MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIONES

PREDICTIVE AND DESCRIPTIVE LEARNING

FINAL REPORT

Developing Deep Learning Models

Case Study: Collision Avoidance Challenge

Sergio Pérez Morillo
Jaime Pérez Sánchez
Carlos Andres Ramiro

Course 2019/20

Table of Contents

1. Problem Description

In this section, we will introduce the chosen topic for the development of this work: *Satellite Collision Avoidance*. Furthermore, we will justify the use of Deep Learning-based approaches to try to solve it, and we will briefly explain which methods are currently used to solve this problem

1.1. Satellite Collision Avoidance

Today, a typical satellite in Low Earth Orbit (LEO) reports hundreds of danger alerts every week for close encounters with other space objects (such as satellites, space debris, etc.). According to estimations done by the European Space Agency (ESA) in January 2019 **[1]**, more than 34,000 objects larger than 10 cm are orbiting our planet. Of these, 22,300 are tracked and their position released in a globally shared catalog. In Figure 1 we can observe the representation of the spatial density of objects in the LEO orbits. After processing and filtering the collected data, the most potentially dangerous encounters must be followed up in detail by an analyst with expert knowledge in the field. On average, more than one collision avoidance maneuver is performed on each satellite every year.

In order to automate this process as much as possible, the ESA launched a competition **[2]** in October 2019 where participants were tasked to **create a collision risk prediction model** between a given satellite and a space object. The competition and the data to be used were published on *Kelvin*[1] platform, a website created by ESA's Advanced Concepts Team in 2015. This platform focuses on space-related competitions so that enthusiasts around the world can actively participate in the advancement of space science.

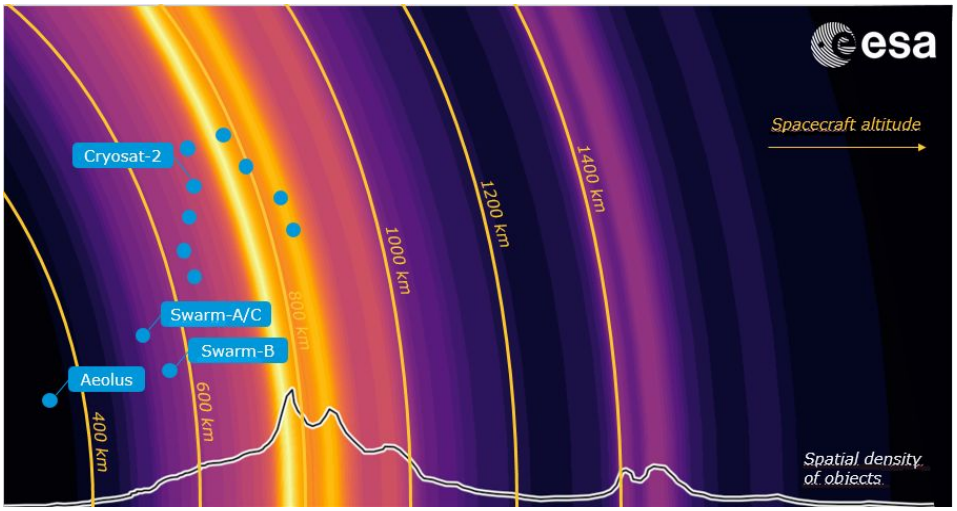

Figure 1. - Spatial density of objects in LEO

Concerning the provided data in the competition, when a potential close approach with any object is detected, the gathered data is assembled in a Conjunction Data Message (CDM). Each CDM contains multiple attributes about the approach, the time of closest approach (TCA), the identity of the satellite, the potential collider object type, etc. It also contains a self-reported risk, computed using some of the CDM's features. Typically, 3 CDMs are

[1] https://kelvins.esa.int/

reported each day (covering one week) for each possible close approach. Thus, we have a time series of CDMs for each event.

1.2. Deep Learning Motivation

For the development of the predictive models, it has been decided to use Deep Learning (DL) methods with Artificial Neural Networks. The main reasons why this has been chosen are:

- The topic of the chosen competition is of high complexity and probably with very non-linear relationships between variables. In addition, none of the team members has advanced knowledge of astrophysics or astrodynamics. Therefore the choice of a complex approach such as DL seems appropriate.
- The information provided by the organizers of the competition indicates that the training dataset is made up of more than 150,000 rows of 13,154 unique events, and 103 columns with different features. This is enough data (and dimensions) for the correct functioning of DL models.
- As explained previously, the data samples are organized as multivariate time series for each potential collision event, with a variable time step. Artificial Neural Networks (especially Recurrent NNs) are the most typical approach to this type of problem because the more classical methods of time series forecasting use a single variable and fixed period.
- Due to the type of data being handled in this competition, it is likely that some part is exposed to noise or missing values. Artificial Neural Networks and DL, in general, is much more robust than the classic ML models to face these problems.

Although the great promises of the application of Deep Learning to this problem, it is important to note that its use also involves significant drawbacks.

- DL models belong to the *Black Box* group of models, due to the high difficulty in interpreting their results. In cases of use like the one proposed by the competition, there would be considerable difficulties for the deployment in production of this type of models, since interpretability is important when there are so much money and effort involved.
- High complexity to train the model adequately, choose the network topology and tune the hyperparameters (especially in Recurrent NNs).
- Related to the previous point, this high complexity implies a very high computational and temporal cost. Thus, we will not be able to dive too deep into the search of the optimal neuronal model.

