An Overview of Asteroid Danger Classification

CIS 520 Final Project

Sriram Tolety[1], Dhruv Gupta[1], and Rishikesh Madabhushi[1]

[1]University of Pennsylvania

Abstract

Identification of potentially dangerous asteroids is costly and of great importance. To this end, a variety of Machine Learning techniques were analyzed in efficacy of asteroid danger detection. Multiple Neural Networks were constructed and tested, in addition to classical techniques such as Random Forest Classifiers and K-Nearest Neighbors. More advanced techniques, namely AutoML, XGBoost, Ensemble, were also used. We find that some individual neural nets perform the best with a perfect recall, but the XGBoost model is the most complete with both a high recall of .803 and MCC of .848.

1 Motivation

Asteroids have caused great damage in Earth's history, and present an ever-present threat to modern day society. The ability to identify potentially dangerous asteroids presents space agencies and governments the opportunity to proactively work to either defend from or attack the offending asteroids. NASA invests heavily in this domain - from the construction of space telescopes [1] to the establishment of CNEOS [2], to classify PHAs (*Potentially Hazardous Asteroids*). Constant surveillance and data analysis is required to correctly classify dangerous asteroids among the NEOs (*Near-Earth Objects*). These methods mostly rely on physical formulations to predict PHAs. These methods are quite intensive and require a thorough understanding of astrophysics. Here, we can make use of machine learning to reinforce those predictions. Thus, various machine learning strategies will be explored to identify asteroids as being potentially hazardous, thereby allowing actions to be taken in response.

2 Related Work

Previous work in the domain consists of various studies that employ some methods that we used along with advanced analyses requiring simulation and combining it with machine learning. A thorough analysis by Jatin Kataria[3] uses traditional and advanced classifiers, following class balancing methods and imputation. The traditional models used here are Logistic Regression, K-Nearest Neighbors, Gaussian Naive Bayes, Random Forest and Extra Trees, and Support Vector Machines. The study emphasises on precision along with recall as a metric for ranking the methods. Deep learning is thus employed to improve on the results from traditional classifiers. The reference sticks to one certain architecture while conducting a grid search over activation functions and optimizers. The results show that the deep learning model outperforms in terms of precision.
Other existing research [4] uses the above traditional classifiers, albeit for a smaller dataset. They also include a method that we have tried in our analysis - XGBoost. Here the metric used is purely test accuracy, whereby the Random Trees (n=15) and XGBoost methods outperform all the others. A more advanced technique is shown by Hefele and Bortolussi that combines a neural network with synthetic data generated using simulation [5]. However they only utilize 5 features/input neurons in their ANN . About 300,000 Potentially Hazardous Asteroids (PHA) are generated using a physics-based algorithm to feed into the neural network and train in. This is an interesting approach to the problem of class imbalance and goes to show how real-life models can be integrated into a machine learning classifier, as opposed to using SMOTE learning or other methods to rectify the imbalance.

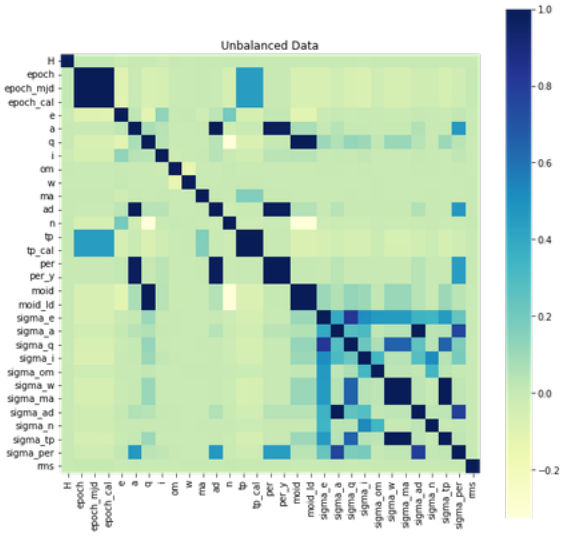

Figure 1: Feature Correlation, Unbalanced Data

3 Dataset

An existing dataset provided by NASA-JPL was used, as published on Kaggle [6]. Nearly 500MB, the data consists of 45

columns each describing an asteroid identified and observed by NASA, totalling 958,523 observations in total.

