NASA Hazardous Asteroids Classification

Alison J. March
University of Colorado, Boulder
Boulder, CO 80309-0552
alma2157@colorado.edu

Abstract

This project aims to study and explore effective data mining and Machine Learning methods to identify and classify hazardous asteroids. In addition, we will compare those methods and determine the best machine-learning models for this project.

1. Introduction

The dataset is provided by NASA's Near-Earth Object Observations Program via NeoWs (Near Earth Web Service) Restful API web service for near-earth asteroid information(https://api.nasa.gov/). All the data is maintained by the NASA JPL Asteroid team(http://neo.jpl.nasa.gov/). It consists of 40 data features(columns) and 4,687 data instances (rows) of asteroid entries [1]. We reviewed and pre-processed the dataset to eliminate unnecessary features before using Machine Learning models to complete the classification task.

2. Related Work

2.1 Data Pre-processing and Feature Engineering

So far, we have exported the dataset and done an initial data review and exploratory data analysis with Jupyter Notebook. In the process of data preprocessing and cleaning, we did not find any missing data. Furthermore, the Hazardous versus non-Hazardous Asteroids ratio is 83.89% and 16.11% respectively.

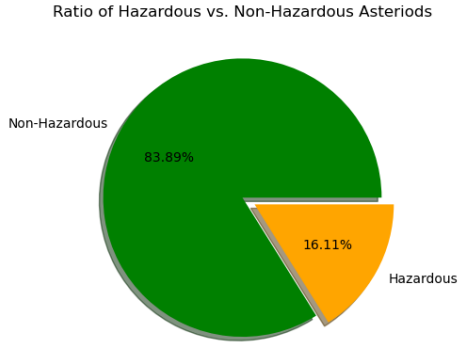


Figure 1

In terms of data normalization, we have further categorized object datatypes into date and categorical columns: date_cols = ['Close Approach Date', 'Orbit Determination Date'] and category_cols = ['Orbiting Body', 'Equinox']. For date_cols, both features have been converted to nanoseconds in a 64-bit-integer format using the *pd.to_datetime()* conversion. On the other hand, category_cols are transformed using the *OneHotEncoder* method.

Redundant features like '***Est Dia in M(min)'*** and '***Est Dia in M(max)'***, '***Est Dia in Miles(min)'***, '***Est Dia in Miles(max)'***, '***Est Dia in Feet(min)'***, and '***Est Dia in Feet(max)'*** were dropped. Furthermore, features like '***Orbiting Body'***, and '***Equinox'*** were eliminated before Feature Selection. A correlation heatmap is also constructed to study the correlation between data features.

2.2 Dimensionality Reduction

2.2.1 Feature Selection

For the remaining 32 data features, we used the Chi-squared based SelectKBest library to rank each feature based on the correlation score: a high score represents a high correlation and vice versa. We want to drop features with a correlation > 90% to prevent multicollinearity, hence the following 6 features have been dropped: ***'Est Dia in M(max)'***, ***'Miles per hour'***, ***'Orbital Period'***, ***'Aphelion Dist'***, ***'Perihelion Time'***, ***'Mean Motion'.*** We also filter out constant and quasi-constant features, and duplicated features, this removes the following 7 features: ***'Neo Reference ID'***, ***'Close Approach Date', 'Orbit Determination Date',' Miss Dist.(lunar)', 'Miss Dist.(kilometers)', 'Miss Dist.(miles)', 'Relative Velocity km per hr***'. Finally, we separate the target feature 'Hazardous' from the list of the remaining 19 feature columns.

We then implemented Sequential feature selection algorithms (SFS) using *Mlxend* (machine learning extensions) library to automatically select a subset of features most relevant to this classification problem (see *Figure 1*). From the below graph, we can see that it indicates that Features #3-11 have relevant high scores. This wrapper method is also called naïve sequential feature selection and works very rarely due to the fact it does not account for feature dependence. To verify its accuracy, we will implement another method for comparison [2].

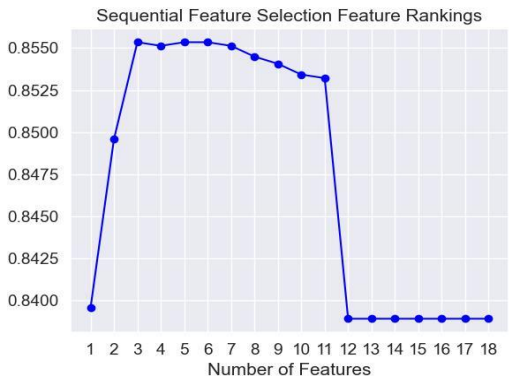


Figure 2

The next algorithm we used is the XGBoost the plot_importance function for feature importance ranking and the result is illustrated in *Figure 2*.

