Satellite Track Fusion Using State Vector Classifer and Random Forest Voting Ensemble Model

CS 6840

Ryan Arnold

12/01/22

Introduction

In target tracking applications, sensor failure and sensor error can be a significant issue in data processing and object identification. A method of mitigating this is to employ sensor fusion, which improves reliability by combining the data from multiple sensors. This can overcome problems caused by noise and can offer higher accuracy and rubustness in classification. A key component of sensor fusion is the object classification step. A novel approach to the classification step is to employ machine learning. For example, Vasuhi et al. demonstrated this capability by using State Vector Machine Classifiers (SVC) [1].

I attempted to extend the approach demonstrated by Vasuhi et al., by utilizing an SVC model approach in conjunction with an ensemble model. Furthermore, I combined SVC models with Random Forest models (RF) on simulated satellite radar tracks using a voting classifier. For each sensor in my experiment, I fit a unique voting classifier ensemble model, and then combined the models together for a cumulitive prediction matrix. Using a machine learning technique like this can offer higher performance on more complex datasets than conventional techniques [1].

Dataset

Using the Python3 library *poliastro*, I simulated the orbital trajectories of three arbitrary satellites. This was performed using a Low-Earth-Orbital (LEO) model with a Keplian Orbit, using Earth as the main body. I iteratively chose the paramters until I visually observed an intersection between the three orbital paths. To stress the machine learning model, it was desirable to have a dataset with overlap to make it nontrivial to distinguish between tracks with their associated satellite objects. This was specifically configured by setting the initial velocity and position vectors for the three satellites in an inertial coordinate frame. After fitting the orbital models using the *poliastro* api, the data was exported by extracting the resulting ephemeris data (time, position, velocity). The data was also exported to *czml*, a json-like formatted file that can by read and rendered by Cesium Ion software. A snippet of the python implementation is shown below:

```
from poliastro.twobody import Orbit


orbit = Orbit.from_vectors(Earth, self.r, self.v,
    epoch=START_TIME)
...
ephem = orbit.to_ephem(strategy=EpochsArray(
    epochs=time_range(start=CROSS_START_TIME, end=CROSS_END_TIME,
    periods=NUM_TIMES
)))
...
export_cesium(<filename>)
```

The Cesium Ion Rendered depiction of the simulated dataset can be viewed in Figure 1.

