MACHINE LEARNING APPLIED TO THE OPTIMIZATION OF SST SENSOR TASKING

Igone Urdampilleta Aldama[1], Daniel Lubián Arenillas[2], Ignacio Grande Olalla[2], and Fernando Pina Caballero[2]

[1]*CDTI, Calle del Cid 4, Madrid 28001, Spain, Email: igone.urdampilleta@cdti.es*
[2]*Deimos Space, Ronda de Poniente 19, Tres Cantos 28760, Spain, Email: {daniel.lubian,ignacio.grande,fernando.pina}@deimos-space.com*

ABSTRACT

In the frame of Spanish Space Surveillance and Tracking (SST) activities, and in particular, in regards to the operations of the national network of SST sensors (S3TSN), the S3TOC (Spanish operational centre for SST) is undertaking research activities aimed at optimizing the use of national resources. One of those initiatives is centred on an analysis of the performances of sensors over three years of operations, with special focus on the five tracking sensors of the total of nine optical sensors contributing to the S3TSN network.

Data since July 2016 is being analysed, with Machine Learning techniques, in order to identify the most relevant patterns within the unsuccessful tasks that have not been accomplished by the sensors. The outcome of the study will be considered for new dynamic planning of activities under evaluation in the S3TOC.

Supervised Machine Learning algorithms from neural networks to ensemble-learning-based estimators are applied to a set of data formed by the original planning requests sent by the S3TOC sensor planner, the orbital information associated to those objects (maintained by the S3TOC cataloguer or based on third party information), and the Sensor Monitoring System (SMS) deployed at S3TOC. SMS retrieves the information from the sensors daily data, including information on any relevant technical unavailability and information on weather conditions.

Over the past three years of operations, more than 1200 planning requests (a request to observe a certain object in a predefined period containing several slots) with more than 150.000 observation slots have been sent to the nine optical telescopes of the S3TSN. The overall 35% of these requests have been identified as unsuccessful tracking requests (after removing bad weather and technical issues unavailability). The study is intended to estimate the probability of a tracking request being successfully observed or not by evaluating which are the main aspects playing a role (faint objects, inaccurate orbital information, geometrical conditions, sky conditions, etc). The aim is to identify the best recommendations for future tasking to minimize the impact of such conditions.

The paper summarizes the statistical analysis carried out as a first step to understand the possible contributing features to the model together with the main findings derived from this work. Additionally, it describes the approach used in deriving the Machine Learning based estimator of a successful observation probability, comparing models like a feed-forward neural network, a random forest or adaptive boosting based on decision trees. This probability can be incorporated into the future planification activities in the S3TOC. Finally, general recommendations for SST observations, not only for the national S3TSN network will be derived.

Keywords: SST; S3TOC; EUSST; machine learning; sensor tasking; random forest.

1. INTRODUCTION

The Spanish Space Surveillance & Tracking Operational Centre (S3TOC) sends planned observation requests for each night to the S3T sensor network. The main objective is to maintain the quality of the space objects catalogue and to provide collision risk assessment as part of the European SST consortium (EUSST). These planned requests are successful if the observation is carried out by the telescope and measurements are sent back to the S3TOC to be processed. Nevertheless, the requests are not always fulfilled due to bad weather, technical issues or other reasons (known or unknown).

The objective of this activity is to find an underlying pattern for these failed observations and determine an estimation of the probability of a request being successful, using machine learning (ML) techniques. With this probability, the observation tasking process could be improved and optimized.

ML is a data-driven modelling technique [1, 2] from the field of Artificial Intelligence (AI) that has been booming in the last decade due to the digitization of the society, the availability of a multitude of data in all industrial sectors and the increase of computing capabilities. This, together with the possibility of extracting and recognizing patterns

easily thanks to programmatic techniques such as ML, makes it very interesting to import them to a sector such as space, and in particular in the area of Space Situational Awareness (SSA).

The high proliferation of space objects in orbits in the recent years and the future prospect, make necessary the application of different mitigation strategies in cooperation with an exhaustive surveillance and tracking services of these objects, as provided by EUSST. Anything that can contribute to the improvement and automatization, and make this services provided more efficient should be definitely encouraged. This includes, for example, the usage of ML techniques in some processes of the SST-chain, like sensor tasking and planning.

ML techniques are mainly based on statistics and optimization. Through this, a base algorithm automatically adjusts a multitude of parameters to a previously selected and prepared set of data. This collection and preparation is perhaps the most important and difficult part of the process. If the data used is not good enough, the model will not be able to find an underlying pattern and could not be used with new similar examples (or samples) of that data. In other words, it will not be able to 'learn' from it. Therefore, it is important to have a model as good as the dataset, with at least similar performance.

In this work, the dataset is directly provided by the S3TOC and it has been expanded with the two-line elements (TLEs) in order to increase the information available for the ML algorithms. Using the information included in these requests and knowing whether the observation request is successful or not (and the reason), a ML model is trained to solve this binary classification problem (successful or unsuccessful observation). Several supervised ML algorithms are considered [1, 2, 6] and evaluated to finally select a Random Forest (RF) classifier as the best candidate for the sensor tasking and planning activity.

This paper is organized as follows. Section 2 offers an overview of the methodology followed along the study. The results derived from the training and evaluation of the different ML algorithms are then described in Section 3. Finally, Section 4 summarizes the main conclusions and future work proposals.