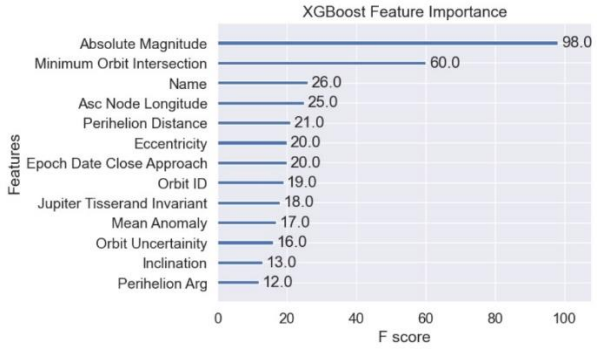


Figure 3

From the above ranking results, we can see that ***'Absolute Magnitude'*** is the most relevant feature followed by ***'Minimum Orbit Intersection'， 'Orbit Intersection', 'Name', 'Epoch Date Close Approach', 'Epoch Date Close Approach',' Orbit ID', 'Orbit Uncertainty', 'Jupiter Tisserand Invariant'，' Perihelion Distance', 'Inclination', 'Eccentricity', 'Asc Node Longitude', 'Perihelion Arg', 'Mean Anomaly', 'Relative Velocity km per sec'*** and ***'Semi Major Axis'***. We can also see why we didn't see ***'Est Dia in M(min)'*** and ***'Epoch Osculation'*** listed in the Feature Importance F1 Score ranking graph, their score is 0 which suggests neither is relevant to predict whether an asteroid is hazardous or not, so we can drop these two features along with ***'Miss Dist.(Astronomical)'***, ***'Semi Major Axis'*** and ***'Relative Velocity km per sec'***. We will be using the remaining 13 predictor features listed in *Figure 2* for Supervised Machine Learning models and evaluations.

2.2.2 Resolving Imbalance

Once the unwanted features have been eliminated, we want to see if the cleaned dataset is balanced or imbalanced since most machine learning algorithms do not work very well with imbalanced datasets. For a classification problem, we can resolve this issue by oversampling the minority class or undersampling the majority class, or simply combining oversampling and undersampling methods to balance the dataset and increase model performance, i.e., by combining *Synthetic Minority Oversampling Technique (SMOTE)* and *Edited Nearest Neighbor (ENN)* method, abbreviated as SMOTE-ENN. Developed by Batista et al (2004) [3], this method combines the SMOTE's ability to generate synthetic minority class examples and ENN's ability to delete some observations from both classes that are having different classes between the observations and its K-nearest neighbor majority class [4]. Before data balancing, we can see that the dataset is imbalanced with 16.11% Hazardous versus 83.89% Non-Hazardous class. After implementing the algorithm, the dataset is more balanced out with 53% versus 47%.