Despite the above-mentioned disadvantages, we consider Deep Learning to be a promising approach to this complex problem, due to the great potential advantages it offers.

1.3. Competition Characteristics

As explained previously, this work is based on a competition proposed by the ESA. Two data files (.csv) are provided, one for the training phase and other for the test phase. In this work,

we have only used the training file since the test solutions have not been released, which makes it practically unusable in this context. The training dataset contains **162,634 rows** belonging to 13,154 unique events. In addition, **103 columns** with different features. These features are mostly related to the position, speed and movement of both objects (satellite and tracked object).

In most real cases the Space Debris Office alerts the control teams 2 days before the closest approach of a potential collision, in order to have enough time to consider possible avoidance maneuvers. Therefore the ESA determined that the **goal of the competition** is to build a prediction model that makes use of the data collected **up to 2 days before the closest approach**, for predicting the **final risk**.

The competition ended on December 16, 2019, where our team achieved 19th place in the leaderboard. Although at that time we had not finished developing the models that had offered the best results, so, with high probability now we would be in a higher position if the competition was still active.

1.4. Literature Review

The challenge we face belongs to a very specific domain and the data provided had not been released until now. Hence the only published works addressing this issue are those carried out by ESA itself. In 2017 a paper **[4]** was published explaining the current collision avoidance system used by ESA's Space Debris Office. In it is explained that the analysis of CDMs is done automatically with specific software (created by experts in astrophysics and astrodynamics), to calculate the probability of collision. The events that show high risk are assigned to specialized teams to consider the appropriate maneuvering decisions.

1.5. Methodology

To accomplish this project we have synthesized the work process in 6 steps. These steps are not only sequential, since, as we will see later, we have approached the problem from several different perspectives that had their specific necessities. Hence the steps described below are more similar to an iterative process.

1. **Data Description:** This is the first common step for all the proposed perspectives. In it, an attempt has been made to understand the nature of the data, the possible problems or difficulties involved in it, check the type of features available, etc. In short, to understand the data we have.
2. **Data Wrangling:** Before training the Deep Learning algorithms we have to make sure that the data is correctly transformed and cleaned so that it can be interpreted by the models. Besides, for each type of proposed approach, it has been necessary to transform the data in a specific way.
3. **Exploratory Data Analysis:** In this step we have attempted to extract the main characteristics and correlations of the dataset (typically with visualizations), to be able to formulate preliminary hypotheses that can lead us in the creation of the DL models and new transformations of the data.

4. **Feature Selection:** Due to the large number of features provided in the dataset, it will be important to check which of them provides more information to the model. In this step, we select these important features according to different criteria. By doing this we also avoid introducing noise into the DL models thus helping to their stability and reliability.
5. **Feature Engineering:** In this step, new features are created to be introduced into the model to help you better solve the problem. The new features can be obtained by brute force (by making mathematical transformations) or from the knowledge that developers have about the data.
6. **Deep Learning Modeling:** In this last step we create the DL models, train them, test them and tune their hyperparameters to achieve better results.

2. Data Manipulation

In this section, we will describe the data itself and its main characteristics, as well as the transformations and techniques used to solve the problems encountered and to feed the DL models.

2.1. Data Description

As it was introduced in the previous section, the organizers of the competition provided two different datasets (for training and testing). However, we will only use the training because the test targets have not been released. The training dataset contains **162,634 rows** belonging to 13,154 unique events. Giving on average about 12 rows per potential collision. Each row corresponds to a single CDM, and each CDM contains **103 features** related to the objects' spatial positions and velocities, that will be described below.

The dataset is made of several unique potential collision events identified by the *event_id* column, and each event is made of several CDMs recorded over time. Therefore, each potential collision event can be thought of as a time series of CDMs. For the column description, we will first explain those with a unique name and then those whose name difference depends on whether they are referring to a characteristic of the satellite (columns that start with a **t**) or the chaser object (columns that start with a **c**). For easier explanation, the shared name features will begin with an **x**. All variables are numerical except *event_id*, *mission_id* and *c_object_type*. All information about the features has been taken from the competition website[2].

2.1.1. Uniquely Named Columns

- **risk:** Self-computed value at each CDM on base 10 log. The last value at each unique event, i.e. time-of-closest approach (tca) is the target to be predicted by the models. For the evaluation metric used, an event is considered of **high risk** when its last recorded risk is greater than -6, and of **low risk** when it is less than or equal to -6. Its possible values range from -30 (minimum risk) to 0 (maximum risk).
- **event_id**: Unique id per potential collision event.
- **time_to_tca**: Time interval between CDM creation and time-of-closest approach [days]. Its possible values range from 7 days (maximum time to collision) to 0 days (minimum time to collision).
- **mission_id**: Identifier of mission that will be affected.
- **max_risk_estimate**: Maximum collision probability obtained by scaling combined covariance.
- **max_risk_scaling**: Scaling factor used to compute maximum collision probability.
- **miss_distance**: Relative position between chaser & target at tca [m].
- **relative_speed**: Relative speed between chaser & target at tca [m/s].
- **relative_position_n**: Relative position between chaser & target: normal (cross-track) [m].
- **relative_position_r**: Relative position between chaser & target: radial [m].