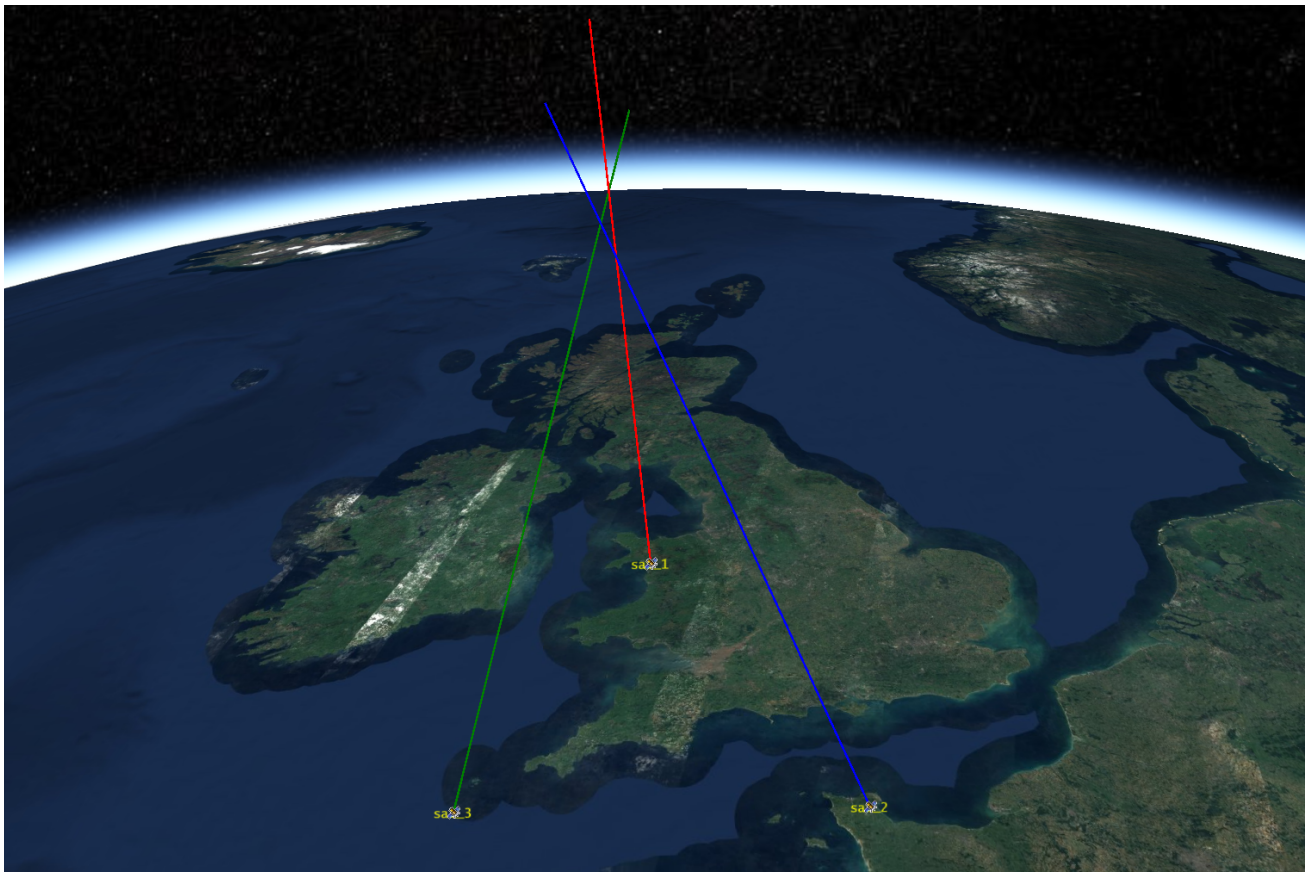

Figure 1: Track Data Rendered in Cesium Ion Server.

In this experiment, two fictitious radars were used. This was accomplisehd by setting *radar_1* to have a range error of $1km$ and *radar_2* to have a range error of $5km$. Each radar contained 1500 observations, for a grand total of 3000 observations. When examining the dataset by satellite, there were 1000 observations per satellite object. By using a random number generator, the original dataset was randomly split into two, equally sized, subsets to emulate two observation datasets from different radars. See the partitioning strategy in the example python code below:

```
ind = np.random.randint(len(RADAR_ERRORS._fields),
    size=states[0].shape)


for i, field in enumerate(RADAR_ERRORS._fields):
```

```
# assuming the states are uniformly dimensioned

epsilon: float = getattr(RADAR_ERRORS, field, 0.0)
noise = np.zeros(states[0].shape) * units.km

noise_ind = ind == i
num_err = noise_ind.sum()
noise[noise_ind] = epsilon * np.random.randn(num_err)

for state in states:
    # attempt to normalize
    state.normalize()
    state.range_km += noise
```

Before performing any machine learning algorithms, the data was pre-processed using the *pandas* python api, after consolidating the ephemeris data to a *pandas.DataFrame*. The dataset was organized by time, spatial position(x,y,z), *satellite_id*, and *radar_id*. The input feature space consisted of 3-D cartesian positions (x, y, z), with a 1-D label dimension using the *satellite_id* as the feature label. The data was partitioned into separate subsets based on *radar_id*. The results are dispayed in Figure 2 and Tables 1-3.