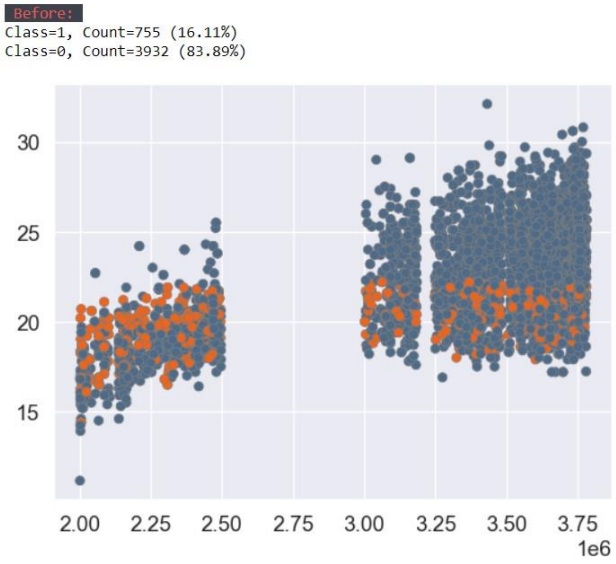


Figure 4. Before the SMOTE+ENN method

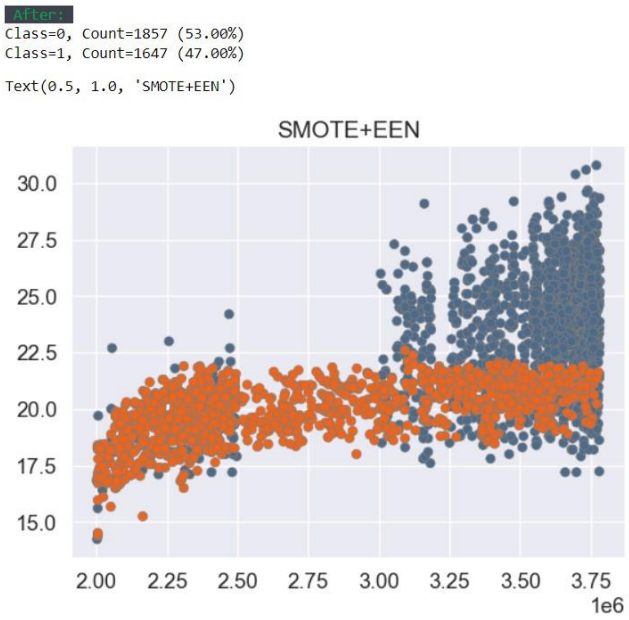


Figure 5. After SMOTE+ENN Method

2.3 Supervised Machine Learning Models

The dataset is split into training and testing datasets with an 80/20 split ratio.

2.3.1 Logistic Regression

Logistic Regression is usually used to estimate the probability of an instance belonging to a specific class, in this example, we are looking for the probability that this asteroid is hazardous. The mechanism behind it is that if the model predicts that the instance belongs to that class (Hazardous), then it's labeled "1", otherwise it's labeled "0" [5].

The logistic regression model in the vectorized form:

$$\hat{p} = h_\theta(x) = \sigma(x^T\theta)$$

Where $\sigma(\bullet)$ is a sigmoid function that outputs a number between 0 and 1, and $\sigma(t) = \frac{1}{1+\exp(-t)}$ is the logistic function. The code can be completed with *skearn.linear_model.LogisticRegression* class [6]. We noticed that before and after the SMOTE-EEN method to balance the dataset, the accuracy score for the Logistic Regression went down from 84.54% to 53.5%. This is normal because SMOTE technique creates synthetic data points based on the original data points and does not copy those data points but creates slightly varied ones. The dataset is balanced so it will predict the minority classes better, but the overall accuracy will decrease.

Furthermore, the most crucial aspect of this result is that we should not rely on the Accuracy score as the parameter to determine the performance. Instead, we can try the F_1 score, AUC, Classification Report containing Recall, and Precision to determine if the model has been improved or not. Figure 5 displays the overall performance metrics for the logistic model.

```
Accuracy: 84.54157783%
```

	precision	recall	f1-score	support
False	0.85	1.00	0.92	793
True	0.00	0.00	0.00	145
accuracy			0.85	938
macro avg	0.42	0.50	0.46	938
weighted avg	0.71	0.85	0.77	938

Figure 6. Metrics Report for Logistic Regression Model

2.3.2 *Decision Trees*

Decision Trees (DTs) are a non-parametric supervised learning method that can be used for both classification and regression, with the goal of creating a model that predicts the target variable value by learning simple decision rules inferred from the data features [7]. The process of Decision Trees is as follows: 1) Import *DecisionTreeClassifer* to make DT models. 2) Split the data into training and testing data. 3) Create a Decision Tree model. 4) Put the training data into the model. 5) Predict the model with the values. 6) Check the accuracy of the model's prediction, where Accuracy= $\frac{TP+TN}{TP+TN+FP+FN}$ [8]. In this decision tree, we see that the tree starts with Minimum Orbit Intersection (<=0.05) and branches out to Absolute Magnitude and branches out to other splits. The Decision Tree predicts a target, whether an Asteroid is hazardous or not. For an asteroid belonging to the testing set, it travels down the tree from the root node until reaches a leaf, deciding to go either left or right node by first testing the feature value against the parent node condition. In this case, Minimum Orbit Intersection causes the split of the training dataset, if Minimum Orbit Intersection is less than 0.05, and Absolute Magnitude is less than 21.85, then the asteroid is hazardous. From the Decision Tree Feature Importance, we can see that the model identifies Absolute Magnitude as the most important feature with an importance of 80% followed by Minimum Orbit Intersection (~20%) and lastly Jupiter Orbit Intersection and Name feature.