[2] https://kelvins.esa.int/collision-avoidance-challenge/data/

- **relative_position_t**: Relative position between chaser & target: transverse (along-track) [m].
- **relative_velocity_n**: Relative velocity between chaser & target: normal (cross-track) [m/s].
- **relative_velocity_r**: Relative velocity between chaser & target: radial [m/s].
- **relative_velocity_t**: Relative velocity between chaser & target: transverse (along-track) [m/s].
- **c_object_type**: Object type which is at collision risk with satellite.
- **geocentric_latitude**: Latitude of conjunction point [deg].
- **azimuth**: Relative velocity vector: azimuth angle [deg].
- **elevation**: Relative velocity vector: elevation angle [deg].
- **F10**: 10.7 cm radio flux index [10^{-22} W/(m^2 Hz)].
- **AP**: Daily planetary geomagnetic amplitude index.
- **F3M**: 81-day running mean of F10.7 (over 3 solar rotations) [10^{-22} W/(m^2 Hz)].
- **SSN**: Wolf sunspot number.

2.1.2. Shared Column Names Between the Charser and the Satellite

- **x_sigma_rdot**: Covariance; radial velocity standard deviation (sigma) [m/s].
- **x_sigma_n**: Covariance; (cross-track) position standard deviation (sigma) [m].
- **x_cn_r**: Covariance; correlation of normal (cross-track) position vs radial position.
- **x_cn_t**: Covariance; correlation of normal (cross-track) position vs transverse (along-track) position.
- **x_cndot_n**: Covariance; correlation of normal (cross-track) velocity vs normal (cross-track) position.
- **x_sigma_ndot**: Covariance; normal (cross-track) velocity standard deviation (sigma) [m/s].
- **x_cndot_r**: Covariance; correlation of normal (cross-track) velocity vs radial position.
- **x_cndot_rdot**: Covariance; correlation of normal (cross-track) velocity vs radial velocity.
- **x_cndot_t**: Covariance; correlation of normal (cross-track) velocity vs transverse (along-track) position.
- **x_cndot_tdot**: Covariance; correlation of normal (cross-track) velocity vs transverse (along-track) velocity.
- **x_sigma_r**: Covariance; radial position standard deviation (sigma) [m].
- **x_ct_r**: Covariance; correlation of transverse (along-track) position vs radial position.
- **x_sigma_t**: Covariance; transverse (along-track) position standard deviation (sigma) [m].
- **x_ctdot_n**: Covariance; correlation of transverse (along-track) velocity vs normal (cross-track) position.
- **x_crdot_n**: Covariance; correlation of radial velocity vs normal (cross-track) position.
- **x_crdot_t**: Covariance; correlation of radial velocity vs transverse (along-track) position.
- **x_crdot_r**: Covariance; correlation of radial velocity vs radial position.
- **x_ctdot_r**: Covariance; correlation of transverse (along-track) velocity vs radial position.

- **x_ctdot_rdot**: Covariance; correlation of transverse (along-track) velocity vs radial velocity.
- **x_ctdot_t**: Covariance; correlation of transverse (along-track) velocity vs transverse. (along-track) position.
- **x_sigma_tdot**: Covariance; transverse (along-track) velocity standard deviation (sigma) [m/s].
- **x_position_covariance_det**: Determinant of covariance (~volume).
- **x_cd_area_over_mass**: Ballistic coefficient [m^2/kg].
- **x_cr_area_over_mass**: Solar radiation coefficient . A/m (ballistic coefficient equivalent).
- **x_h_apo**: Apogee (-R_{earth}) [km].
- **x_h_per**: Perigee (-R_{earth})[km].
- **x_ecc**: Eccentricity.
- **x_j2k_inc**: Inclination [deg].
- **x_j2k_sma**: Semi-major axis [km].
- **x_sedr**: Energy dissipation rate [W/kg].
- **x_span**: Size used by the collision risk computation algorithm (minimum 2 m diameter assumed for the chaser) [m].
- **x_rcs_estimate**: Radar cross-sectional area [m^2].
- **x_actual_od_span**: Actual length of update interval for orbit determination [days].
- **x_obs_available**: Number of observations available for orbit determination (per CDM).
- **x_obs_used**: Number of observations used for orbit determination (per CDM).
- **x_recommended_od_span**: Recommended length of update interval for orbit determination [days].
- **x_residuals_accepted**: Orbit determination residuals.
- **x_time_lastob_end**: End of the time interval in days (with respect to the CDM creation epoch) of the last accepted observation used in the orbit determination.
- **x_time_lastob_start**: Start of the time in days (with respect to the CDM creation epoch) of the last accepted observation used in the orbit determination.
- **x_weighted_rms**: Root-mean-square in least-squares orbit determination.

2.2. Problems Encountered

Throughout the study of the dataset we have encountered a large number of problems that needed to be solved for the DL models to perform properly. The main problem is the imbalance of the high risk and low risk classes. The main problem is the imbalance of the high risk and low risk classes. In Figure 2 we can observe the histogram distribution of the risk variable. In it we can clearly notice that the high risk events are a very small percentage of the total. This is a very common problem in real world cases, and very important for the models to behave in the desired way.

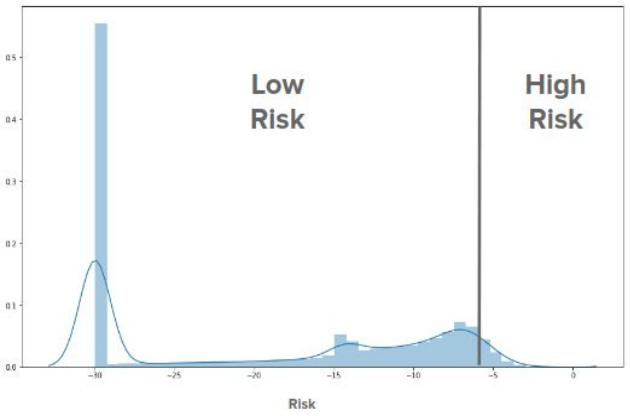

Figure 2 - Histogram of the feature *risk*

A problem related to interpreting each potential collision event as a time series is that, typically, a time series has a fixed sample period. However, our dataset (where time is interpreted as the feature *time_to_tca*) has a variable period and with real values, not integers. An example of a sequence to appreciate this problem is shown in Figure 3.

