


# Hazardous Asteroid Classification

By : Peter Brown and Will Tower

---




# Problem

- Objects which impact earth, even if small, cause major damage
    - D=1km impact → serious regional loss of life
    - D=10km impact → global mass casualty events
  - X-Risk!
  - With enough early warning, we can mitigate risk
  - Goal is to detect and filter for hazardous asteroids ASAP, provide max. prep. time
- 
- 

# Challenges

- Of the over 90836 asteroids sampled, only 8,840 deemed “potentially hazardous”
  - Class split is roughly 90/10 Non-hazard/hazardous, so accuracy is a bad metric
    - a model that guesses “nonhazardous”, is right 90% of the time, and wrong every time it could matter
    - In this universe everyone dies, but we get our EoY bonus, so
  - We need to avoid overvaluing the majority class
    - Pushes us away from 0-1 loss
-

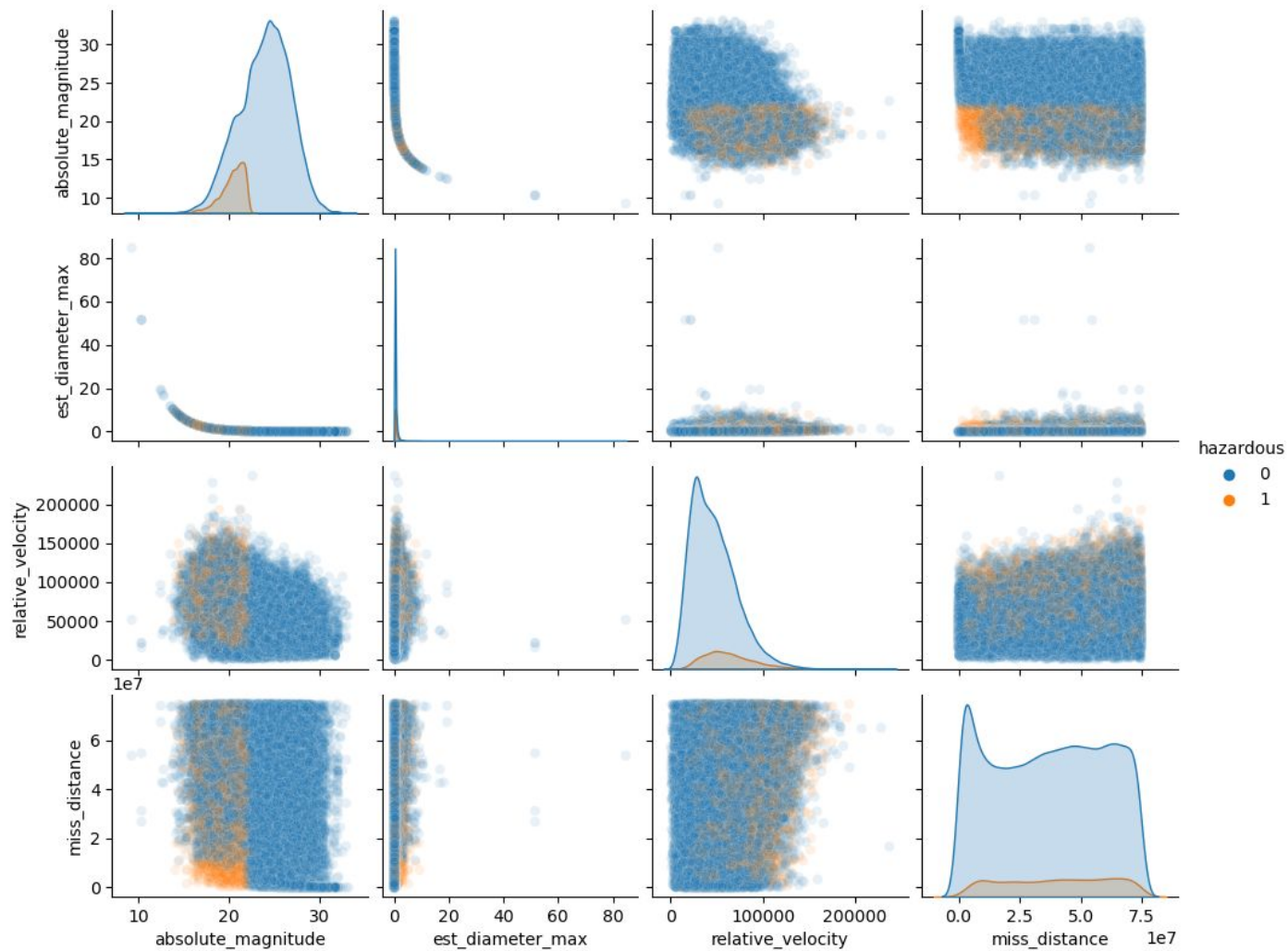
# Approach

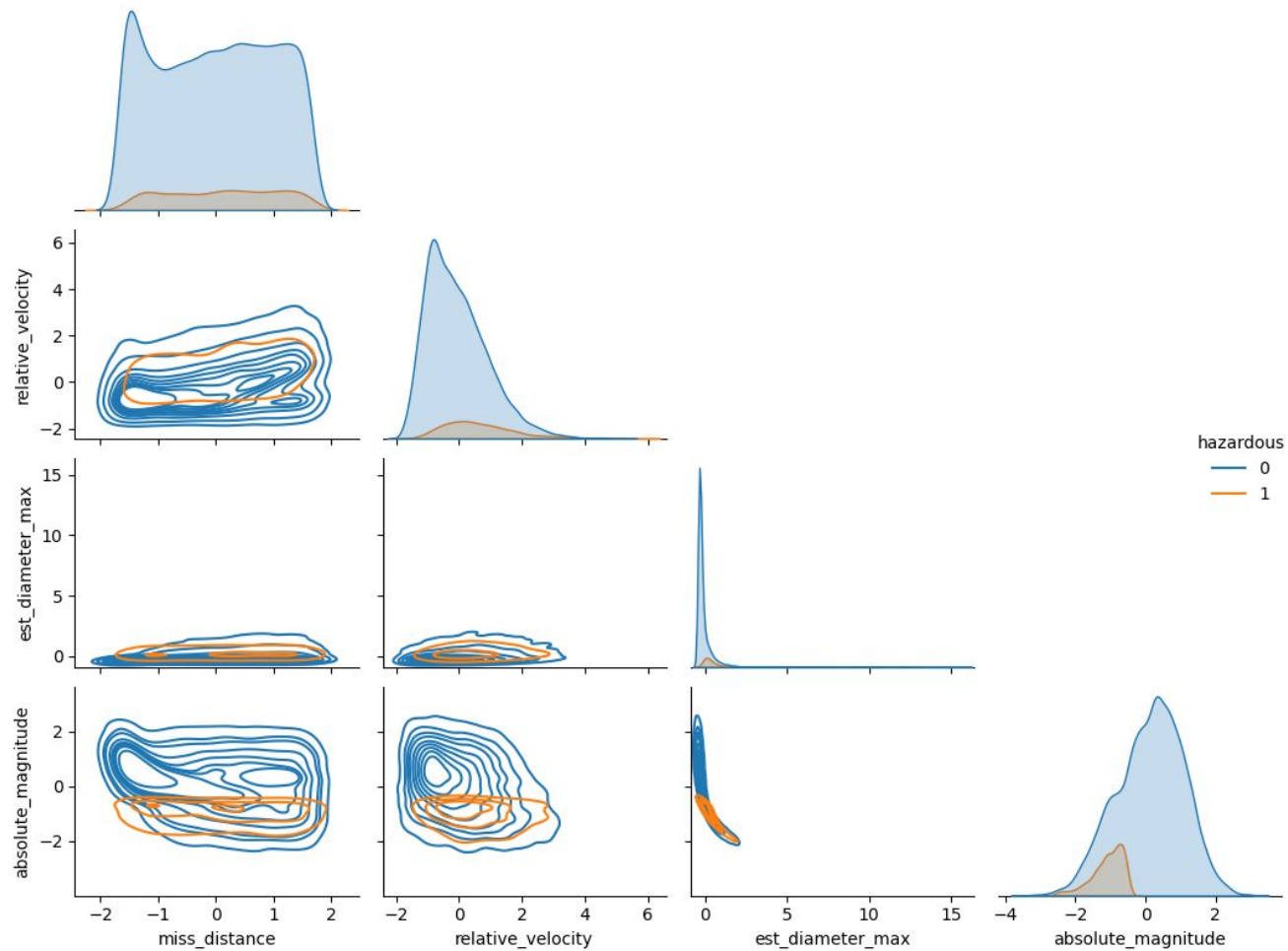
- Analyze the data, see if there are obvious trends
  - Test methods to correct unbalanced dataset
  - Design Neural Network for classification
  - Optimize to reduce False Negatives
    - Important to ensure that we do not misclassify a dangerous asteroid as a non-hazardous one. Top priority.
- 
- 