2. METHODOLOGY

The first step in the methodology of working in the field of ML is to define the problem properly. In this case, the aim is to classify an observation request into two classes (possibly successful or unsuccessful). The problem is then a binary classification, and the number provided by the algorithm as output will be the probability that an observation request is successful or unsuccessful.

Given the data-driven nature of ML, the second step once the problem has been defined is to get the data needed to solve it. This data shall be prepared, filtered and expanded, when it is possible. For example, it should be evaluated the generation of new information (or *features*) from the raw dataset, which can influence the problem to be solved. Another activity could be the elimination of those samples that are easily distinguishable as erroneous or uninformative, like outliers.

Subsequently, we proceed to select the algorithm or algorithms that can be used to solve the problem. There are algorithms specialized in the treatment of images, time series or a vector of numbers. Moreover, the metric used to evaluate the goodness of the model depends on the type of problem, since a regression problem (with a continuous output) is not the same as a classification problem (discrete output).

In the final step, these ML algorithms are trained (fit) with a part of the dataset (Training set and Test set), while a smaller part (Validation set) is reserved for the evaluation of the model, as there are usually many hyperparameters to optimize in the algorithm, see Fig. 1.

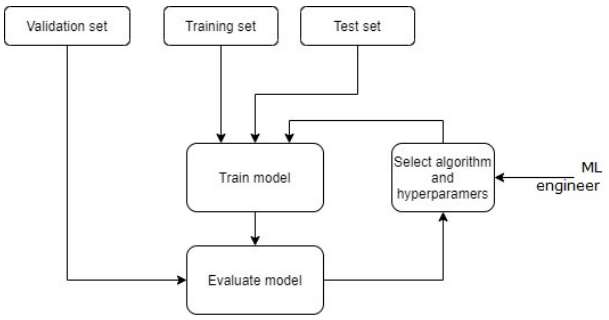


Figure 1. Activity diagram of the Model Trainer and Evaluator component.

2.1. Dataset

The initial dataset contains 1200 observation requests with more than 150.000 observation slots from the six sensors of the S3STN (TJO, TRACKER, IAC80 and BOOTES network: B2-ESP, B3-NZL, B5-MEX). From each slot the target object, time slot, sensor and sensor location information have been derived for a time period between May 2017 and June 2020. If the observation request is unsuccessful, the reason is codified in four classes: 'bad weather', 'technical issues', 'other' and 'unknown'. This is shown in both Fig. 2 and Fig. 3.

By means of TLE sets, this initial dataset is expanded by adding more features like the classical orbital elements, descriptors of the right ascension, declination, azimuth, elevation, and features related to background illumination or the magnitude of the target object.

The list of features evaluated in this study is as follows:

- `sensor_id`: sensor identifier.

- `sensor_lat`: latitude of the location where the sensor is located in degrees.
- `sensor_lon`: longitude of the location where the sensor is located in degrees.
- `sensor_alt`: altitude of the location where the sensor is located in meters.
- `slot_duration`: duration of the observation slot.
- `target_sma`: semi-major axis of the target object in kilometres.
- `target_inc`: inclination of the target object in degrees.
- `target_raan`: right ascension of the ascending node of the target object in degrees.
- `target_ecc`: eccentricity of the target object.
- `target_aop`: argument of the perigee of the target object in degrees.
- `night_fraction`: fraction of the night, from 0 to 1 (0 is sunset and 1 is sunrise). This feature would show if an observation was taken early in the night or at the end, when there is more background light noise.
- `observer_sun_sat_angle_max`: maximum solar angle in degrees.
- `observer_sun_sat_angle_min`: minimum solar angle in degrees.
- `ang_distance_moon_min`: minimum angular (degrees) distance to the moon during the predicted track.
- `moon_phase`: phase of the moon between 0 and 1 (new to full moon).
- `elev_min`: minimum elevation during the track in degrees.
- `elev_max`: maximum elevation during the track in degrees.
- `az_min`: minimum azimuth during the track in degrees.
- `az_max`: maximum azimuth during the track in degrees.
- `ra_min`: minimum right ascension during the track in degrees.
- `ra_max`: maximum right ascension during the track in degrees.
- `dec_min`: minimum declination during the track in degrees.
- `dec_max`: maximum declination during the track in degrees.

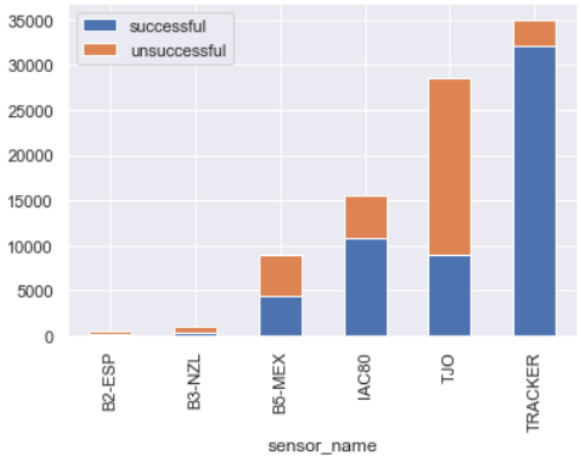


Figure 2. Distribution of successful/unsuccessful request samples per sensor (absolute values).