In addition, another important issue related to this is that the length of each event sequence is not always the same. Varying between 21 and 2 samples per event. This is especially important for recurrent models such as those based on LSTM neurons since they are designed for fixed-size input dimensions. In Figure 4 we can observe the histogram of the lengths of the event sequences.

event_id	time_to_tca
4	4.966244
4	4.030424
4	3.066467
4	1.797727
4	1.528456
4	1.258629
4	0.973420
4	0.592587
4	0.273166

Figure 3 - Example of event sequence sequences

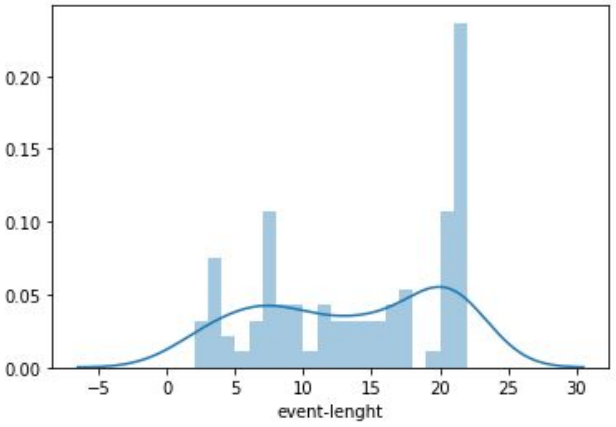

Figure 4 - Histogram of the lengths of the event

Finally, another issue found in the dataset is the considerable number of missing values. In Figure 5 we observe a heatmap showing these lost values and its distribution among the features. Spaces marked in light blue indicate that data exists, and spaces marked in dark blue indicate that the value is missing.

Figure 5 - Heatmap showing the missing values

2.3. Data Wrangling

2.3.1. Dataset Versions

As we mentioned, the rows of the dataset have information about individual messages from each event. This hinders its use as we want to asses entire events not each event's message independently. Thus, we reshaped the dataset into two main 'versions' to address this issue and use them in the rest of the project.

Before explaining them, there is a common characteristic in both, the generation of the input array 'X' and the target feature 'y'. The input array only keeps events that have messages between the maximum time to closest approach (seven days) and two days, as in the goal of the competition. The target feature only keeps events that have a message between two days and the minimum time to closest approach (zero days), in particular, it only keeps the closest message to the minimum (it is a one-dimensional array). So the events need to meet these two conditions, at least one message between seven and two days, and at least another message between two and zero days.

The first version is a three-dimensional array whose shape is events (each row has information of just one event), timesteps (window size or the number of messages of each event), and features (original dataset columns), respectively. This version was used mainly in deep learning models that need 3D arrays to work such as recurrent neural networks (RNNs), and convolutional neural networks (CNNs), and their variants such as LSTM Autoencoder or Manhattan LSTM.

The second version is a two-dimensional array whose shape is events (as in the first version) and time-shifted features (now each column is an original feature in a particular instance or timestamp), respectively. This new second dimension encodes the features in the time dimension and hence enables the use of the dataset in two-dimensional analysis or models described later on such as feature selection techniques, dimensionality reduction techniques, or FeedForward neural network models, to name a few.

2.3.2. Imputation Techniques

There are a wide variety of methods to address the problem of missing values. The naive approach we used in the first place was to eliminate rows containing any NaN. And in the case of the column c_rcs_estimate, where the vast majority of values were missing, we removed the column completely.

However, this approach didn't seem to be appropriate because we were eliminating too much data that could be useful for the model. So we decided to use an approximation based on the Nearest Neighbours algorithm. The k-NN is commonly used for classification and data aggregation. This algorithm, for our use case, uses feature similarity to predict the values of the missing data. That is, the new points are assigned based on how closely they resemble the other samples of the dataset, by grouping them into a number k of nearest neighbors. Specifically, this algorithm creates a basic mean impute and then uses the resulting complete list to construct a KDTree. Then, it uses the resulting KDTree to compute the k-NN. After it finds the k nearest neighbors, it takes the weighted average of them. To implement this algorithm we have used the Python library *Impyute*.

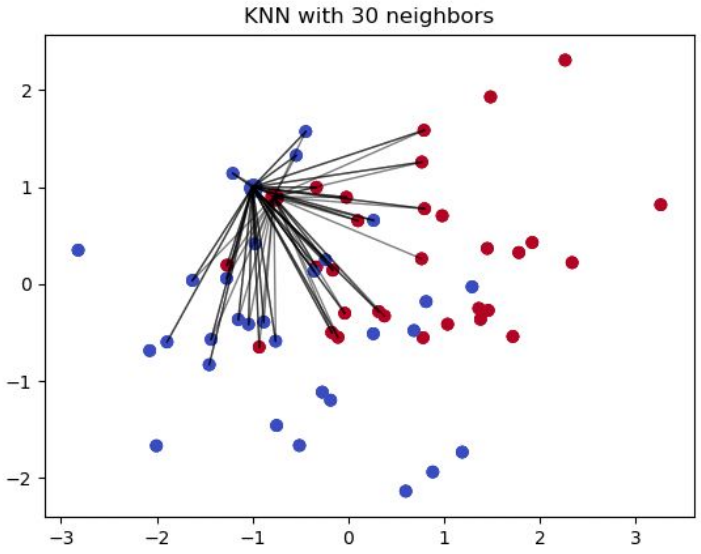

Figure 6 - Example of k-NN algorithm imputing new data points

The major advantage of using this algorithm is that it can be much more precise than the more typical methods such as the mean, median or most frequent value. However, its great disadvantage is that it is quite sensitive to outliers. Besides, it is computationally very expensive, depending on the number of near neighbors used.