Including an id for each asteroid as well as its classification as being potentially dangerous, there are 33 numerical values, 6 strings, 2 booleans, and 4 other fields.

Therefore, in total, the dataset provides 43 input features to 1 output label. There are 2,066 identified as potentially hazardous, 936,537 determined to be safe, and 19,921 unclassified. Immediately, it must be noted that the low number of positive identifications (<1%) presents a challenge in the forthcoming analysis.

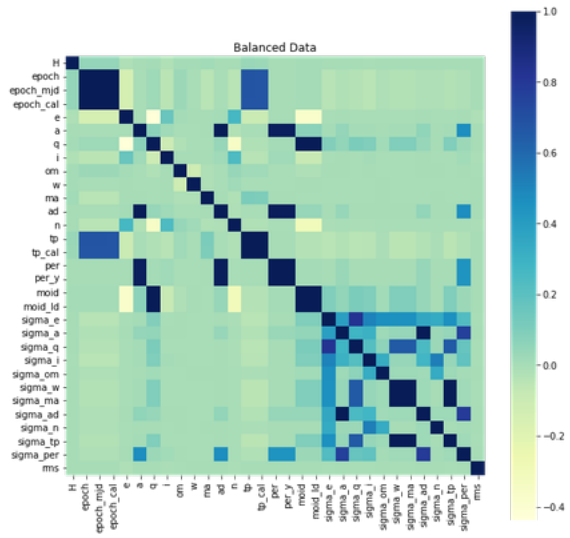

Figure 3: Feature Correlation, Balanced Data

4 Problem Formulation

The problem can be formulated as such: given an observation containing 241 features (explained in Section 5), the output must be a classification of the input asteroid as being potentially hazardous or not. It must be noted that the number, types, and scales of features were modified during data prepossessing. This task can be represented in terms of binary classification, between the classes potentially hazardous and safe. Binary Cross-Entropy was used as the loss function - as it is the default loss function in binary classification problems [7].

5 Data Preparation

In order to create a binary classification problem all unclassified observations were removed from the dataset. The features 'id', 'spkid', and 'full_name' were unique to each observation and were hence removed. 'pdes' and 'name' are generated from 'full_name', and so those features were also removed. The features 'prefix' and 'equinox' were singular values and were therefore removed due to their un-importance in facilitating analysis.

The features 'diameter', 'albedo' and 'diameter_sigma' had 85% missing values and were removed, due to their inability to be calculated nor measured. However, there were only a few values missing from 'H', representing absolute magnitude. These missing values were calculated using 'albedo' (p) and 'diameter' (D), and was imputed using the a linear regression model across other features [8].

Package: *sklearn.linear_model.LinearRegression()*

$$D = \frac{1329}{\sqrt{p}} * 10^{-.2H}$$

The features 'moid', 'sigma', and 'pha' were missing for the same rows wherein 'pha' was missing, and represented 2% of the data. Due to the small scope of observations, these entries were also removed. Any entries which were missing 'sigma_ad' and 'ma' (1 observation) were also removed.

The features 'neo', 'pha' and 'class' were converted to categorical variables. The feature 'orbit_id' feature had 525 unique labels, of which 331 occurred fewer than 10 times. There were renamed to 'others' as as to prevent loss of data but reduce the complexity of the feature. All numerical features were normalized using min-max scaling. Finally, 'neo', 'class', and 'orbit_id' were converted into one-hot encoded values.

Therefore, in total there were 241 input features, and 1 output label.

A 70-30 split was used to generate the train and test data. A 25-75 (20-80 in some cases) split was used to generate the validation and test sets from the test data.