# Initial Analysis

- Some columns had no relation to the data (id, orbiting body, etc)
- After cleaning the data, there is only one strong correlation : absolute magnitude
- This is in line with space agency guidelines, of which a dangerous asteroid is classified by its magnitude







# Logistic Regression Model

We started with a Logistic Regression Model due to the speed of which they train. Due to its simplicity, we opted to start with this as a baseline

Hyperparameter Tuning:

- Class weights : Corrects class imbalance between hazard/non-hazard
- C : Regularization to reduce overfitting

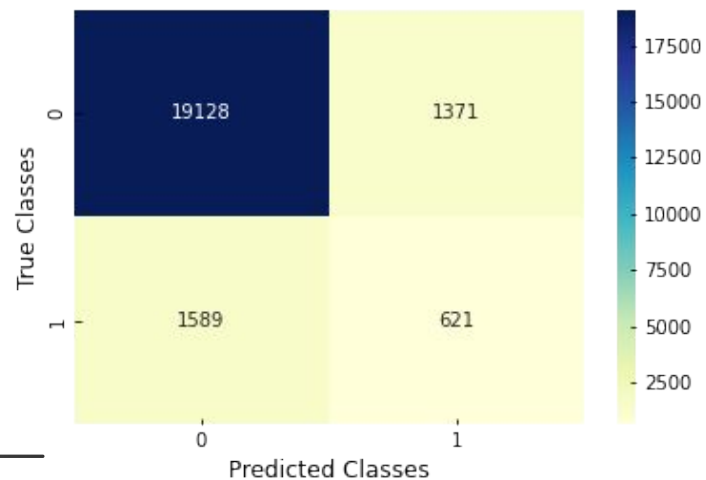
With tuning, the best parameters became:

- Class weights : { True : 20%, False : 80% }
  - C : 0.8
-



# Logistic Regression Results


	precision	recall	f1-score	support
False	0.92	0.93	0.93	20499
True	0.31	0.28	0.30	2210
accuracy			0.87	22709
macro avg	0.62	0.61	0.61	22709
weighted avg	0.86	0.87	0.87	22709