2.3.3. Sequence Padding

A huge problem with the dataset is that its events have different lengths, and deep learning models need fixed-size dimensions to work. To address it and be able to use more than one message, we pad the sequences using a constant padding value. Bear in mind, that this step played a big role in the results.

We fixed it to three different values: the minimum, the maximum, and the mean (for each feature), as it was the easiest way to approach it giving the future data scaling (more on this and their performance in section 6.1.1.).

Other padding strategies considered such as independent padding value for each feature but were ultimately discarded due to time constraints.

Another discarded approach that would also help to alleviate the variable periodicity problem is an interpolation. This would make the sequences of messages of each event have constant periods. It was discarded because we found it too complex to implement due to the high variability in the periodicity of the events.

2.3.4. Oversampling & Undersampling

To address the problem of the large imbalance between high and low-risk classes, several algorithms available in the Python library *Imbalanced-learn* have been used. This library implements numerous algorithms to, at the data level, address the unbalance of classes with undersampling, oversampling or combinations of both methods.

The method that has provided the best results is that proposed by Chawla et al. in 2002: Synthetic Minority Over-sampling Technique (SMOTE). This algorithm is considered a state-of-art oversampling technique and works very well in a wide variety of applications. It generates new synthetic data based on the feature space similarities between existing minority instances.

To do this, find the K-Nearest Neighbors of each instance of the minority class, randomly select one of them and calculate linear interpolations to create new data in the neighborhood.

The main disadvantage of the SMOTE algorithm is that it does not take into account the neighboring examples of the other classes, which can result in an increase in overlapping and can introduce noise.

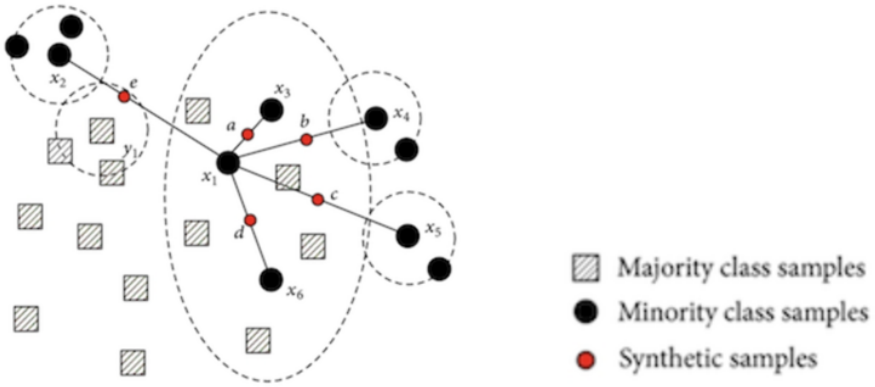

Figure 7 - SMOTE algorithm visualization

In addition to the explained method, different oversampling algorithms have been tested such as ADASYN (Adaptive Synthetic Sampling, like SMOTE, makes use of the k-NN algorithm but adapts some weights to focus on the most difficult instances to learn), SVMSMOTE (variant of SMOTE that helps the SVM algorithm to group the neighboring data) and several more. Unfortunately, these methods have given worse results in the testing, so finally the used algorithm has been the SMOTE.

Additionally to the oversampling methods, different undersampling techniques and combinations of both have been tested. But no significant improvement in the test results has been observed, and in some cases, a degradation has even been noted. Therefore it has finally been decided not to use these algorithms for DL models, only SMOTE.

3. Exploratory Data Analysis

In this section we will carry out a statistical study of the data, we will talk about how the variables are distributed, how they relate to each other and if they provide us with useful information about the target variable

3.1. Distributions

To study the distributions of the numerical variables, we have selected those that have a greater correlation with risk since it was impossible for us to visualize the 103 features.