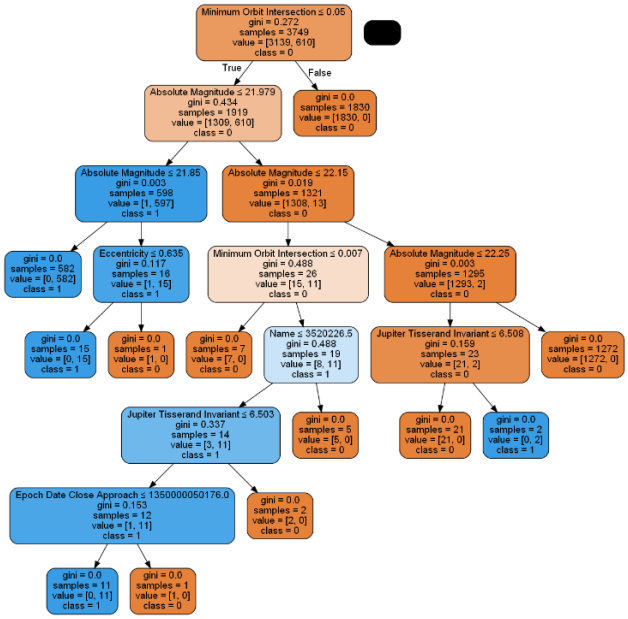


Figure 7. The Decision Tree Graph

The calculated Accuracy score is 99.47%, and ROC AUC is 98.28%, it also gives the feature importance ranking (shown below).

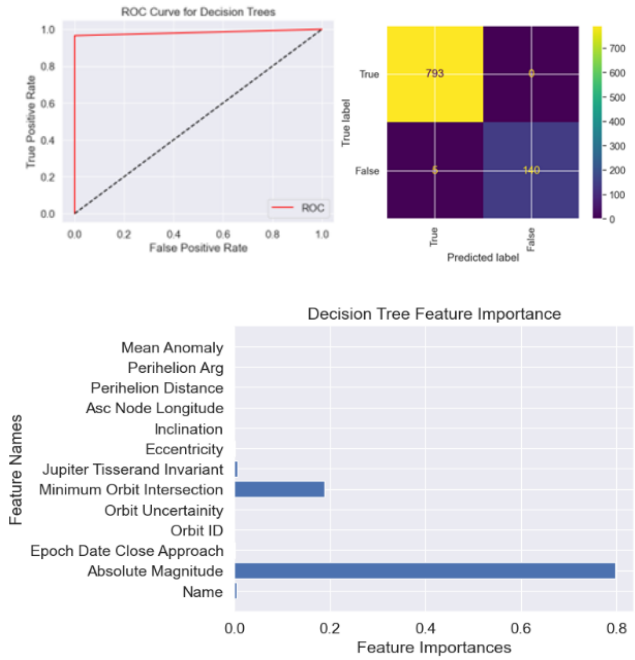


Figure 8

2.3.3 *Support Vector Machines*

SVMs are particularly suitable for the classification of complex-small or medium-sized datasets, which is perfect for our dataset. We will create a pipeline and hyperparameters and an optimal method for searching for the optimal hyperparameters is to use Scikit-Learn's *GridSearchCV*. Aurélien Géron suggested that when you have no idea what value a hyperparameter should use, a simple approach is to try out consecutive powers of 10 or smaller numbers if a fine-grained search is preferred [9]. We used the linear kernel and set the gamma parameter to "auto". The calculated accuracy score is 95.10%, and the ROC AUC score is 98.69%. The downside of this machine learning algorithm is that the training time took much longer than other models, we waited for more than two hours before it outputs the result.

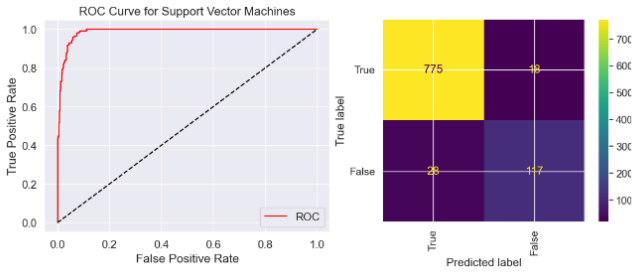
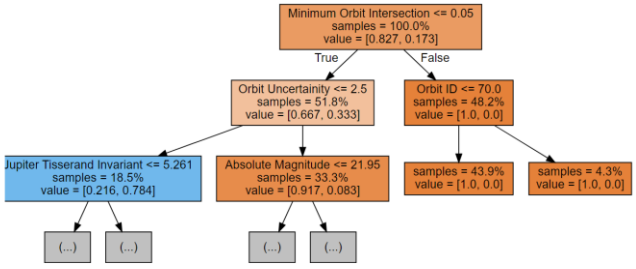


Figure 9

We will incorporate Stochastic Gradient Descent SGD Classifier to compute the ROC curve for both the training and testing dataset with this condition: *model = SGDClassifier(loss ='hinge', alpha = 0.0001, penalty='l2', class_weight='balanced')* where hinge refers to an implementation of Linear SVM, and we have 97.92% for the training set and 98.29% for the testing set respectively.