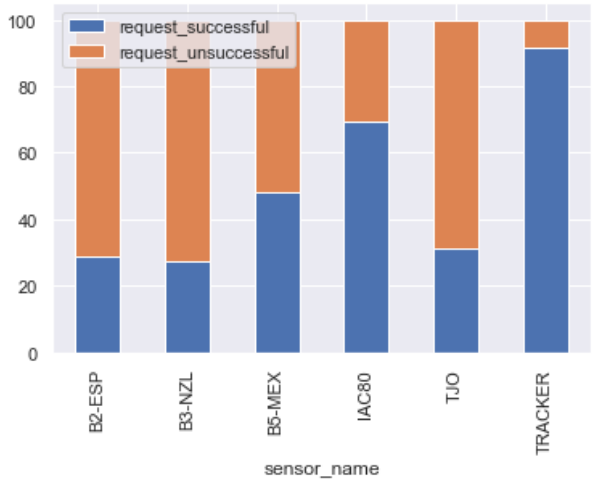


Figure 3. Distribution of successful/unsuccessful request samples per sensor (relative to total observation per sensor).

2.2. Machine Learning models and algorithms

Since this is basically a binary classification problem, different models suitable for this problem [1, 2, 6] are considered:

- Multilayer Perceptron (`MLPClassifier`): the most basic kind of Feed-forward Neural Network (NN).
- Decision Tree (`DecisionTreeClassifier`): predicts the value of a target variable by learning simple decision rules inferred from the data features.
- Random Forest (hereafter RF, `RandomForestClassifier`): using a set of decision trees, each built from a sample drawn with replacement and at each node the best split

is found using information of a subset of features. Then, the prediction of all trees is averaged for the result of the ensemble. This technique is called ensemble learning.

- Extremely Randomized Trees (`ExtraTreesClassifier`): similar to RF, but the splits are selected from the best of a set of random splits, increasing the randomness of the process.
- Logistic Regression (`LogisticRegression`): consists on performing a regression for a binary variable using a logistic function.
- Linear model fitted with Stochastic Gradient Descent (SGD, `SGDClassifier`): fits a linear model using the SGD optimiser.
- Gradient Boosting (`CatBoost`, `XGBoost` and `LightGBM`): gradient boosting on decision trees, from the `catboost` [12], `xgboost` [11] and `lightgbm` [10] libraries. Similar to RF, but each decision tree of the ensemble is added and trained in conjunction to the previously trained decision trees.
- Adaptive Boosting (`AdaBoost`): adaptive boosting on an ensemble of decision trees. Similar to RF, but instead of averaging all decision trees, it is a weighted sum that is learned during the training process.

Each model has its advantages and disadvantages. For example, simpler algorithms like Logistic Regression or the Decision Tree might not be able to capture the complexity of the problem if the pattern is not very clear. Another example is that the Extremely Randomized Trees algorithm is, in principle, less prone to high variance and bias than the RF. On the other hand, RF has the advantage of reducing the bias, although it can have an increased variance compared to other ensemble learning methods [3].

Half of the selected models listed above rely on a technique called *ensemble learning*. This technique consists of a set of 'weak' learners, which are trained together and their results are considered at the same time to create a 'strong' learner. An example of this kind of weak learner is a Decision Tree, which is a very simple model (a set of if-else rules whose thresholds are computed automatically). Depending on how a set of these Decision Trees is trained (rules of node splitting, voting, etc.), different *ensemble learning* algorithms arise.

The implementation of these models is done in the `scikit-learn` Python library [6, 7], the standard library for Machine Learning in Python, except where otherwise specified.

2.3. Training and evaluation process

The aforementioned models are going to be tested, or trained with default parameters and basically no tuning

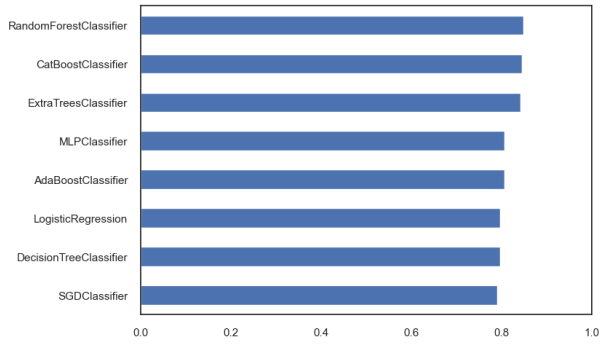


Figure 4. ML model score with default parameters considering features directly related to the sensors

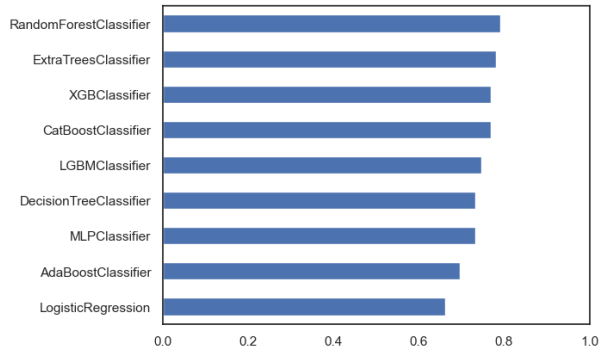


Figure 5. ML model score with default parameters removing features directly related to the sensors

of hyperparameters, in two exercises. First one, with a dataset including the features that identify directly a sensor (sensor ID, sensor coordinates and observation slot duration) and second one, removing the features related with the sensor characterization. The results are shown in Fig. 4 and Fig. 5. These figures contain the score of the model to perform the classification of the test set against known results. The way to compute the score differs from model to model, but the library manages to define it in a way that it is comparable between them. The closer it is to 1, the better the accuracy of the model is. The results are better, in average, for the first exercise, since it includes information from the sensor. Removing that information decreases the accuracy of the model since there is a direct link between the sensor identifier (or any other feature directly related to the sensor like the location) and the distribution between successful/unsuccessful requests, as it can be seen in Fig. 2.