Table 1: Data Preview using DataFrame.head()

	satellite_id	radar_id	time_rel_s	x_km	y_km	z_km
0	sat_1	radar_1	10.000000	633.321711	-5363.930987	5014.454006
1	sat_1	radar_1	10.005005	633.127425	-5362.304441	5012.983412
2	sat_1	radar_1	10.010010	633.225240	-5363.151863	5013.825616
3	sat_1	radar_1	10.015015	633.128131	-5362.348358	5013.124427
4	sat_1	radar_2	10.020020	632.900068	-5360.435713	5011.386305

Table 2: Data Summary Statistics

	time_rel_s	x_km	y_km	z_km
count	3000.000000	3000.000000	3000.000000	3000.000000
mean	12.500000	631.079652	-5354.205352	5022.489484
std	1.445061	2.604249	8.065200	8.582994
min	10.000000	626.667610	-5374.651893	5000.146205
25%	11.250000	629.331781	-5360.419547	5015.774211
50%	12.500000	630.209829	-5354.327071	5022.429047
75%	13.750000	632.237250	-5348.121667	5029.226351
max	15.000000	639.474429	-5330.217558	5045.998444

Table 3: DataFrame Feature and Label Descriptions

#	Column	Non-Null Count	Dtype
0	satellite_id	3000 non-null	object
1	radar_id	3000 non-null	object
2	time_rel_s	3000 non-null	float64
3	x_km	3000 non-null	float64
4	y_km	3000 non-null	float64
5	z_km	3000 non-null	float64