To account for the large imbalance in training data, a SMOTE re-balance was used on the training set so as to provide higher-quality training data for ML algorithms. Synthetic Minority Oversampling Technique (SMOTE) is a technique which is used in imbalanced classification problems. SMOTE selects examples which are relatively close in the features space, and generates a synthetic example by examining the space between the selected examples. In this way, SMOTE is used to over-sample minority classes.

Package: *imblearn.over_sampling.SMOTE*

	Safe	Danger
Raw	655586	1431
SMOTE	655586	655586

6 Metrics

The following metrics are provided for analysis: Accuracy, Precision, Recall, F1 Score, MCC, and Geometric Mean. It is important to note that accuracy is insignificant in this problem, due to the extremely high prevalence of true-negatives in the dataset. Precision is unimportant in this problem, as the sheer magnitude of a true-positive dwarfs the marginal cost of a false positive - in other words, it is better to be safe than sorry. F1 score was used to combine Precision and Recall in an understandable manner. The Mathews Correlation Coefficient (MCC) was used. MCC is calculated in the following manner:

$$MCC = \frac{TP*TN - FP*FN}{\sqrt{(TP+FP)*(TP+FN)*(TN+FP)*(TN+FN)}}$$

MCC takes into account all four values in a confusion matrix, and ranges from -1 to 1. A high value close to 1 represents the model predicting both classes well, even if one class is disproportionately under-represented, as is the case in this problem.

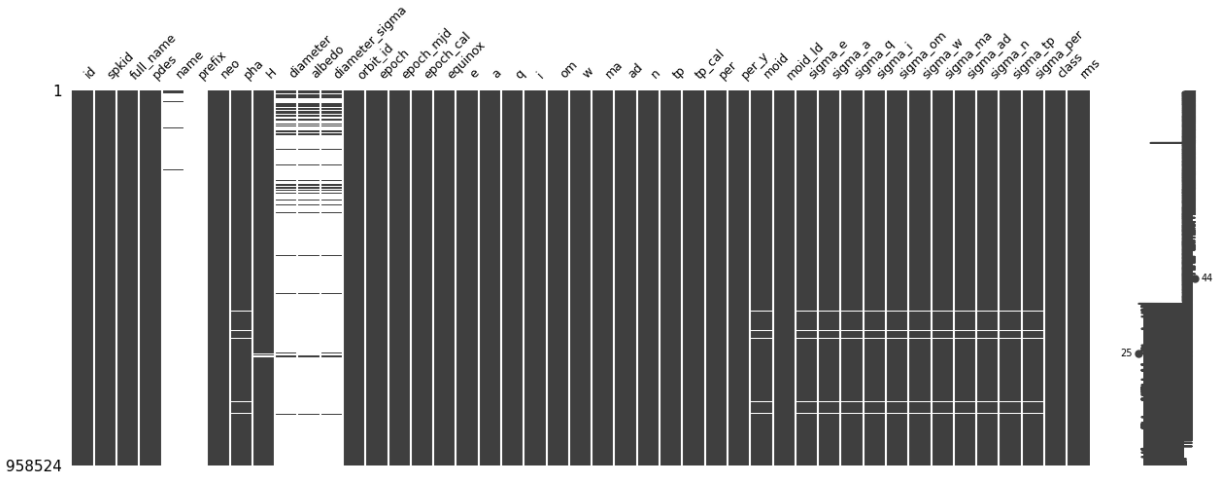

Figure 2: Missing Data Analysis, Raw Data

The geometric mean is the square root of the product of sensitivity and specificity, thereby combining the two metrics and accounting for the imbalance of our dataset.

Given this context, Recall and MCC were prioritized in evaluation of models.