The comparison between Fig. 4 and Fig. 5 confirm that the dataset is unbalanced regarding the sensor information. Additionally, the distribution of successful/unsuccessful samples differs among them for the different sensors, as shown in Fig. 3. Therefore, including features that directly link those difference regarding to the sensor characteristics would hide underlying patterns. For example, TJO has a large number of unsuccessful requests due to contractual differences and operational framework. The most basic

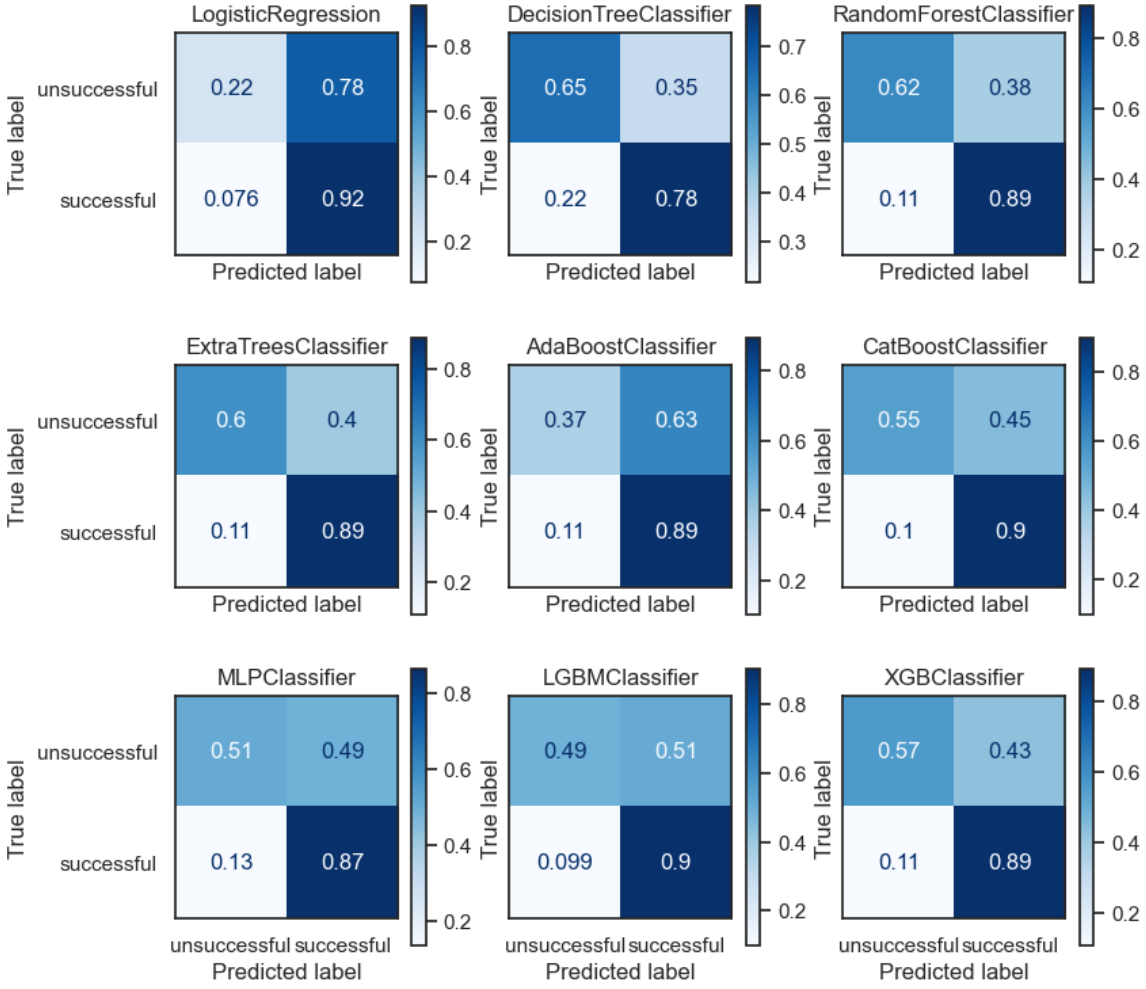


Figure 6. Confusion matrix for different models when no sensor features are considered. It shows the number of TN (top left), FP (top right), TP (bottom right) and FN (bottom left).

(and dummy) model that predicts the probability of a request being successful for TJO would say that it is 30%. Removing sensor-related features reduces the possibility of this happening, and would force the appearance of underlying patterns. For all these reasons the sensor-related features are neglected and only sensor-agnostic features are used instead in the present work. Apart from the obvious sensor-related features like the sensor identifier or its location, the slot duration was also dismissed because some sensor had smaller values which would have created a direct link to the sensor.

The Confusion Matrix (see Fig. 6) is an additional tool for evaluating the performance of classification models. The figure shows clockwise the number of true negatives (TN), false positives (FP), true positives (TP) and false negatives (FN), adimensionalized with respect to the total number of true samples per category. The TP measures the proportion of positives (successful observations) that are correctly identified. Otherwise, the TN measures the rate of negatives (unsuccessful observations) that are correctly identified. The rule of thumb proves that the more diagonal the matrix is, the better is the classification.