2.3.4 *Random Forest*

Random Forests is an ensemble learning method for both classification and regression as well as other tasks that operate by constructing a multitude of decision trees at training time and is generally trained via the bagging method. We will use the *RandomForestClassifier* class from sklearn.ensemble to train the model and calculate the training and testing accuracy scores respectively. With parameters of 500 estimators, and 18 leaf nodes, our accuracy is 99.04%, and the ROC AUC score is 99.96%.



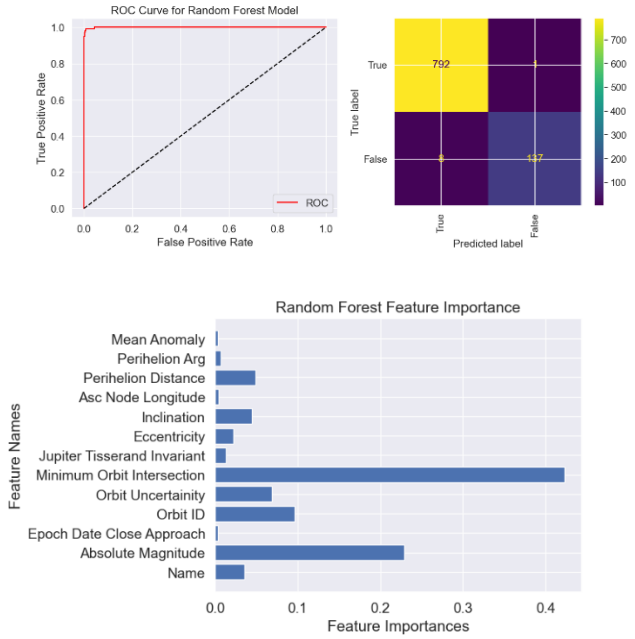


Figure 11

The Feature Importance graph illustrates the algorithm ranks ***Minimum Orbit Intersection*** (42.68%) as the most important feature followed by ***Absolute Magnitude*** (23.11%), ***Orbit ID*** (8.69%), ***Orbit Uncertainty*** (6.58%), ***Perihelion Distance*** (4.68%), ***Inclination*** (4.67%), ***Name*** (3.81%), ***Eccentricity*** (2.06%), ***Jupiter Tisserand Invariant*** (1.28%), ***Perihelion Arg*** (0.66%), ***Asc Node Longitude*** (0.42%), **Mean Anomaly** (0.33%)**,** and ***Epoch Date Close Approach*** *(*0.41%). This result differs from the feature rankings of Decision Trees as well as XGBoost. However, all three algorithms agree that both ***Minimum Orbit Intersection*** and **Absolute Magnitude** are the key features to determine if an asteroid is hazardous.

For the Random Forest algorithm, we also explored the idea of using Hyperparameter Tuning with Randomized Search with *RandomizedSearchCV* function. The mechanism behind it is we sample values from a statistical distribution where a specific number of estimators are defined in the parameter list. For example, for this experiment, we set the following condition: param_dist = {"n_estimators": [20,50,100,200,250,300,350,400],"max_depth": scipy.stats.randint(1, 1500), "max_features": ['auto', 'sqrt'], 'criterion':["gini"]

n-estimators represent the maximum number of trees in a forest, and max–features are the maximum number of features taken into consideration when splitting a node, here we set it to be automatic at every split, and max-depath to be a random integer from 1 to 1500. We also set the cross-validation splitting to 5 folds and the number of iterations to 5 as well. The best hyperparameters are

{'criterion': 'gini', 'max_depth': 807, 'max_features': 'sqrt', 'n_estimators': 250}

Our final training accuracy is 100% and the testing accuracy is 99.25%, which is a slightly better improvement than the original Random Forest model: 99.04%.