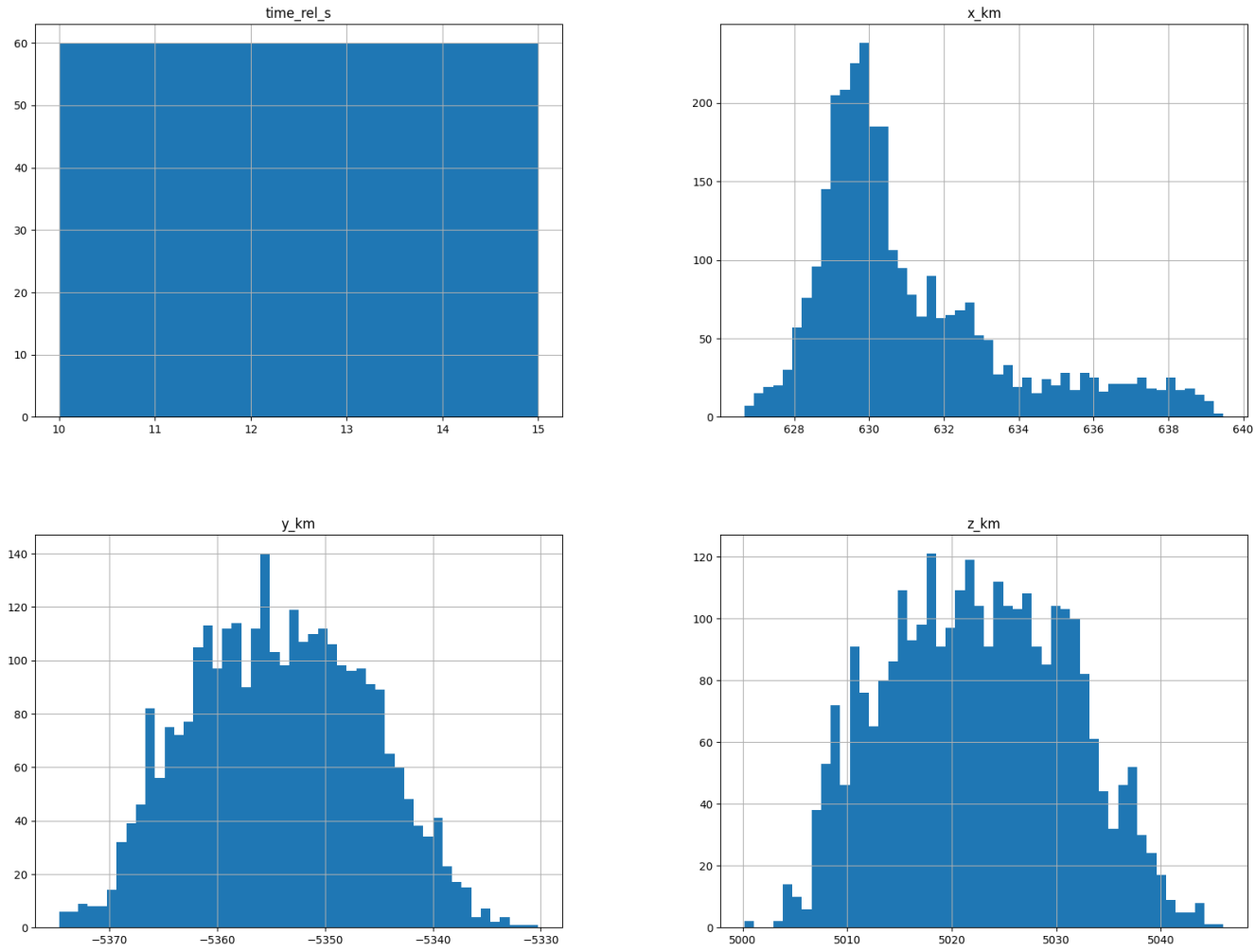

Figure 2: Histograms of input Features.

As presented in Figure 2, the datasets are mostly evenly distributed. However, the x position dataset seems to be skewed left, with a concentration in the lower component range in $[km]$. Since the range errors were generated using Python Gaussian algorithms, it was expected that the data would mostly be evenly distributed.

The plot in Figure 3 is the compilation of the two radar feature sets in one viewing space. The colors represent the different satellite objects, and the marker symbols represent the two radar groups. Note the overlapping region of data near the center. This is the critical region, where the

machine learning model decision functions become more strained.