Package: *sklearn.metrics and imblearn.metrics*

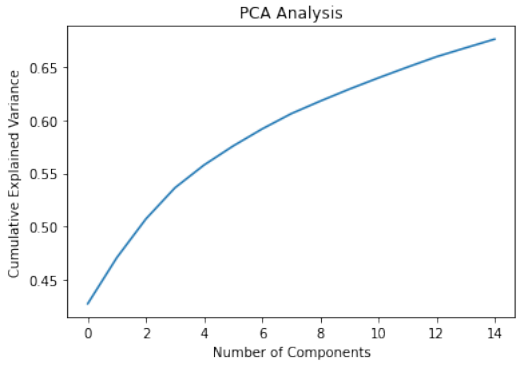

Figure 4: PCA Analysis

7 Methods

7.1 PCA

To reduce the dimensionality of the problem, PCA was used so as to focus on critical eigenfeatures. This was corroborated with a Random Forest so as to identify critical features. PCA was used to reduce the input to the 15 eigenfeatures which explained the highest proportion of variance within the dataset.

Package: *sklearn.decomposition.PCA*

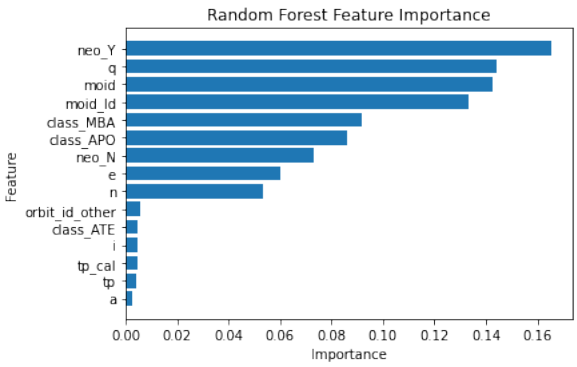

Figure 5: Random Forest Feature Importance

7.2 Traditional Methods

7.2.1 Baseline: K-Nearest Neighbors

A baseline method was used so as to establish a base case against which other algorithms could be tested. For this purpose, K-Nearest Neighbors was chosen, as it was a simple non-parametric method which seems applicable in the context of the problem. We thought that within this problem, asteroids with similar characteristics would tend to behave similarly.

Package: *sklearn.neighbors.KNeighborsClassifier*

7.2.2 Logistic Regression

Logistic Regression was chosen due to its simplicity and base-level efficacy for binary classification problems. Logistic Regression was performed on the non-PCA training data as well as post-PCA data.

Package: *sklearn.linear_model.LogisticRegression*

7.2.3 Random Forest

A Random Forest classifier was used for classification. Comprised of many decision trees, it avoids the pitfalls of over-fitting which decision trees are prone to. This may lead to an increase in performance.

Package: *sklearn.ensemble.RandomForestClassifier*

7.2.4 Support Vector Machines

SVMs provide an understandable and clean separation of observations in binary classification problems. Utilizing a kernel transformation, the best hyper-plane decision boundary is found in order to generate 2 classes. However, training would be quite expensive due to the large size and dimensionality of the dataset. As a result, SVMs were not included in our analysis.

7.3 Advanced Methods

7.3.1 Sequential Neural Networks

A variety of neural networks were considered, with varied architectures, activation functions, and optimizers. The input layer consists of 15 neurons, with a single output neuron. Presented below are the various multi-layer networks used, following the naming scheme <optimizer>_<activation>_<Hidden Layer 1 Size>_<Hidden Layer 2 Size>...

- adam_relu_10_10
- adam_relu_20_10
- adam_relu_20_20
- adam_relu_8_4_2
- adam_relu_20_10_5
- adam_tanh_10_10
- sgd_relu_10_10 *(momentum = 0.9)*
- sgd_tanh_10_10

Package: *keras.models.Sequential*

7.3.2 Ensemble Neural Network

Using a mean-voting scheme, an ensemble model was constructed using the 3 best performing models as determined by Recall and MCC.