2.3.5 K Nearest Neighbor

K-Nearest Neighbor (KNN) algorithm is a popular supervised machine learning technique that can be used in both classification and regression problems. Neighbors-based classification is a type of instance-based learning which does not attempt to construct a general internal model, but rather stores instances of training data [11]. We use *KNeighborClassifer* to implement learning based on the k nearest neighbors of each query point, and we also cross-validation to find the best k value with the help of cross_val_score function from sklearn.model_selection, and gives k=1 as the best value for the highest accuracy score(see below).

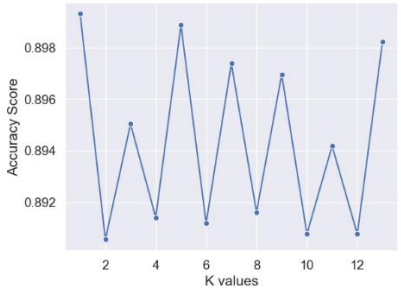


Figure 12

The final ROC AUC score is 83.72%, Accuracy is 92.86%, Precision is 78.68%, and Recall is 73.79%.

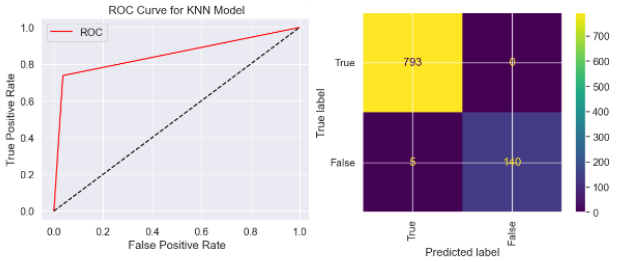


Figure 13

2.3.6 XGBoost

XGBoosting stands for Extreme Gradient Boosting and is an optimized implementation of Gradient Boosting, and aims to be extremely efficient, flexible, and portable [12]. In recent years, XGBoost has gained significant favor due to the fact it helped individuals and teams win most of the Kaggle structured data competitions [13]. Due to this reason, I have much more faith in expecting better model results than some of the other machine learning models.

The Accuracy score for XGBoost model is 99.147% and the ROC AUC score is 99.76%.

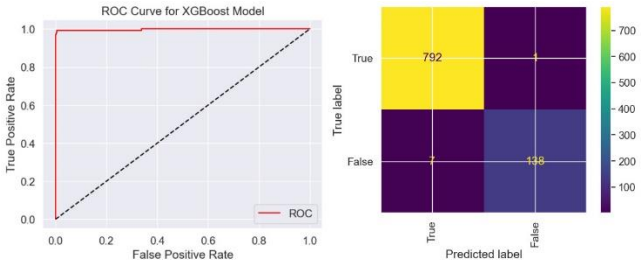


Figure 14

3. Proposed Work

3.1 Evaluation Plan

So far, we have split the cleaned dataset into the ratio of 80/20 train/test sets. Six machine learning models have been implemented, trained, and tested. We have also incorporated Cross-Validation to evaluate a model by using Scikit-Learn's cross_val_score() function. All models contain a precision-recall curve as well as the confusion matrix. A precision_recall_curve() function can be utilized to visualize the precision and recall versus the decision threshold. The second method to evaluate the model is to compute the confusion matrix, this can be done with Sciki-Learn's cross_val_predict () function. It provides precision and recall scores, and we can combine these two into a single metric called F_1 score. These two methods have helped us rank the efficiency of the six models effectively, but we need further evaluation, this will involve model calibration and TensorFlow.

Model calibration is crucial information to have and helps access whether our machine-learning classifiers are predicting probabilities of data belonging to each possible class [14]. For this reason, we have implemented Calibration plots (also known as Reliability curves). Those plots show any potential mismatch between the model-predicted probabilities. A perfectly calibrated model will have a straight line corresponding to the identity function. The general rule of thumb to read the plot is that the better the calibration, the closer the plot curve is to the straight line. This plotting method is also known as Platt scaling (or sigmoid method) which transforms the outs of a classification model into a probability distribution over classes [15].

Lastly, TensorFlow will be used for learning purposes to perform binary classification on the NASA Asteroid dataset and compare the results with the traditional ml models.

3.2 Calibration Plots

To obtain the calibration curves, we have imported CalibratedClassiferCV and CalibrationDisplay from Sklean.calibration and used the scikitplot as well as the plot_calibration_curve function.