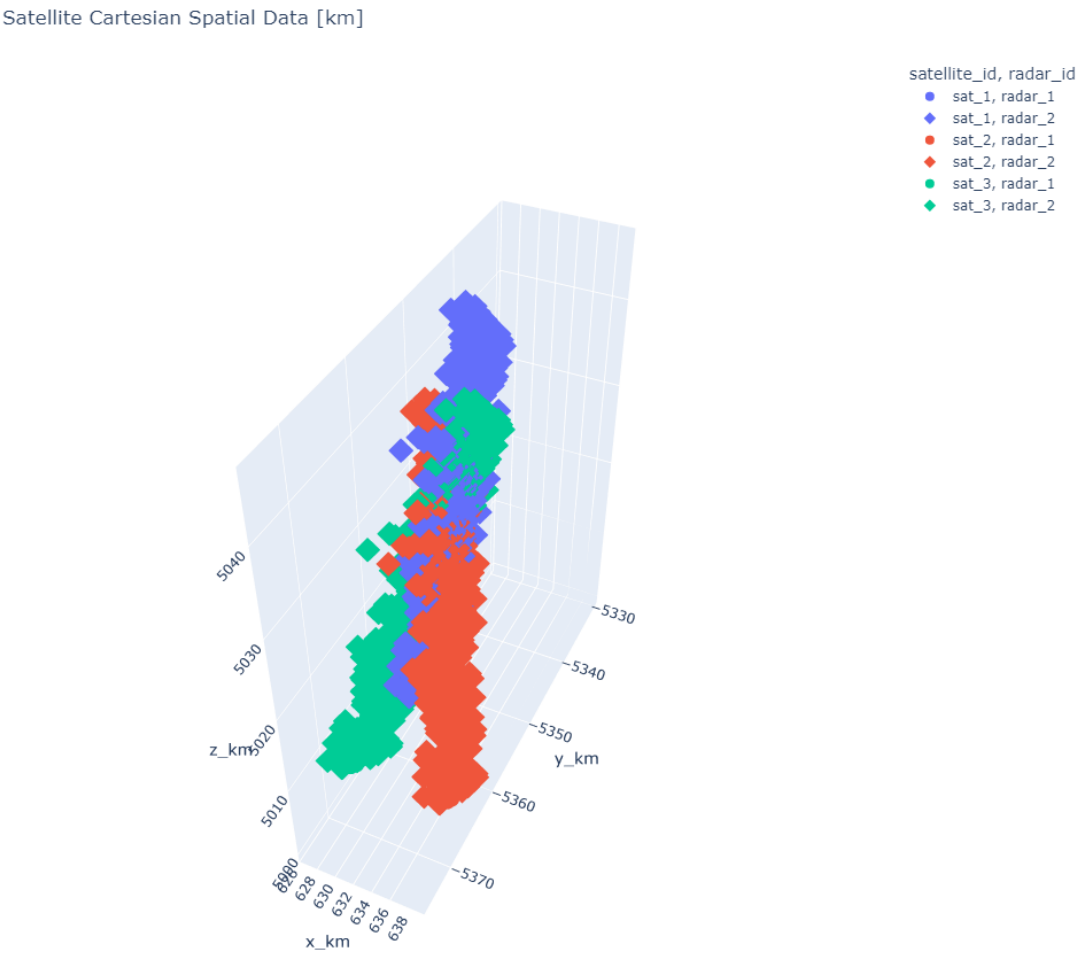

Figure 3: Combined Feature Space.

Methods

Primary Python Libararies

- astropy
- czml3
- matplotlib
- numpy
- pandas
- plotly
- poliastro
- scikit-learn
- tabulate

Methodology

The primary strategy for fusion involved finding an optimal shallow learning model for each radar, and subsequently fusing the predictions from each radar data subset. The predictions from each radar were combined in a cumulitive DataFrame, and used for the final classification results. For each radar data subset, both a RF and SVC model were applied to the training datasets. The training data and test data were split using a $80\%/20\%$ split respectively, via the *sklearn.model_selection.train_test_split()* method.

Each sub-model was hypertrained by passing hyperparameter ranges to the *sklearn.model_selection.GridSearchCV* method. This performed and exhaustive "fit" and "score" search to determine the optimal hyperparameters for each model with the provided training data. After each model was fit for a given radar subset, the optimized models were passed togehter to a vote-based bagging ensemble method: *sklearn.ensemble.VotingClassifier*.

The voting classifier object selected the best model for the subset, based on a hard vote, using the mean accuracy as the metric criterion.

After the final voting classifiers were fitted for each radar subset, the predictions were combined between the two voting classifiers to get a comprehensive prediction DataFrame. The models were assessed by using accuracy, macro F1 score, and total loss values. For the Rf models, a log-loss function was used; for the SVC models, a hinge-loss function was used. As a pre-emptive observation, the two models performed similarly. Also, considering the skewness of the X feature dimension, it was deemed necessary to run several fits and average the final results to properly assess model performance. Training and testing data splits were constructed using stratified k-fold splits via the *sklearn.model_selection.StratifiedShuffleSplit* method. A total of 10 splits were fitted, and the model fitting scores were averaged. In addition, the winnning models for each split were tabulated. Confusion matrices were also computed to better visualize the model performances.

A sample snapshot of the model code is shown below:

```
grid_search = GridSearchCV(
    svm.SVC(),
    params_grid,
    cv=5,
    n_jobs=-1,  # run jobs in parallel to speed things up
    #scoring='neg_mean_squared_error',
    return_train_score=True,
    verbose=3
)
```

```
...

# retain only the spatial data
X = subset.drop(
    columns=['satellite_id', 'radar_id', 'time_rel_s'])
y = subset.satellite_id

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=RANDOM_STATE
)

if not final_cache_filename.is_file():
    clf_svc = fit_svc(X_train, y_train,
        svc_cache_filename)
    clf_rf = fit_random_forest(X_train, y_train,
        rf_cache_filename)

    estimators = [('svc', clf_svc), ('rf', clf_rf)]
    clf = VotingClassifier(estimators, voting='hard')
    clf.fit(X_train, y_train)
```

Table 4: K-Fold Experiment Results: Radar 1 (K=10)

Radar 1	
Total Times SVC Chosen	Total Times RF Chosen
7	3

Table 5: K-Fold Experiment Results: Radar 2 (K=10)

Radar 2	
Total Times SVC Chosen	Total Times RF Chosen
2	8

Table 6: Final Model Results: **Radar 1**

Classifier	Accuracy	Loss [hinge-SVC/log-RF]	Macro F1 Score
SVC	0.970684	1.75211	0.96994
RF	0.970684	0.123589	0.969483