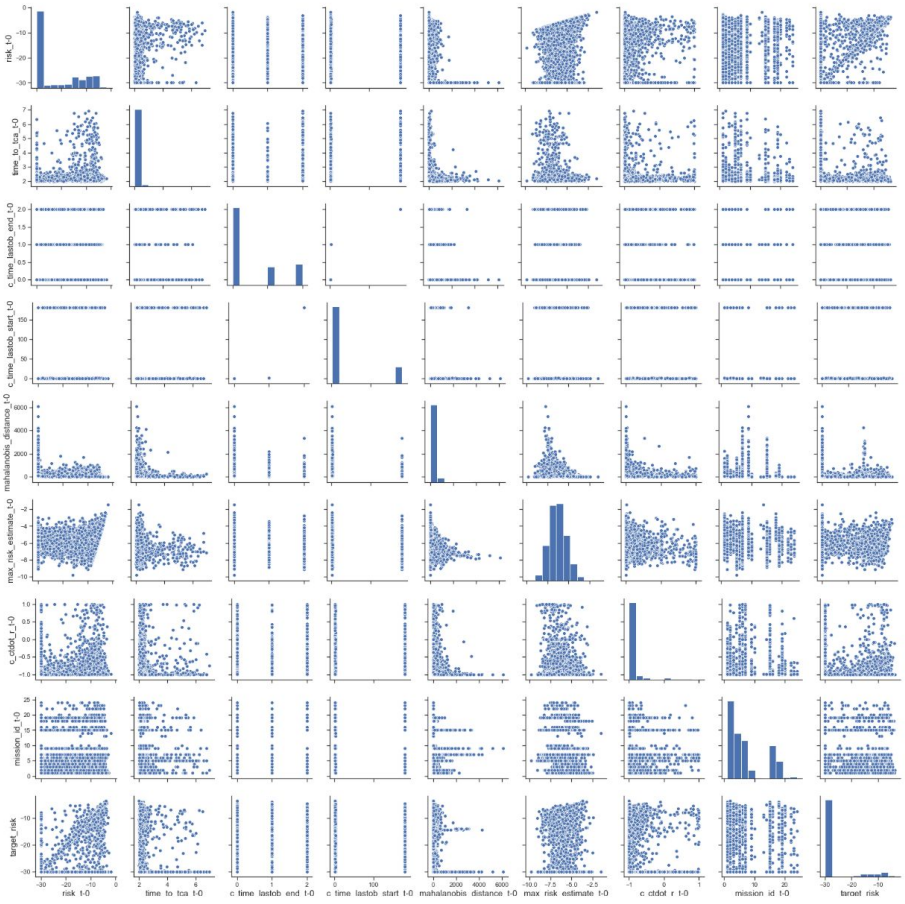

Figure 8 - Scatter Matrix

In the scatter matrix we can observe the distributions that are so irregular in most of the variables, this in the "risk" presented us a great problem since we had to use techniques to compensate the target variable to be able to train the models correctly. Only "max_risk_estimate" has a Gaussian distribution.

We can also point out that no variable has a linear relationship with the target variable. We emphasize the relationship between "time_to_tca" and "risk" since we observe that even if there is little time left for the impact the risk does not have to increase.

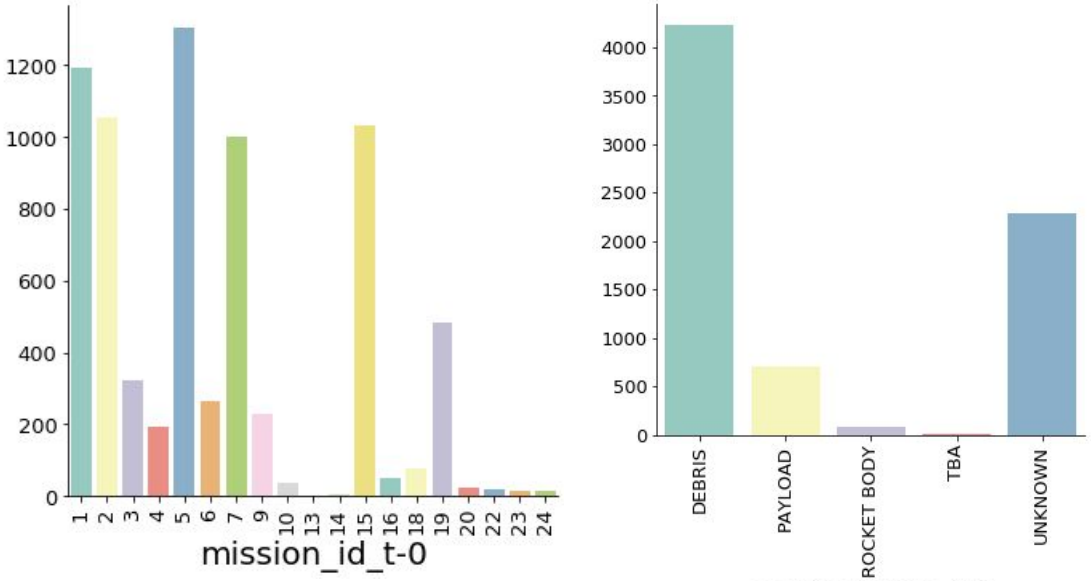

Figure 9 - Distribution of the categorical features

As for the categorical variables, we highlight the great difference in moments of events identified with a *mission_id* and a specific type of object. Since we can find more than 1000 of one type and less than 100 of another.

3.2. Correlations

To better observe if there is any variable linked in any way to the *risk* feature we map the correlations. This will give us great information when choosing the best variables or discarding them and thus reduce the dimensions of the dataset.