# Support Vector Machine

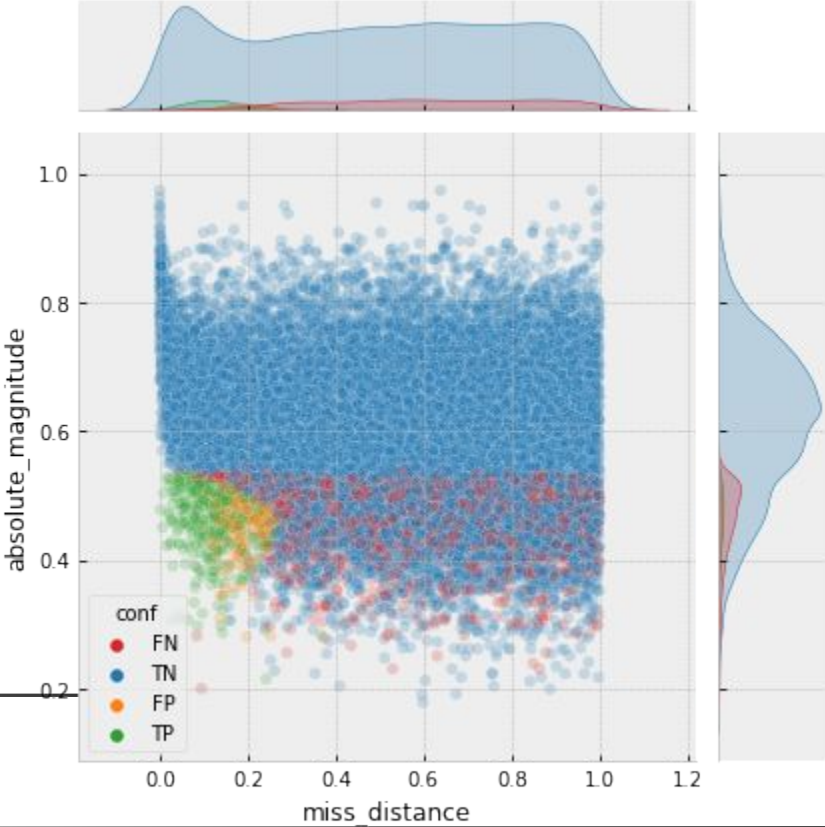
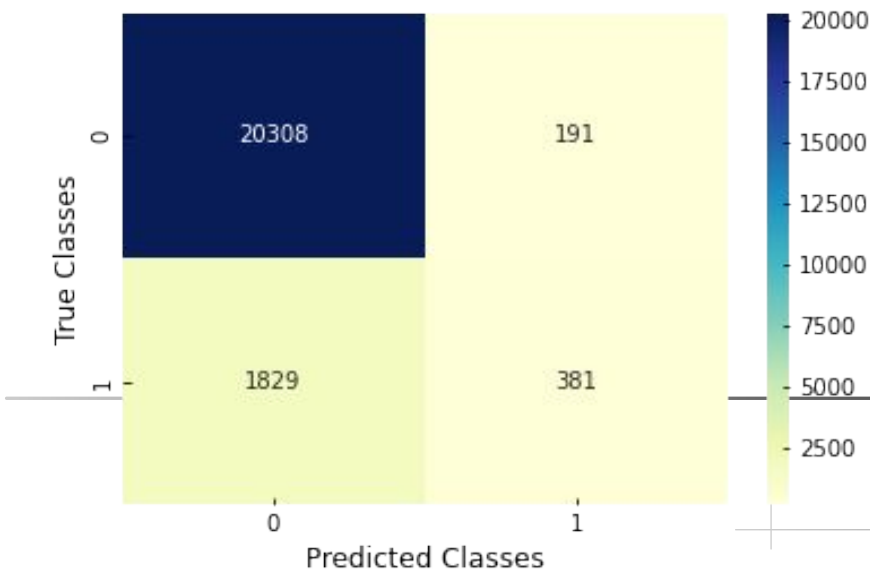
SVMs are favorable because of their fast predict times and they always converge. In addition, SVMs also perform better in edge case scenarios, which we want since there is no definite boundary between hazard/non-hazard objects.

Hyperparameter Tuning:


- Class Weights : { True : 60%, False : 40% }
  - C : 1.2
- 
- 

# SVM Results

	precision	recall	f1-score	support
False	0.92	0.99	0.95	20499
True	0.67	0.17	0.27	2210
accuracy			0.91	22709
macro avg	0.79	0.58	0.61	22709
weighted avg	0.89	0.91	0.89	22709



# Neural Network

- Our original model architecture was selected based on prior experience with binary classification tasks.
  - The metrics blocks and weight variable are derived with inspiration from [TensorFlow tutorial on imbalanced datasets](#).
  - internal layers, neurons, and learning rate were rule-of-thumb.
  - tanh activation function selected for its ability to fit complicated curves
  - optimizer selected for the excellent convergence properties of SGD with momentum and look-ahead.
- 
- 