Table 7: Final Model Results: **Radar 2**

Classifier	Accuracy	Loss [hinge-SVC/log-RF]	Macro F1 Score
SVC	0.935374	2.22855	0.926926
RF	0.911565	0.235218	0.909554

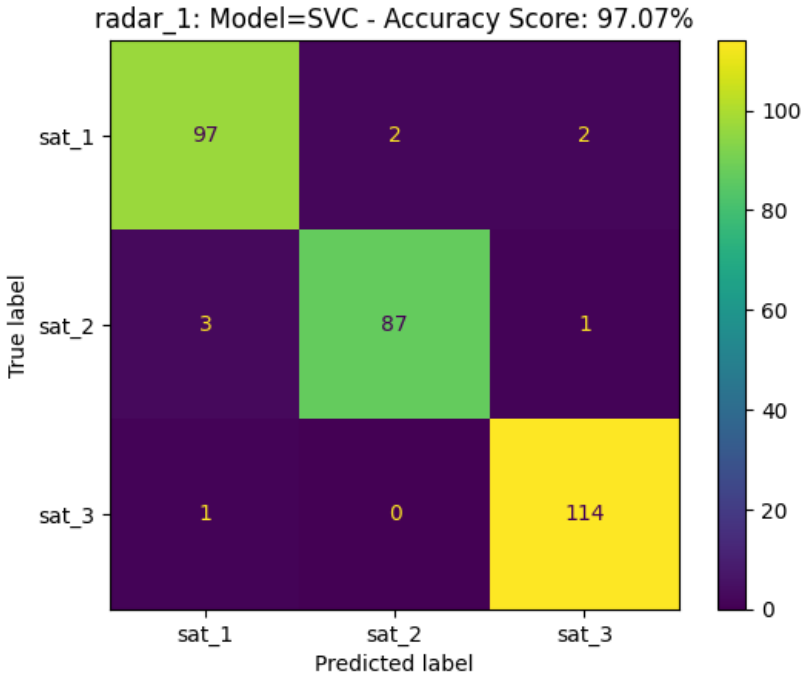

Figure 4: Confusion Matrix Ensemble Classifier Radar 1.

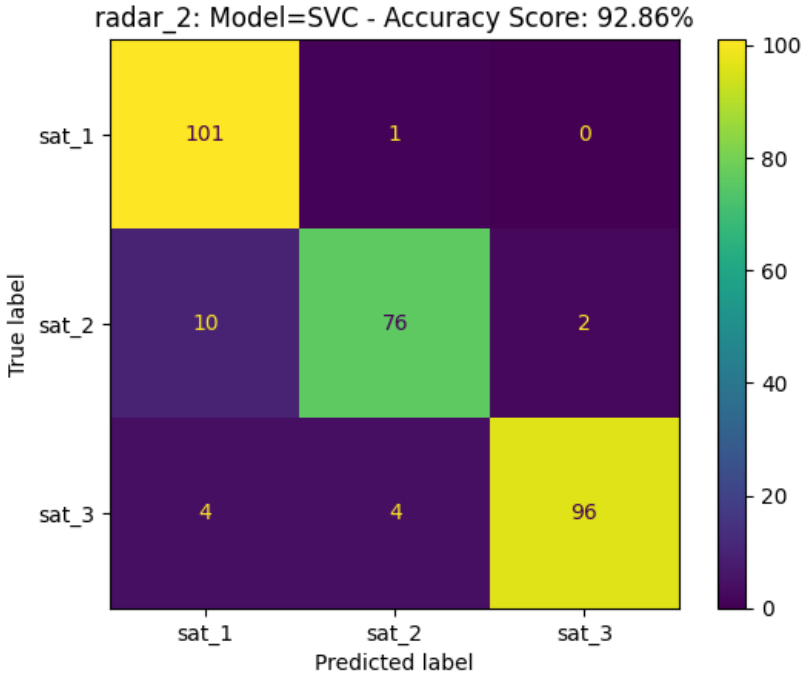

Figure 5: Confusion Matrix Ensemble Classifier Radar 2.