Package: *keras.Model*

7.3.3 XGBoost

XGBoost is a fast gradient tree boosting algorithm suitable for medium sized data. Performing parallel tree generation and computations, in addition to pruning, cache-aware access, and regularization, XGBoost was chosen due to its speed and high performance while avoiding over-fitting. It was chosen over Adaboost due to its speed and use of strong learners as opposed to weak learners.

Package: *xgboost.XGBClassifier*

7.3.4 AutoML

AutoML automates the search and generation of optimal model architecutres as well as hyperparameter tuning over a set of predetermined models. It creates a final ensemble based on the metric or scoring function that is provided to it. We use *autosklearn.metrics.recall* and provide a list of classifiers for the ensemble to consider.

Auto-SKLearn can be described as:

$$A^*, \lambda_* \in argmin_{A^{(j)} \in A, \lambda \in \Lambda^{(j)}} \frac{1}{K} \sum_{i=1}^{K} L(A_\lambda^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)}) \quad (1)$$

Package: *autosklearn.classification.AutoSklearnClassifier*

8 Experiments and Results

8.1 Traditional Methods

8.1.1 Baseline: K-Nearest Neighbors

After 5-fold Cross Validation over a Grid Search on the k value from (1, 25), an optimal value of k = 7 was found. KNN was ran on non-PCA imbalanced data, non-PCA balanced data, and finally PCA balanced data. Given the large increase in performance when using balanced data, SMOTE balanced data was used during analysis of all other models.

8.1.2 Logistic Regression

Logistic Regression was run on the training data both before and after PCA dimensionality reduction. 5000 was set as the max number of iterations.

8.1.3 Random Forest

Due to a lack of time, the Random Forest algorithm was run using only 100 estimators, without any hyperparameter tuning. It is important to observe that the performance of the algorithm is significantly worse when run on post-PCA data. This will be explored further.

8.2 Advanced Methods

8.2.1 Sequential Neural Networks

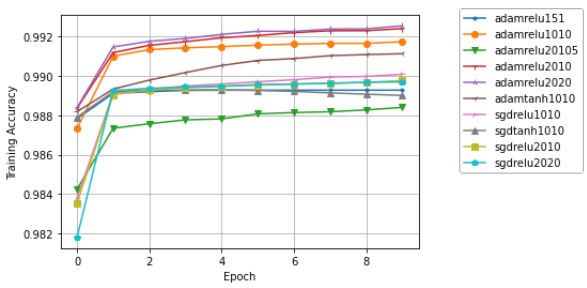

Figure 6: Training Accuracy per Epoch Curves for Neural Nets

Model	Accuracy	Precision	Recall	F1	MCC	GMean
KNN(7)	.998	.448	.216	.291	.310	.466
KNN(7)'	.994	.243	.786	.372	.435	.884
KNN(7)*'	.997	.008	.874	.015	.067	.805
LR'	.987	.147	.970	.255	.375	.979
LR	.998	.555	.143	.227	.281	.381
LR*'	.841	.014	.995	.028	.108	.915
RF'	.998	.540	.860	.663	.663	.927
RF*'	.995	.005	.006	.006	.003	.092
adam_relu_15_1*'	.871	.017	1.000	.034	.122	.933
adam_relu_8_4_2*'	.904	.017	.748	.034	.105	.823
adam_relu_10_10*'	.909	.023	.928	.044	.137	.918
adam_relu_20_10_5*'	.876	.017	.975	.034	.122	.924
adam_relu_20_10*'	.914	.021	.822	.041	.123	.870
adam_relu_20_20*'	.882	.012	.630	.024	.075	.746
adam_tanh_10_10*'	.903	.023	.995	.044	.142	.948
sgd_relu_10_10*'	.903	.023	1.000	.044	.143	.950
sgd_tanh_10_10*'	.891	.020	1.000	.040	.134	.944
sgd_relu_8_4_2*'	.902	.022	.995	.044	.142	.948
sgd_relu_20_10*'	.897	.021	1.000	.042	.138	.947
sgd_relu_20_20*'	.897	.022	1.000	.042	.139	.947
Ensemble	.911	.024	.977	.047	.147	.943
XGB'	.999	.897	.803	.847	.848	.896
XGB*'	.979	.015	.130	.027	.038	.358
AutoML*'	.822	.012	.991	.025	.100	.902