## Loss Function

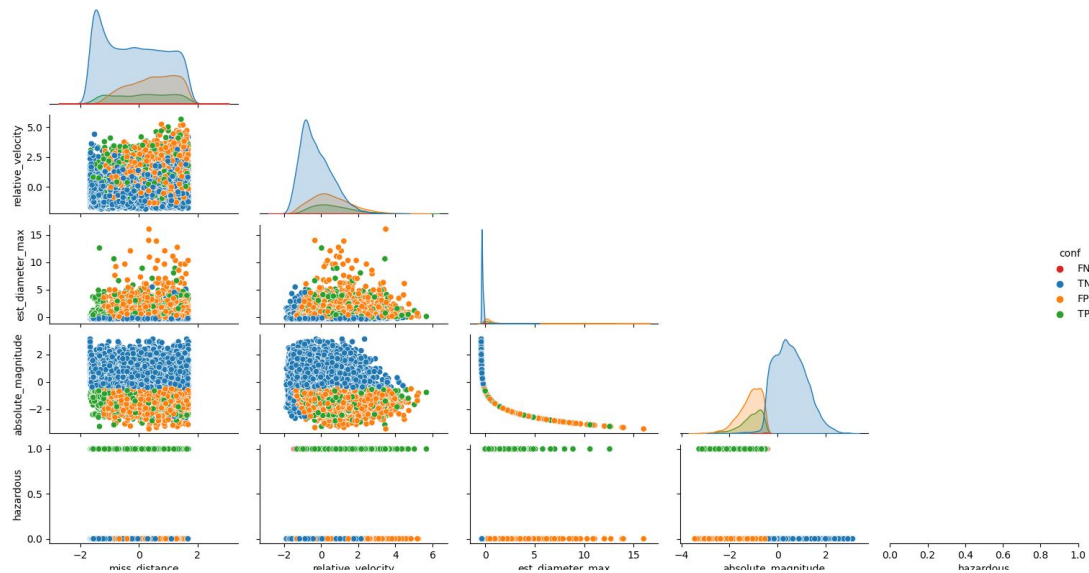
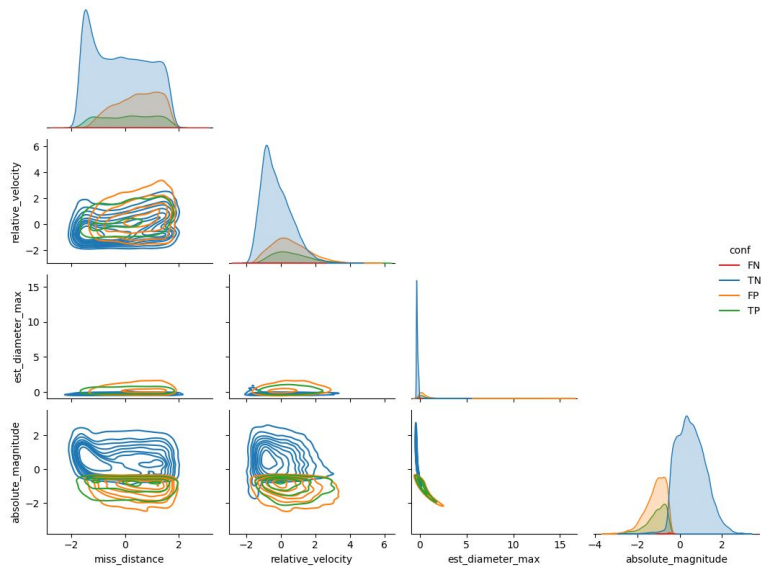
We had originally selected a binary-crossentropy loss function, but were disappointed with its performance on the dataset, even with the provided weights parameter. After some shopping around, we settled on a Binary Focal Loss metric, defined:

$$L(y, \hat{p}) = -\alpha y (1 - \hat{p})^\gamma \log(\hat{p}) - (1 - y) \hat{p}^\gamma \log(1 - \hat{p}) \quad (1)$$

where  $\gamma$  is the focusing parameter, specifying how high-confidence correct predictions contribute to the overall loss (higher  $\gamma$ , translates to down-weighted easy-to-classify samples), and  $\alpha$  is a hyperparameter that controls the precision-recall tradeoff by setting weights for errors on the positive class ( $\alpha=1$  is no weighting)