3. RESULTS

The initial results of the ML models evaluation based on Confusion Matrices (see Fig. 6) present a TP rate close to 90% for most of the models. However, the TN rate covers a large variety of values from 22% (Logistic Regression) to 65% (Decision Tree), confirming that, at first, not all the models can estimate the TN with enough accuracy. Additionally, in general all the models have a high rate of FP, between 30–60%, which means that a lot of requests are being flagged as successful when it is known that they are not. This value can be still improved and it is important to reduce it as much as possible, although it is preferable to a high rate of FN.

For both cases (with and without sensor information), ensemble-learning algorithms based on trees (Decision Tree, RF and Extremely Randomized Tree) seem to perform better. Amongst them, the RF algorithm is the most promising one due to its good performance and accuracy. Thus, it is selected for further investigation. Another model considered for the next step is the Feed-forward NN, because its potential capabilities, flexibility and their different principles compared to other algorithms.

3.1. Feed-forward Neural Network

Neural Networks are the basis of the ML field known as Deep Learning [5]. Since the multi-layer perceptron seemed to work relatively well, the Feed-forward NN is selected for a more precise training and evaluation phase. The Feed-forward NN is a generalization of the multi-layer perceptron.

However, `scikit-learn` is a specialized ML library but does not have the capabilities for NN models. For this reason, it is necessary to make the leap to another Python library designed and developed specifically for the Deep Learning field, i.e. NN. The selected library is Tensorflow, from Google [9]. This library includes several optimizers, layers and evaluation functions for all kinds of NN.

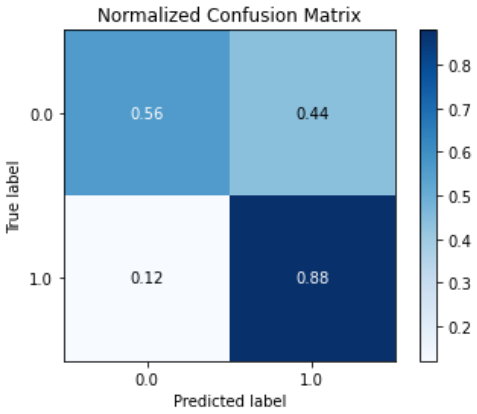


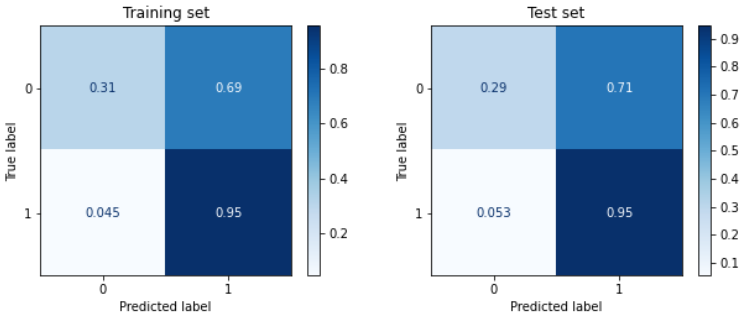
Figure 7. Confusion Matrix for the NN found by autokeras.

As part of this phase, several architectures are tested to find the optimal one by using the `autokeras` library [13] on top of Tensorflow. This way one can automatically tune the hyperparameters, improve the architecture and performance of the NN (see Fig. 7). After several trails with the optimal architecture, the results obtained in all the cases show lower performances than the RF algorithm (see Fig. 10). Moreover, since the SW development effort using a library like Tensorflow is very high (it is computationally expensive), this model is ultimately discarded. Nevertheless, further analysis is highly recommended, since the NN is expected to provide better results. Some future upgrades could include the improvement the hyperparameters tuning, the trial of different scaling methods or the application of alternative regularization techniques as neuron dropping, among others.

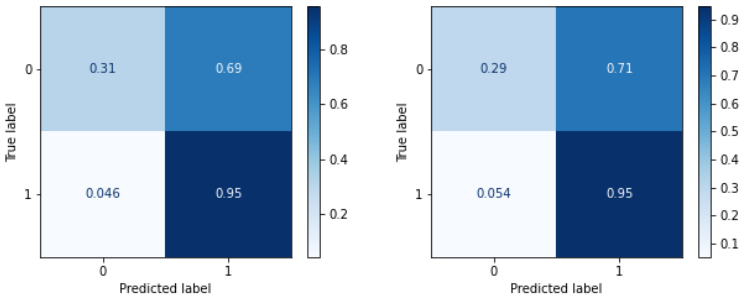
3.2. Random forest

The implementation of the RF algorithm [4] is provided by `scikit-learn`. This algorithm presents the best performance among the ones tested during the first selection. After an initial analysis, the hyperparameters used for the first training, make the model overfit the Training set. The Confusion Matrix for the Training set shows that the TP rate is 1, which means a 100% of success, whereas the Confusion Matrix for the Test set shows a TP rate of 0.89. This difference appears in the under or overfit estimators.