Figure 7: Model Results: PCA(*), Balanced(')

Due to their ability to provide robust results in a variety of situations, multiple neural networks were trained, utilizing a variety of activation functions, optimizers, and architectures, over 10 epochs and a batch size of 64. Binary Cross-Entropy Loss was used, following literature in the subject. An 80-20 split was used to validate the networks as well. Each dense layer was followed by a dropout with probability 0.2. A fixed learning rate of 10^{-3} is used after having performed a grid search across $(10^{-1}, 10^{-3}, 10^{-5})$. We use a single-layer perceptron with Adam optimizer and ReLU activation function as the simplest neural net (15 inputs to 1 output). The output layer uses a Sigmoid activation function to transform the output to the range [0, 1]. *Keras* (from *tensorflow*) is used to create the neural net models and the metrics class from *sklearn* is used to gauge the different architectures on a range of metrics.

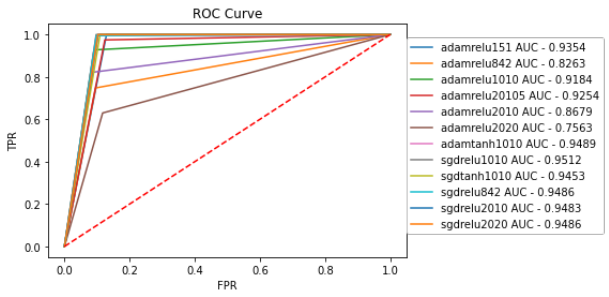

Figure 8: ROC Curves for Neural Nets

8.2.2 Ensemble Neural Network

Using the 3 models with the highest F1 score and recall, as well as 2 adam_relu models, we construct an ensemble method that averages the output of each of the individual models. We use adam_tanh_10_10, sgd_relu_10_10, sgd_relu_8_4_2, adam_relu_8_4_2, and adam_relu_10_10 for this purpose.

8.2.3 XGBoost

We run the XGBoost on the SMOTE rebalanced data. We also perform 5 fold cross validation to tune the hyperparameters. The parameters we searched for were gamma for values [0.5, 1, 5], min child weight: [1, 5, 10], and maxdepth for values [3, 5]. The gridsearch found the best model to be gamma = 1, max depth = 5 and min child weight = 1. After determining the optimal hyperparameters, two XGBoost models were trained - one on the PCA Balanced data, and one on the Balanced, no PCA data.

8.2.4 AutoML (auto-sklearn)

Recall was used as a scoring function alongside a list of allowed classifiers. To avoid overfitting, the autosklearn manual recommends that the ensemble size be restricted to 1 - also termed as vanilla auto-sklearn [9]. Running a model with no restriction on the ensemble gives us a model with recall of .025 and test set accuracy of .960 - this is the result of overfitting.

model_id	rank	ensemble_weight	type	cost	duration
3	1	0.98	gradient_boosting	0.003102	151.909420
16	2	0.02	k_nearest_neighbors	0.004387	48.710248

Figure 9: Overfit autosklearn Model

Thus, following the vanilla auto-sklearn methodology , running it for 30 minutes, we obtain *gradient boosting* as the best model, with a recall close to that obtained from our previous top-performing models, as shown in the table.

model_id	rank	ensemble_weight	type	cost	duration
3	1	1.0	gradient_boosting	0.007095	122.880877