In general, when the dots are above the dotted line, the model is under-predicting the true probability, and over-predicting below the line. From the results below, we can see that Decision Tree Classifier model is perfectly calibrated, and Logistic Regression and SVM have relatively similar calibrations that are closer to the perfectly calibrated line. KNN, on the other hand, remains aligned from the [0, 0.2] range and begins diverging out afterward, the model overestimates mean predicted values. Lastly, we see the most drastic curves present are XGBoost (red) and Random Forest(blue), where they both have underestimation (inflated region) and overestimation (deflated region): XGBoost curve

contains inflated region in [0, 0.5] range, and [0.85, 1] for deflated region in [0.48, 0.76], and [0.08, 0.18] for Random Forest curve.

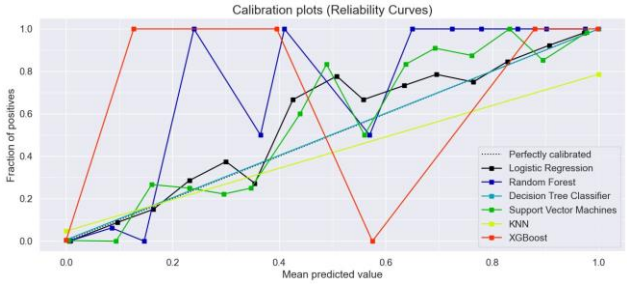


Figure 15

3.3 TensorFlow and Neural Networks

In this final step of our data mining and machine learning project, we have used Neural Networks algorithm and compared the result with the traditional machine learning models. Prior to that, we made a copy of the cleaned_features and converted True and False to 1 and 0 for the target column before shuffling the rows and splitting the dataset to 80% training and 20% testing. The purpose of shuffling data is to reduce variance and ensure that models remain general and overfit less and thus increase accuracy, normalization process is also carried out before creating a *keras* sequential neural network.

The number of neurons in the input layer is equal to features in the data, whereas the number of neurons in the output depends on whether the model is a regressor or classifier. Since we are doing classifier, hence it will have a single neuron based on the class label. For experiment purposes, we have used 128 for both the input layer and hidden layer, and 1 for the output layer. Relu(rectified linear unit activation function) is used on the first two layers and the Sigmoid function in the output layer.

For binary classification problems, *binary_crossentropy* is used as the loss function and *rmsprop* as the optimizer due to the fact it calculates the moving average of squared gradients and scales the learning rate, so the algorithm goes through saddle point faster than most other optimizers [16].

With 256 epochs and a batch size of 10, our training data has 99.33% accuracy and testing data has 98.51% accuracy respectively at epoch # 28.

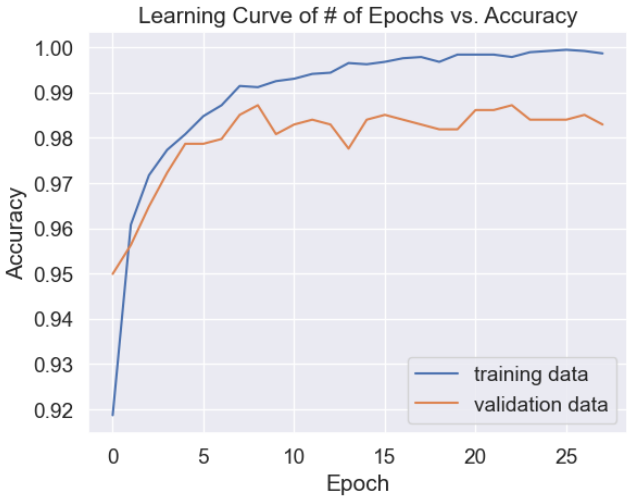


Figure 16

We also check what the model predicts and test if it is really accurate by verifying with the first 10 randomized rows.

```
print(prediction[0:10].round())
```

```
4034    False                [[0.]
3607    False                 [0.]
3933    False                 [0.]
1736     True                 [0.]
1533    False                 [1.]
2899    False                 [0.]
1892    False                 [0.]
3375    False                 [0.]
2048    False                 [0.]
4331    False                 [0.]]
Name: Hazardous, dtype: bool
```

From the left, we see only Row # 1533 as a hazardous asteroid, which the model predicts correctly on the right side, and predicts the rest correctly as non-hazardous.

Lastly, we want to check whether using an accuracy metric is sufficient to evaluate the neural network model. Our results show 98.51% in Accuracy, 94.56% in Precision, 95.86% in Recall, and 95.21% in F1 score. This demonstrates that our model is very efficient in terms of predicting the class label of the NASA asteroids.