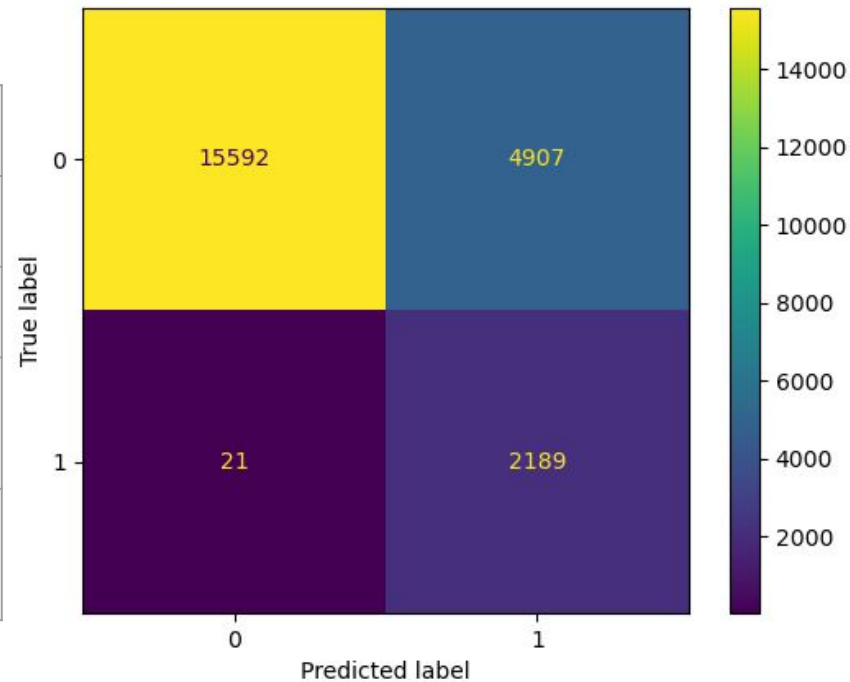
This loss allows us to focus on the hard to classify samples, and to value recall over precision, exactly what we want.

## Naive NN Performance



# Naive NN Performance

	precision	recall	f1-score	support
0	1	.77	.87	20499
1	.31	.99	.48	2210
Macro avg	.66	.88	.67	22709
Weight avg	.93	.79	.83	22709



# Keras Tuner

The Tuner works by calling a `build_model` function over a randomized search space, where the set of randomly tunable parameters is defined by various `hp.*` calls in `build_model` itself. We find the best parameters are:

```
'num_layers': 3,  
'units': 30,  
'lr': 0.009263303924328512,  
'gamma': 0.11835983615085716
```

Where `units` is synonymous with hidden layer neurons

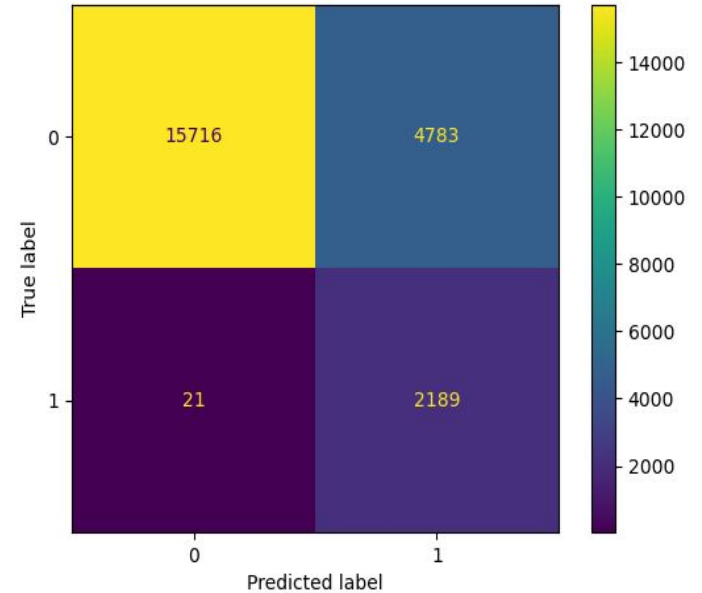
Tuner results best visualized at [tensorboard.dev](https://tensorboard.dev)

```
# Results summary  
# Results in asteroids_nn_tuning/asteroids  
# Showing 3 best trials  
# <keras_tuner.engine.objective.Objective object at 0x7f8e842c3e50>  
# Trial summary  
# Hyperparameters:  
# num_layers: 3  
# units: 30  
# lr: 0.009263303924328512  
# gamma: 0.11835983615085716  
# Score: 0.9974660515785218  
# Trial summary  
# Hyperparameters:  
# num_layers: 3  
# units: 15  
# lr: 0.0004000695122985253  
# gamma: 0.06352943477687838  
# Score: 0.9941176414489746  
# Trial summary  
# Hyperparameters:  
# num_layers: 4  
# units: 25  
# lr: 0.0004926783901540332  
# gamma: 1.6928927504819127  
# Score: 0.992941164970398  
  
# tuner.search_space_summary()  
# Search space summary  
# Default search space size: 4  
# num_layers (Int)  
# {'default': None, 'conditions': [], 'min_value': 2, 'max_value': 4, 'step': 1, 'sampling': None}  
# units (Int)  
# {'default': None, 'conditions': [], 'min_value': 15, 'max_value': 30, 'step': 5, 'sampling': None}  
# lr (Float)  
# {'default': 0.0001, 'conditions': [], 'min_value': 0.0001, 'max_value': 0.03, 'step': None, 'sampling': 'log'}  
# gamma (Float)  
# {'default': 0.05, 'conditions': [], 'min_value': 0.05, 'max_value': 5.0, 'step': None, 'sampling': 'log'}
```