Experimental Results

Discussion

All of the final enesmble models had accuracies $> 90\%$, which are promising resutls. In the experiemnt using a single randomized train/test split method, the SVC model slightly outperformed the RF model. However, this was insignificantly better in the Radar 1 subset and approximately 2.4% in the Radar 2 subset (see Tables 6 and 7). Given the similarity in performance, it is more insightful to inspect the results of the k-fold analysis, where the models were fit to training data multiple times with different folds.

Interestingly, the RF model actually outperforms the SVC model a majoriy of the time with the Radar 2 subset (80%), and 30% of the time for Radar 1 (See Tables 4 and 5). The distinguising

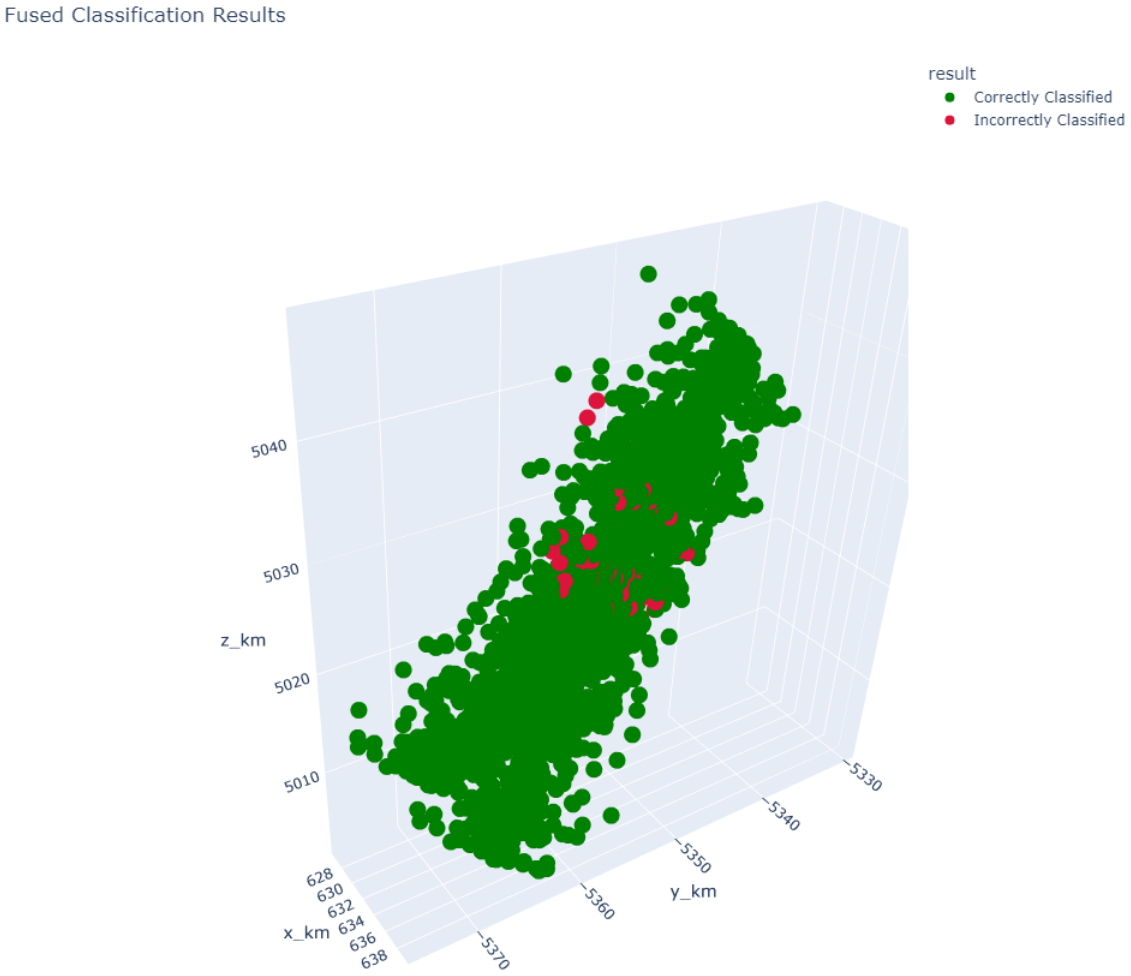

Figure 6: Confusion Matrix Ensemble Classifier Radar 2.