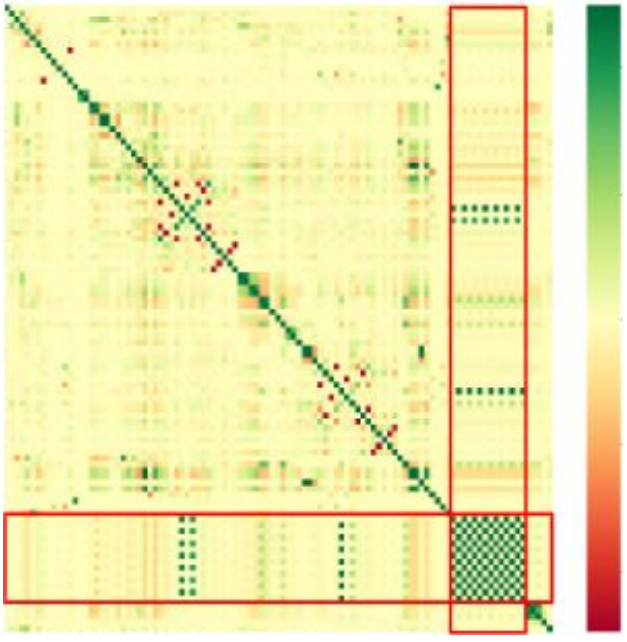

Figure 10 - Heatmap showing the Pearson's correlation

In the correlation heatmap, we can observe in the first line that there is no variable with a high correlation with the target variable so we should use feature selection techniques to choose the features that help us produce the best results. We can also point out that there are several variables with a strong correlation which will facilitate the task of reducing the dimension of the dataset.

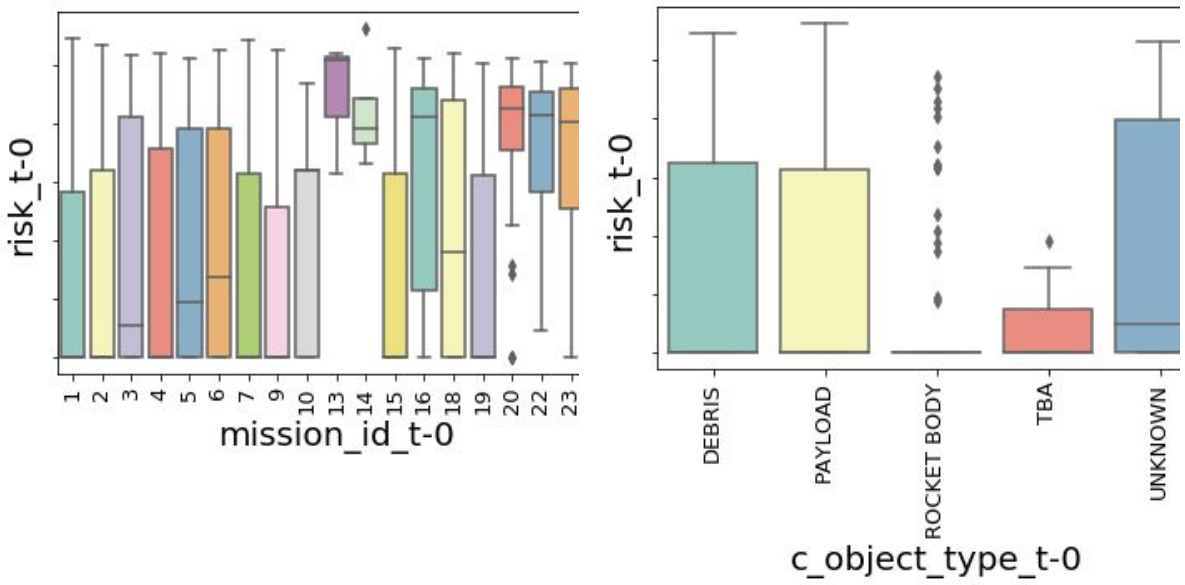

Figure 11 - Relation between the categorical features and risk

As for the categorical variables we can highlight that there are some *mission_id* that only present a high risk like 13, 14 or 20 and that there are some satellite chasing objects that do not present any risk, like rocket body or TBA. All this information will help us to isolate the events that present a high risk

3.3. Dimensionality Reduction

Finally, we utilized dimensionality reduction techniques to assess whether high-risk events can be isolated by reducing the dimensions of the dataset. To do so, we selected the most popular technique, *principal component analysis* or PCA.

We tried many configurations using different dataset versions, padding values for event sequences, scaling methods, and timesteps. In the final configuration, we used (i) the second dataset version (all features from the original dataset are shifted as many time as the predefined timestep, see Section 2.3.), (ii) padding value equals to the minimum value for each feature, (iii) minimum-maximum scaling and (iv) timestep equals to five.

The following figure shows the final results. Here, data are oddly shaped as a bird-like figure. In the “bird’s head”, there exists a cluster where most high-risk events lie in. However, these events are far from isolated from the rest. Thus, we can conclude that using PCA, a simple linear approach to dimensionality reduction, the high-risk events can be somehow put together but not isolated from the rest. Non-linear methods such as deep-learning-based ones might help to achieve better results.

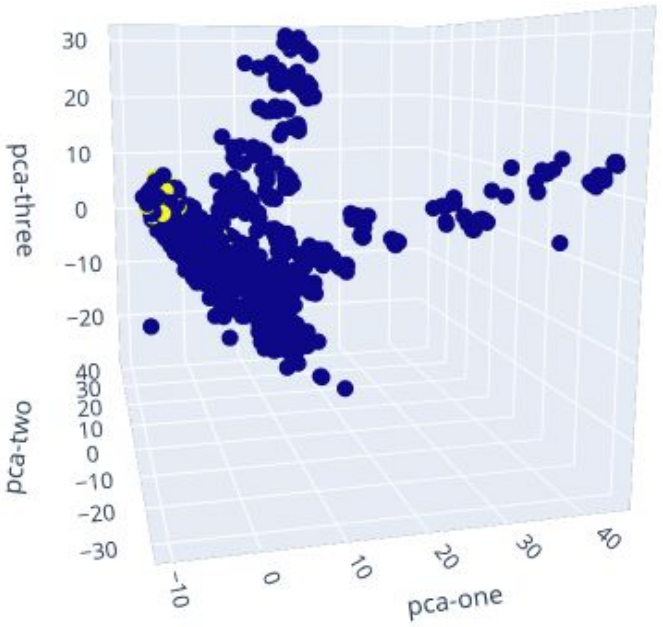

Figure .12- PCA final results: low-risk events (blue) and high-risk events (yellow).