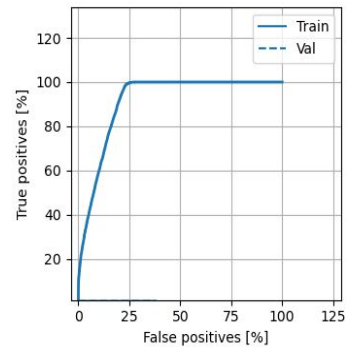
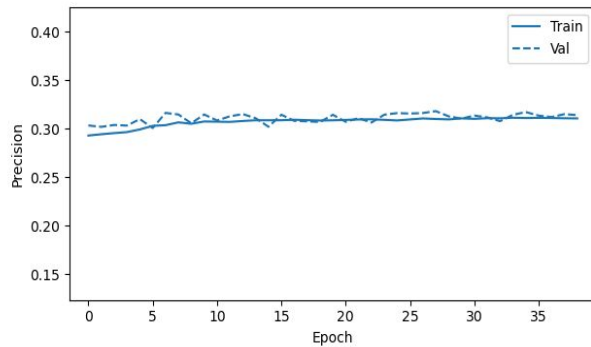
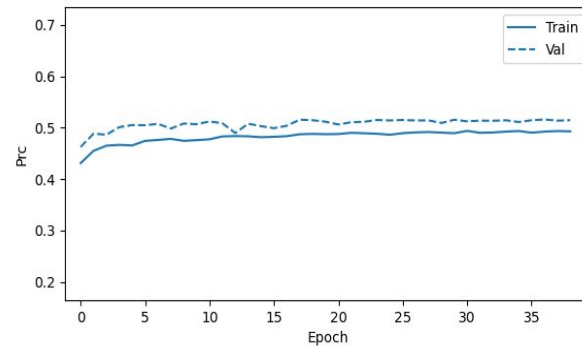
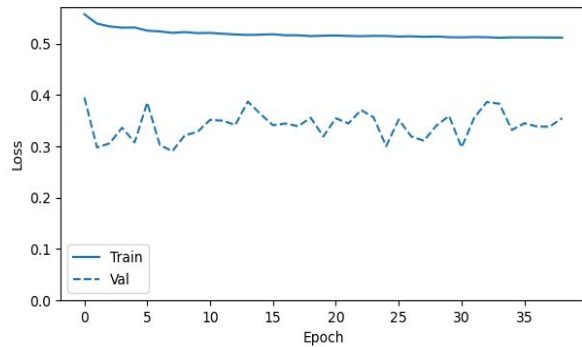


## Tuned NN Performance

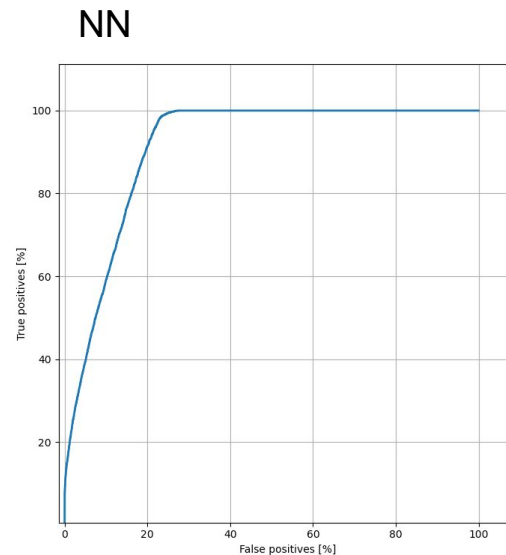
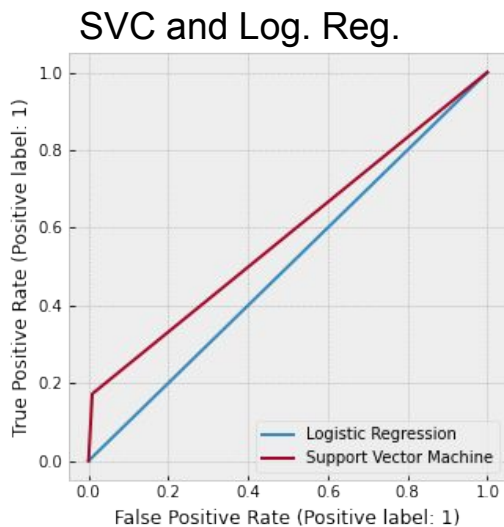
	precision	recall	f1-score	support
0	1	.77	.87	20499
1	.31	.99	.48	2210
Macro avg	.66	.88	.67	22709
Weight avg	.93	.79	.83	22709