parameter of Radar 2 was the higher range error (five times that of Radar 1). It is possible that The RF model performs better on noiser datasets than the SVC model. This is likely due to the state vectors being more error prone in noiser environments. When there is not a clear distinciton between the feature sets, it is more difficult to establish a hyperplane that can properly spatially separate the features. Conversely, the RF model does not have such strict spatial dependence as the SVC model, so it makes sense that the noiser spatial data would be easier to separate using an RF model.

Both confusion matrices in Figures 4 and 5 had satisfactory results, with a high concentration of True Positive predictions. The only noteworthy error was for Radar 2 in Figure 5, there seemed to be a disproportionatly high amount of False Positives for satelltite 1, when the true object was satellite 2. Overall, according to the fused results displayed in Figure 6, the model seemed to have the most trouble accurately classifying the objects in the overlapping region, concentrated mostly at the center. This is indicated by the presence of red markers in this regime. This was expected, given the close proximity of the points in that region, making it difficult to calculate rigid decision boundaries. The presence of noise/error further complicates the decision boundary formation.

Challenges and Future Steps

The biggest challenge of this project computationally was the hypertraining via the Sklearn grid search implementation. When I used finer parameter ranges within the grid, the program would often get stuck on threads,m making it impossible to optimize the parameters for that resolution. Optimizing the code, using better hardware, or allowing more computation time may have improved this.

Another challenge was defining the ratio of overlapping points to non-overlapping points. Since the solution to the initial orbit parameters was done primarily throught trial and error, a more empirical solution would have been ideal. Moreover, having a proper solution could allow for an examination of the proportion of overlapping points versus non-overlapping points, and how this would affect the model fitting/selection process.

In terms of future work, better preprocessing could have been performed. For example, using techniques like PCA and data scaling, could have been implemented to better screen the data. Other models could have been tested as well, instead of exclusively RF and SVC classification models. A semi-supervised clustering algorithm could have also been insightful, since distance is an important parameter to that algorithm, and is makes intuitive sense to use as an optimization paramter for a spatial dataset. Finally, in order to create a higher fidelity model, more advanced trajectories with a higher object count and additional sensors should be tested.

Conclusion

Using the *poliastro* Python library, it was possible to simulate overlapping satellite trajectories for three satellites. To incorporate a track fusion algorithm, different radars were emulated by randomly splitting the track observations into equally sized subsets, and corrupting the data with range error noise, unique to two different radars. Overall, the ensemble model demonstrated high performance, with a mean accuracy of 97.0% and 92.9% for Radar 1 and Radar 2 respectively. Given that the Radar 2 subset had a higher simulated measurement error, it was expected that performance would be lower. Interestingly, according to the K-fold evaulation experiment, the Random Forest model outperformed the State Vector Machine Classification model for the noiser dataset associated with Radar 2. Given that the RF model has less computational overhead, this model may be more desirable in applications that require high performance (such as real-time

applications). Also, if the dataset is more prone to measurement error, the RF model could be a better option, based on the results in Table 5.

References

[1] Vasuhi, S., Vaidehi, V., Midhunkrishna, P. R. (2011). 'Multiple target tracking using Support Vector Machine and data fusion.' *3rd International Conference on Advanced Computing, ICoAC* 2011, 407–411. https://doi.org/10.1109/ICoAC.2011.6165210

[2] Paialunga Piero. "Ensemble Learning with Support Vector Machines and Decision Trees" *Towards Data Science* (13 December 2020). https://towardsdatascience.com/ensemble-learning-with-support-vector-machines-and-decision-trees-88f8a1b5f84b