete, the feature vector and the target were split, and the data was then split into 80% training and 20% testing data set with stratification and a random state of 42.

C. *Classifier Training*

- The problem requires multi class classification with non linear decision boundaries, so the most simple algorithm for this is a decision tree classifier. To create the model we simply invoke the *DecisionTreeClassifier()* from sci-kit learn and fit the feature vector to the target column which is the encoded orbital class, the code to do so can be seen in Fig 6.



```
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
```

Figure 6: Model Training

- Post training, we can also see the order of dominance of features in the decision-making process, semi-major axis was the most dominant feature among all features with a 95% importance metric followed by eccentricity, inclination and period, this can be seen in the feature importance bar plot in Fig 7.

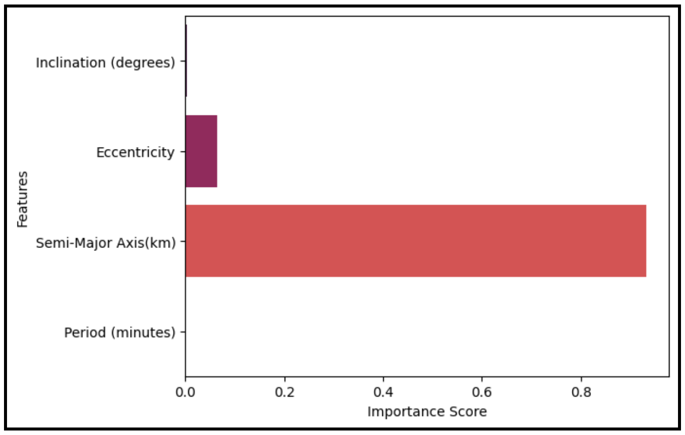


Figure 7: Feature Importance Plot

- The decision flow of the classifier involves recursion until a Gini impurity value of 0.0 is achieved which shows the tree recursively reached the purest nodes thus, signifying precise decision boundary for different target labels, this flow was generated using the *plot_tree* method provided by the *sklearn.tree* module. The decision tree is shown in Fig 8.

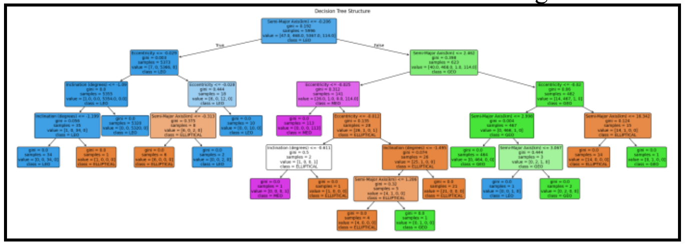


Figure 8: Decision Tree

- Once the model has trained over the training set, the predict() method can then be used to predict class labels for the test set which aids in generating metrics pertaining to accuracy and model performance. Once the predictions on the test set are collected, we can then use different tools like the confusion matrix, accuracy score and classification report which are provided by the *sklearn_metrics* module. A more detailed model evaluation is provided in the *results* section of the report.

IV. RESULTS

The resulting algorithm can be considered a success mainly because of the achieved prediction accuracy of roughly 99.93% with a perfect precision and recall score for 3 orbital classes and only 1 misclassification for the GEO class as can be seen in Fig 9 and Fig 10.

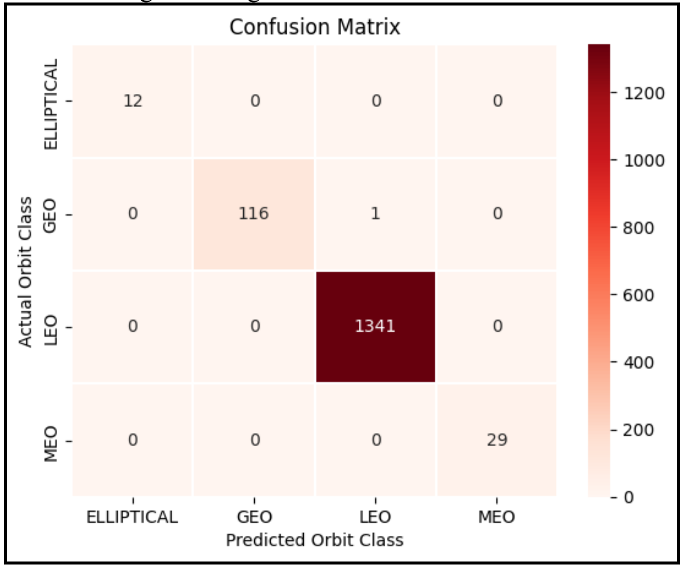
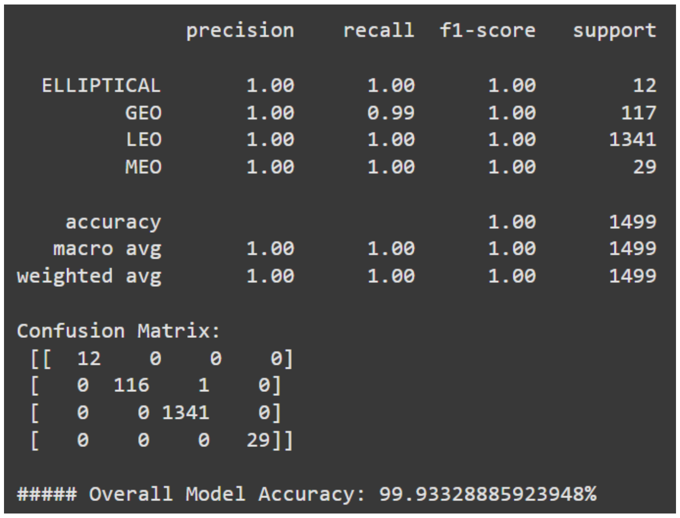