## Tuned Model Metrics



# Comparison of Results



---

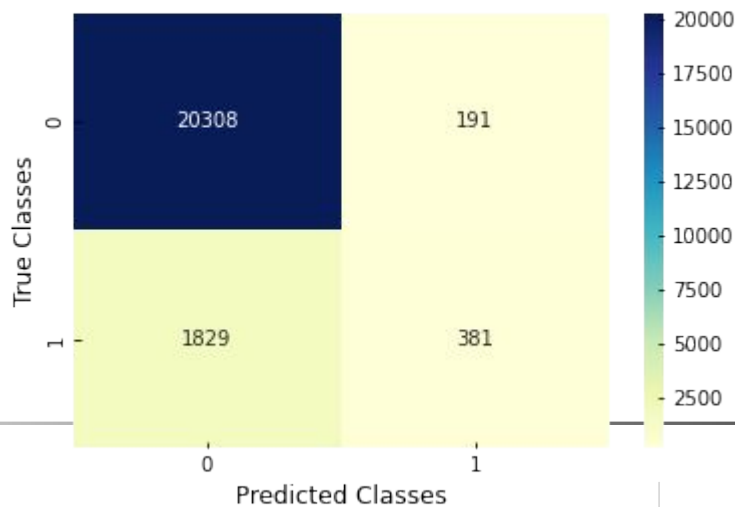
Neural Net is clear winner

# Comparison of Results

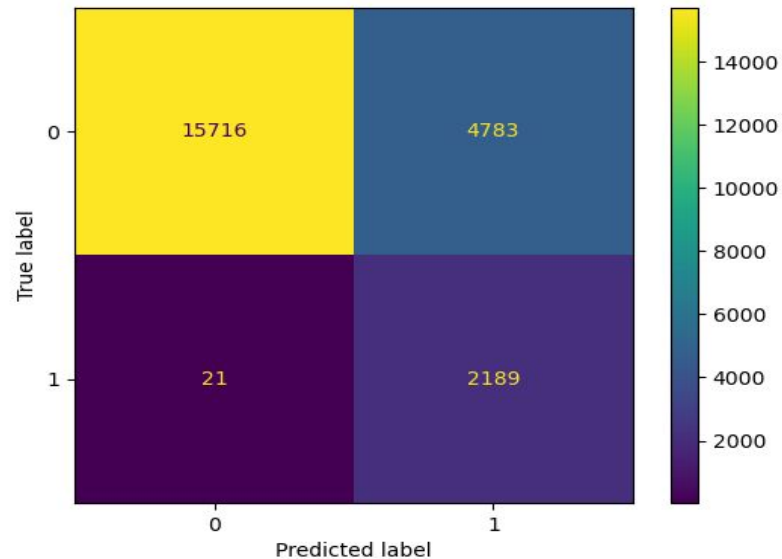
Neural net has less FN, but a higher FP than the SVC

Neural net detects more hazardous asteroids while misclassifying very few dangerous ones as well

SVC



Neural Net



# Conclusion

- Obvious from visualizations that there's not enough data to achieve perfect classification
  - Appears some classification is being made with data we don't have access to
- Special care required for a minority positive class to classify it at all
- Judging a model requires careful thought about Type I and Type II errors
- Tensorboard is pretty cool