4. Discussion

We now can see from the model summary table; we see that Decision Tree has the most accuracy and Random Forest has the highest ROC-AUC score. The ROC-AUC curve is a performance measurement to determine how much the model is capable of distinguishing between classes. The higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. Random Forest and XGBoost models also performed extremely well. On the other hand, when AUC is approximately 0.5, this indicates that the model has no discrimination capacity to distinguish between positive and negative class (hazardous vs. non-hazardous). Figure 17 shows both KNN, Support Vector Machine and Logistic Regression all have this score and therefore did not perform well for the classification.

Model summary table

Machine Learning Models	Accuracy	ROC AUC Score
1. Logistic Regression	84.54% before SMOTEEN, 53.5% After	54.96%
2. Random Forest	99.04%	99.96%
3. Decision Tree	99.47%	98.28%
4. Support Vector Machine	95.10%	98.69%
5. KNN	92.86%	83.72%
6. XGBoost	99.15%	99.76%

Figure 17

5. Conclusion

We have determined that three machine learning models: Random Forest, XGBoost, and Decision Tree along with Neural Network to be more effective models to classify asteroids. Coincidentally all three models have a Features Importance function to help us identify the most relevant feature variables- particularly using XGBClassifer feature importance, with just top two features ***Absolute Magnitude*** and **Minimum Orbit Intersection**, we can predict 100% accuracy. Random Forest and XGBoost standing out as the winners as both their accuracy score and ROC-AUC reached close to 100%, however, the Decision Tree model is the most perfectly calibrated model which suggests it might be a more reliable model than the other two. Finally, the Sequential Neural Network trained and tested using TensorFlow also shows similar results as the previously mentioned three models, effectively concluding our project analysis.

6. References

[1] Shruti Mehta. 2018. NASA: Asteroid classification. (March 2018). Retrieved May 20, 2023, from https://www.kaggle.com/datasets/shrutimehta/nasa-asteroids-classification

[2] https://analyticsindiamag.com/a-complete-guide-to-sequential-feature-selection/

[3] Batista, G. E., Prati, R. C., & Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD Explorations Newsletter, 6(1), 20–29. https://doi.org/10.1145/1007730.1007735

[4] Viadinugroho, R. A. A. (2021, September 30). Imbalanced classification in Python: Smote-enn method. Medium. https://towardsdatascience.com/imbalanced-classification-in-python-smote-enn-method-db5db06b8d50

[5] Aurélien Géron. 2019. Chapter 4. Training Models. In Hands-on machine learning with SCIKIT-learn, Keras, and TensorFlow: Concepts, tools, and Techniques. SEBASTOPOL: O'REILLY MEDIA, 142–151.

[6] scikit-learn(2009). Sklearn.linear_model.logisticregression. Retrieved May 21, 2023, from https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[7] scikit learn. (2009). 1.10. Decision Trees — scikit-learn 0.22 documentation. Scikit-Learn.org. https://scikit-learn.org/stable/modules/tree.html

[8] Classification: Accuracy. (n.d.). Google for Developers. https://developers.google.com/machine-learning/crash-course/classification/accuracy

[9] Aurélien Géron. 2019. Chapter 2. End-to-End Machine Learning Project. Hands-on machine learning with SCIKIT-learn, Keras, and TensorFlow: Concepts, tools, and Techniques. SEBASTOPOL: O'REILLY MEDIA, 76.

[10] XGBoost Documentation — xgboost 1.7.5 documentation. (n.d.). https://xgboost.readthedocs.io/en/stable/

[11] 1.6. nearest neighbors. scikit. (n.d.). https://scikit-learn.org/stable/modules/neighbors.html

[12] What is XGBoost? (n.d.). NVIDIA Data Science Glossary. https://www.nvidia.com/en-us/glossary/data-science/xgboost/#:~:text=The%20GPU%2Daccelerated%20XGBoost%20algorithm,dataset%20concurrently%20on%20the%20GPU.

[13] Aurélien Géron. 2019. Chapter 7. Emsemble Learning and Random Forests. Hands-on machine learning with SCIKIT-learn, Keras, and TensorFlow: Concepts, tools, and Techniques. SEBASTOPOL: O'REILLY MEDIA, 208.

[14] Blachowski, W. (2021, October 4). A guide to model calibration. Wunderman Thompson Technology. https://wttech.blog/blog/2021/a-guide-to-model-calibration/#:~:text=The%20most%20common%20way%20of,the%20probabilities%20observed%20in%20data.

[15] Wikimedia Foundation. (2023, January 30). Platt scaling. Wikipedia. https://en.wikipedia.org/wiki/Platt_scaling

[16] Bushaev, V. (2018, September 2). Understanding rmsprop - faster neural network learning. Medium. https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a