Figure 9: Confusion Matrix

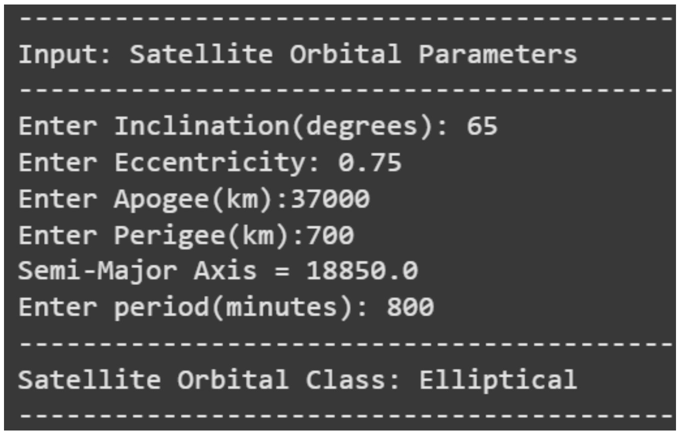


```
              precision    recall  f1-score   support

  ELLIPTICAL       1.00      1.00      1.00        12
         GEO       1.00      0.99      1.00       117
         LEO       1.00      1.00      1.00      1341
         MEO       1.00      1.00      1.00        29

    accuracy                           1.00      1499
   macro avg       1.00      1.00      1.00      1499
weighted avg       1.00      1.00      1.00      1499

Confusion Matrix:
 [[  12    0    0    0]
 [   0  116    1    0]
 [   0    0 1341    0]
 [   0    0    0   29]]

##### Overall Model Accuracy: 99.93328885923948%
```

Figure 10: Classification Report and Accuracy

After the evaluation proved the trustworthiness of the classifier, a user input collection logic was deployed to collect the 4 required features from the user, prepare the collected inputs to feed to the algorithm and generate the prediction and display the orbital label.

An example of an orbit classification based on user input can be seen in Fig 11, the classified orbit was elliptical due to the semi-major axis value, derived from apogee and perigee values alongside the extreme inclination paired with the high eccentricity, the model correctly classified this orbit and when ran for actual data for other orbits the model was highly accurate in its predictions.



```
-----------------------------------------
Input: Satellite Orbital Parameters
-----------------------------------------
Enter Inclination(degrees): 65
Enter Eccentricity: 0.75
Enter Apogee(km):37000
Enter Perigee(km):700
Semi-Major Axis = 18850.0
Enter period(minutes): 800
-----------------------------------------
Satellite Orbital Class: Elliptical
-----------------------------------------
```

Figure 11: User input-based prediction

Such an algorithm is of high relevance in aerospace industry mainly due to the rapidly rising launch cadence due to the entry of private players in the industry and an increasing demand for orbital spots due to increased connectivity needs of the growing population. Such a system would be critical for space traffic management and collision detection as if expanded to compute the probability of collision by considering other satellites with similar parameters could help reduce load on launch agencies to perform manual calculations and comparison pre-launch. Satellite licensing requirements include declaration of orbit type and an automated classification tool would reduce the manual overhead and misreporting chances due to human error. The algorithm can be expanded to classify other forms of orbital debris to create a chart of orbital clutter to push orbital cleanup initiatives especially in the LEO between the 200-900km range. The algorithm could also help in monitoring orbital drift if these parameters change for a certain satellite and could be used to flag undesired orbital behavior. Overall, this algorithm is highly relevant by itself and is great for integration purposes with other pre-existing orbital tracking and documentation systems.

REFERENCES

[1] Wikipedia contributors. (2025, July 23). *List of private spaceflight companies*.Wikipedia. https://en.wikipedia.org/wiki/List_of_private_spaceflight_companies

[2] Płoński, P. (2020, June 22). *Visualize a Decision Tree in 5 Ways with Scikit-Learn and Python*. MLJAR. https://mljar.com/blog/visualize-decision-tree/

[3] *DecisionTreeClassifier*. (n.d.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[4] *Satellite database*. (2005, December 8), Updated (1/2/2024). Union of Concerned Scientists. https://www.ucs.org/resources/satellite-database

[5] *sklearn.metrics*. (n.d.). Scikit-learn. https://scikit-learn.org/stable/api/sklearn.metrics.html

[6] seaborn: statistical data visualization — seaborn 0.13.2 documentation. (n.d.). https://seaborn.pydata.org/

[7] *pandas.DataFrame.dropna — pandas 2.3.1 documentation*. (n.d.). https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.dropna.html

[8] *7.3. Preprocessing data*. (n.d.). Scikit-learn. https://scikit-learn.org/stable/modules/preprocessing.html

[9] Awan, A. A. (2022, October 3). *The Machine Learning Life Cycle Explained*. datacamp. https://www.datacamp.com/blog/machine-learning-lifecycle-explained

[10] *Understanding Earth's orbits and their applications*. (2023, December 6). Keep Track. https://keeptrack.space/deep-dive/orbital-regimes/

[11] Wikipedia contributors. (2025, July 20). *Low Earth orbit*. Wikipedia. https://en.wikipedia.org/wiki/Low_Earth_orbit

[12] Wikipedia contributors. (2024, October 10). *Medium Earth orbit*. Wikipedia. https://en.wikipedia.org/wiki/Medium_Earth_orbit

[13] Wikipedia contributors. (2025a, May 20). *Geostationary orbit*.Wikipedia. https://en.wikipedia.org/wiki/Geostationary_orbit

[14] Wikipedia contributors. (2025b, June 10). *Elliptic orbit*. Wikipedia. https://en.wikipedia.org/wiki/Elliptic_orbit