Figure 10: Vanilla autosklearn Model

9 Conclusion and Discussion

9.1 Model Comparison

In line with Section 6, we chose to focus on Recall and MCC as our primary metrics. In this context, we conclude that the XGBoost model trained using balanced data performed the best when it comes to overall performance. This model has a recall of .803 and MCC score of .848. It must be noted that due to XGBoost's multiple optimizations described in previous sections, it had a significantly lower training time compared to models of similar complexity and/or performance. However, we see that a few neural networks, such as sgd_relu_0_10 perform with perfect recall. It must be noted that a few non-neural net ML algorithms also performed remarkably well. These include a Random Forest classifier trained on balanced data, and a Logistic Regression Classifier trained on post-PCA balanced data. The RF classifier had recall of .860 and MCC of .663, while the LR classifier had a recall of .995 and MCC of .108. It must be noted here that although the LR classifier had an extremely high recall, its MCC score was significantly lower than other models, signifying that it had trouble with other classes of observations, such as False Positives.

The ensemble created a model with a high recall as well as a higher MCC compared to the other neural nets. This shows that the ensemble method when leveraging the proper models, can be more efficient that any single classifier.

In line with intuition hyperparameter tuning saw substantial improvements in model performance.

9.2 Dimensionality Reduction

In order to save on time and capture the most important aspects of the data, PCA was done on the balanced data, keeping the first 15 eigenfeatures. Using the reduced data did not affect the efficacy of most of the models; however, when trianing tree-based models with post-PCA data, we saw significant drawbacks in performance across the board. For example, when looking at the results of the XGBoost model trained with the PCA data, the precision, recall, and F1 scores dropped by a factor of .8. This may be because the dimensionality reduction leads to the dropping of the features with the most information gain, leading to less effective models.

Interestingly, when running the PCA, one of the eigenvectors accounts for around 42 percent of the variance in the data. However, as shown in figure 5, none of the individual features can account for that much of the variance on their own. This demonstrates how PCA can generate features that despite having no realistic meaning, can explain a large portion of the data. In this case, we were able to reduce 241 features to 15, since the top 15 features accounted for a sizable part of the variance in the data.

9.3 Future Work

Due to time constraints, we were only able to tune a few hyper parameters. Future work should expand the breadth of hyper parameter search with the models.

To account for the data imbalance, we resampled the data using SMOTE. This was mostly due to the usage of SMOTE in industry, but we may benefit from using other resampling methods such as ADASYN.

Furthermore, as mentioned in the conclusion, recall in this context is a more important performance metric than precision/F1 score, however, it is important to note that precision and thus F1 score would be important to lower cost for potential governmental countermeasures. Since we trained most of our models with recall and MCC in mind, the precision of the models were not as high as desired. Future work could involve building more complex ensemble models that leverage higher precision models and higher recall models to leverage the best of both.

10 Acknowledgements

The authors would like to thank Professor Lyle Ungar of the University of Pennsylvania as well as the TAs for CIS 520 for their guidance and support throughout this semester.

References

[1] NASA approves asteroid hunting space telescope to continue development. JPL NASA. Last accessed 12 December 2021.

[2] Center for NEO studies. CNEOS. Last accessed 12 December 2021.

[3] Jatin Kataria. Detecting potentially hazardous asteroids using deep learning. Blog, 2020. Last accessed 12 December 2021.

[4] Anish Si. Hazardous asteroid classification through various machine learning techniques. *International Research Journal of Engineering and Technology (IRJET)*, 07(3):5388–5390, 2020.

[5] Hefele, John D., Bortolussi, Francesco, and Zwart, Simon Portegies. Identifying earth-impacting asteroids using an artificial neural network. *A&A*, 634:A45, 2020.

[6] Mir Sakhawat Hossain. Asteroid dataset. Kaggle, 2020. Last accessed 12 December 2021.

[7] Will Arliss. Why and how to use cross entropy. , 2020. Last accessed 12 December 2021.

[8] Dan Bruton. Conversion of absolute magnitude to diameter for minor planets. SFASU. Last accessed 12 December 2021.

[9] Vanilla auto-sklearn. automl. Last accessed 12 December 2021.