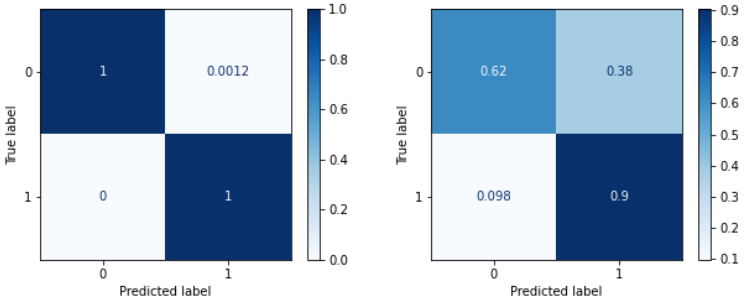
Three hyperparameters are tuned to reduce the overfitting: the number of estimators (`n_estimators`), the maximum depth of the decision trees (`max_depth`) and the percentage of the total number of samples to consider for node splitting (`max_samples`). The training



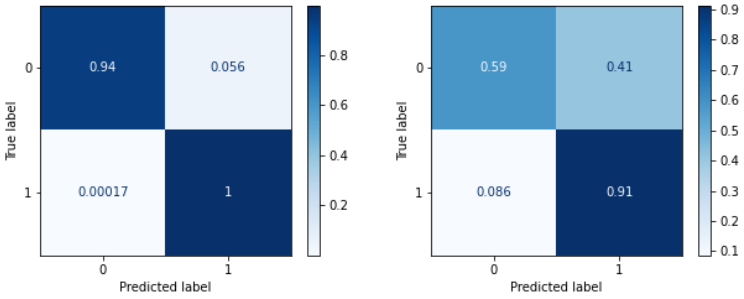
MODEL 0. Score = 0.703
boostrap = True
ccp_alpha = 0.0
class_weight = None
criterion = gini
max_depth = 8
max_features = auto
max_leaf_nodes = None
max_samples = None
min_impurity_decrease = 0.0
min_impurity_split = None
min_samples_leaf = 1
min_samples_split = 2
min_weight_fraction_leaf = 0.0
n_estimators = 100
n_jobs = 8
oob_score = False
random_state = None
verbose = 0
warm_start = False



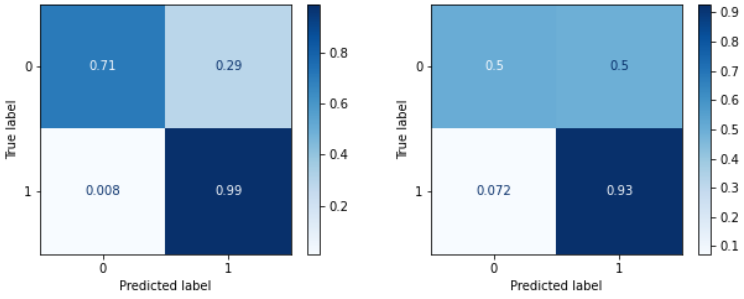
MODEL 1. Score = 0.703
n_estimators = 130



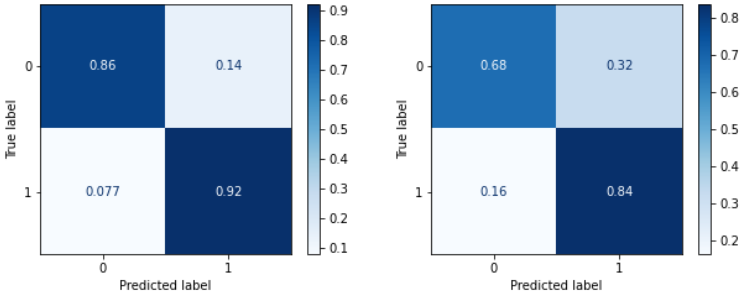
MODEL 2. Score = 0.798
n_estimators = 200



MODEL 3. Score = 0.792
max_depth = 20



MODEL 4. Score = 0.771
max_depth = 15



MODEL 5. Score = 0.779
ccp_alpha = 1e-05
class_weight = balanced
n_estimars = 150

score and the cross-validation score should be similar in value when no under or overfitting is present. See Fig. 9 for the validation curves of these hyperparameters, with `n_estimators`= 100, `max_depth`= None and `max_samples`= 1 as default values.

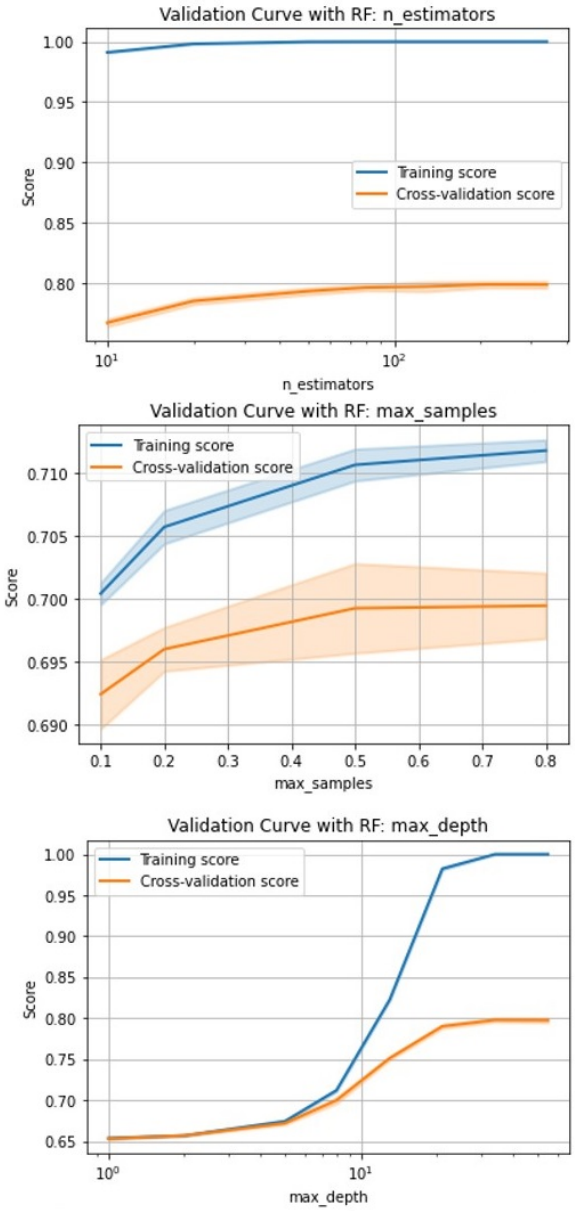


Figure 9. Cross-validation curves for the RF model. The blue and orange lines show the training and cross-validation scores, respectively. The shaded area represent the variability in the 5-fold cross validation set.

Several combinations of these values are selected and tested further, see the hyperparameters for the five different models in Fig. 3.2, *Right*, and their Confusion Matrices in Fig. 3.2, *Left* and *Middle*. When comparing the Confusion Matrix for the Training set and the Test set, it is possible to see that overfitting happens in most scenarios, so a compromise solution has to be taken. Overfitting happens when the model is fitted to the Training set and is not able to generalize in the same way to the Test set (or new data, eventually). Looking to the figures (Fig. 3.2), this means that the model would perform significantly worse in the Test set when compared with the performance on the Training set. Therefore, when both matrices are close, the model is less overfitted. With that in mind, Model 5 (the more fine-tuned) is the best candidate and is selected as final RF model (Fig. 10). It has additional variation of hyperparameters (see Tab. 1) that improves the general performance of the model. As a result, 84% of successful observations (TP rate) and 68% of unsuccessful observation (TN rate) are truly identified and classified. The rate of FP in the Test set is not negligible: 32%. This proves a remarkable good performance for an initial analysis, even though, the score is still not completely optimized, and it can be improved in future studies.

Table 1. Hyperparameters for the final RF model.

Hyperparameter	Value
`bootstrap`	True
`ccp_alpha`	1.00E-05
`class_weight`	balanced
`criterion`	gini
`max_depth`	15
`max_features`	auto
`max_leaf_nodes`	None
`max_samples`	None
`min_impurity_decrease`	0
`min_impurity_split`	None
`min_samples_leaf`	2
`min_samples_split`	10
`min_weight_fraction_leaf`	0
`n_estimators`	150
`n_jobs`	8
`oob_score`	False
`random_state`	None
`verbose`	0
`warm_start`	False

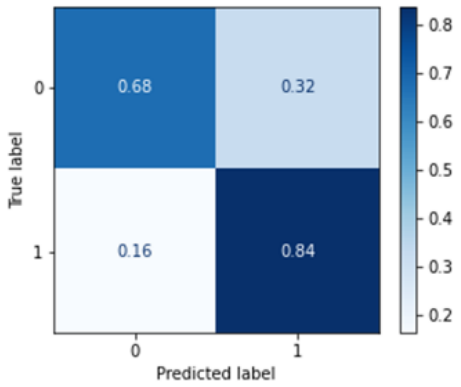


Figure 10. Confusion Matrix for the final RF model.

To evaluate the goodness of the classifier, Tab. 2 provides useful metrics (Precision, Recall and F_1). The closer are these values to 1, the better they are. The Precision

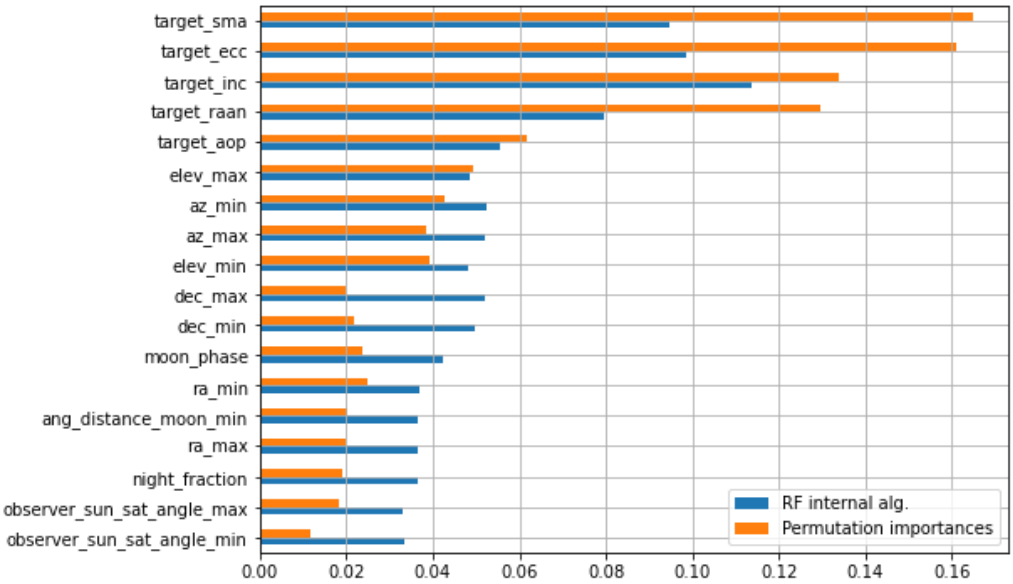


Figure 11. Features importance for the internal RF algorithm (blue) and the permutation importance algorithm (orange).

measures the ability of the classifier not to label as positive a sample that is negative. It is also known as positive predictive value. The Recall or Sensitivity is, intuitively, the ability of the classifier to find all the positive samples. It can be viewed as the probability that a successful request is labelled as such. The Recall values for successful and unsuccessful match the diagonal of the confusion matrix shown in Fig. 10. The F_1 score can be interpreted as a weighted average of the Precision and Recall, where an F_1 score reaches its best value at 1 and worst score at 0.

$$\text{Precision} = \frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}} \quad (1)$$

$$\text{Recall} = \frac{\sum \text{True positive}}{\sum \text{Condition positive}} \quad (2)$$

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3)$$

Table 2. Classification metrics for the final RF model.

	Precision	Recall	F_1
Unsuccessful	0.71	0.68	0.70
Successful	0.82	0.84	0.83
Accuracy	-	-	0.78
Macro average	0.76	0.76	0.76
Weighted average	0.78	0.78	0.78

The results of Tab. 2 show a Precision, Recall and F_1 higher than 0.8 for successful observation request and around 0.7 for unsuccessful, with an accuracy score of $\sim$80%. These values, representative of the binary classification metrics, confirm the very good behaviour of the RF model, already explained above.

One of the most interesting capabilities of the tree-based models is its explainability [14]. In other words, it is possible to compute directly from the model the features importance as the weight of each feature in predicting the probability of the sample. In Fig. 11 these values are shown, which are computed with two algorithms: the internal RF algorithm (blue), and the permutation importance algorithm (orange).

The first one relies on the criterion used when splitting new branches and leaves, and the number of samples it involves. This is based on the mean decrease in impurity during the training phase. Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labelled if it was randomly labelled according to the distribution of labels in the subset. The impurity-based feature importance of RFs suffers from being computed on statistics derived from the Training set: the importance can be high even for features that are not predictive of the target variable, as long as the model has the capacity to use them to overfit.

The second one is algorithm-agnostic since it relies on permuting features to assess the impact of that operation in the prediction. The permutation feature importance is defined to be the decrease in a model score when a single feature value is randomly shuffled. This solves the limitations of the RF internal algorithm as it is completely independent on the training phase.

Once the sensor-related information is removed, the main feature patterns can be recovered. Fig. 11 reveals that the most relevant features, which directly affects in the probability of success, are the classical orbital elements. Mainly the ones related to the size and the shape of the orbit: the semi-major axis and the eccentricity. This is probably due to the presence of a high number of GEO and GTO objects in the dataset.

4. CONCLUSION AND FUTURE WORK

The present work focuses on a ML model training with S3TOC data to predict if an observation request might be successful or not. Once the binary problem definition has completed, the next step has consisted of the dataset preparation, filtering and expansion. In the final step, an exhaustive evaluation of different ML models and algorithms has been performed concluding that the Random Forest is the most suitable for the optimization of sensor tasking. The RF shows a precision of 80% and a successful positive rate of 84%. These results present a remarkable performance of this model for a first study.

One of the main finding derived of this ML model development is related to the S3TOC provided dataset. There are important and relevant differences between the samples depending on each sensor. This means that a model considering directly features related to the sensor could give more weight to the more represented sensor in the sample, hiding underlying patterns. For this reason, sensor-agnostic (no sensor-related) features have been selected. This study has additionally revealed that the classical orbital elements are the features with higher influence in the probability of success calculation.

Future work will include an improvement in the dataset towards a more equally balanced sensor-wise dataset. Additionally, the hyperparameters tunning and the application of a dimensionality reduction algorithm will be revisited. Finally, the integration of the probability in the planification process of the S3TOC chain will be evaluated.

ACKNOWLEDGMENTS

The EU SST activities have received funding from the European Union programmes, notably from the Horizon 2020 research and innovation programme under grant agreements No 760459 and No 952852.

Disclaimer: The content of this paper reflects only the view of the SST Cooperation and the European Commission and the Research Executive Agency are not responsible for any use that may be made of the information it contains.

REFERENCES

1. Bishop CM., (2006). *Pattern Recognition and Machine Learning*, Springer
2. Hastie T., Tibshirani R., and Friedman J., (2008). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer, 2n ed.
3. Zhou Z.-H., (2012). *Ensemble Methods: Foundations and Algorithms*, Chapman & Hall
4. Breiman L., (2001). Random Forests, *Machine Learning*, **45**, 5–32. DOI:10.1023/A:1010933404324
5. Goodfellow I., Bengio Y. and Courville A., (2016). *Deep Learning*, MIT Press
6. Pedregosa F., Varoquaux G., Gramfort, A., Michel, V. et al, (2011), Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research*, **12**, 2825–2830
7. Scikit-learn documentation (https://scikit-learn.org/), version 0.23
8. Keras documentation (https://keras.io/), version 2.4
9. Tensorflow documentation (https://www.tensorflow.org/), version 2.3
10. LightGBM documentation (https://lightgbm.readthedocs.io/), version 3.0
11. XGBoost documentation (https://xgboost.readthedocs.io/), version 1.2.1
12. CatBoost documentation (https://catboost.ai/), version 0.24
13. Autokeras documentation (https://autokeras.com/), version 1.0
14. Murdoch W.J., Singh C., Kumbier K., Abbasi-Asl R., Yu B, (2019) Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences of the United States of America* 116(44):22071-22080. DOI:10.1073/pnas.1900654116