4. Feature Selection

As we have mentioned before, we had a dataset with 103 variables that were not correlated with "risk", that is, we could make the mistake of using some features that only included noise and that did not help at all to the training of the models.

On the other hand, we also found another problem in our dataset since each event had a different length of time. In order to work with the data, we decided to set a fixed time window. In this section, we will decide which features we will use for the modeling and which size will be the temporary window used.

4.1. Target Feature Analysis

In this analysis, we will select the features that will help us predict risk. We will use different feature selection techniques: filtering, wrapping and embedded techniques. In addition to this, we have created a new random number variable that will help us discard those features that only add noise.

- **Filtering techniques:** The first technique used is the Pearson correlation as we have seen in the previous section. The second is the Univariate Selection, it examines individually all the features to compare each of them with the target feature. As we can see in the figure, the most important columns are: time_to_tca, c_time_lastob_end, c_time_lastob_start, max_risk_estimate, c_ctdot_r

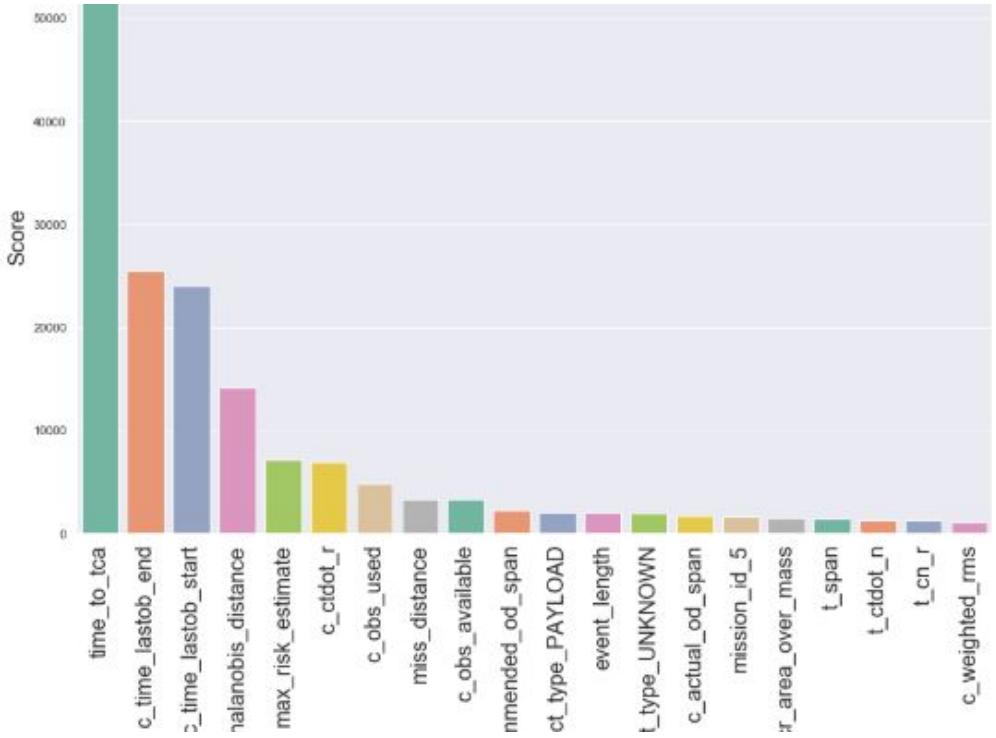

Figure 13 - Filtering technique: Univariate Selection

- **Wrapping Techniques:** The selected wrapping technique is a recursive feature elimination (RFE). Its goal is to recursively remove the weakest features until it gets to a specific number. We have used several models (Linear Regression, Random

Forest Regressor, Lasso and Ridge) to perform this technique and we have selected the ones chosen by more than one model.

We also tried the variant RFE-CV but it didn't give us any useful information.

- **Embedded Technique:** We used Feature Importance. It uses built-in parameters from some models that weight the importance of each feature such as trees and ensemble of trees. We used several models as well such as Decision Trees, Random Forest Regressor and Extra Trees Regressor.

Finally, the final list of selected variables will be formed by those selected by each technique. The result is a total of 24 features:

'time_to_tca', 'c_time_lastob_end', 'c_time_lastob_start', 'max_risk_estimate', 'c_ctdot_r', 'c_obs_used', 'miss_distance', 'c_obs_available', 'c_recommended_od_span', 'c_object_type_PAYLOAD', 'event_length', 'c_object_type_UNKNOWN', 'c_actual_od_span', 'c_cr_area_over_mass', 't_h_per', 'relative_velocity_t', 'relative_speed', 't_rcs_estimate', 'c_cd_area_over_mass', 'c_crdot_t', 'c_sigma_t', 'c_sigma_rdot', 'max_risk_scaling', 't_span'

4.2. Time Series Analysis

In this analysis our purpose will be to determine what the optimal time window should be. To do this we will use the same techniques as in the previous analysis. The variation is in how we have treated the data for this analysis. What we have done is to treat each instant of an event as a different feature and then check which ones offer more information to predict the final risk. The end result was that the optimal window should be between 4 and 8 moments long.

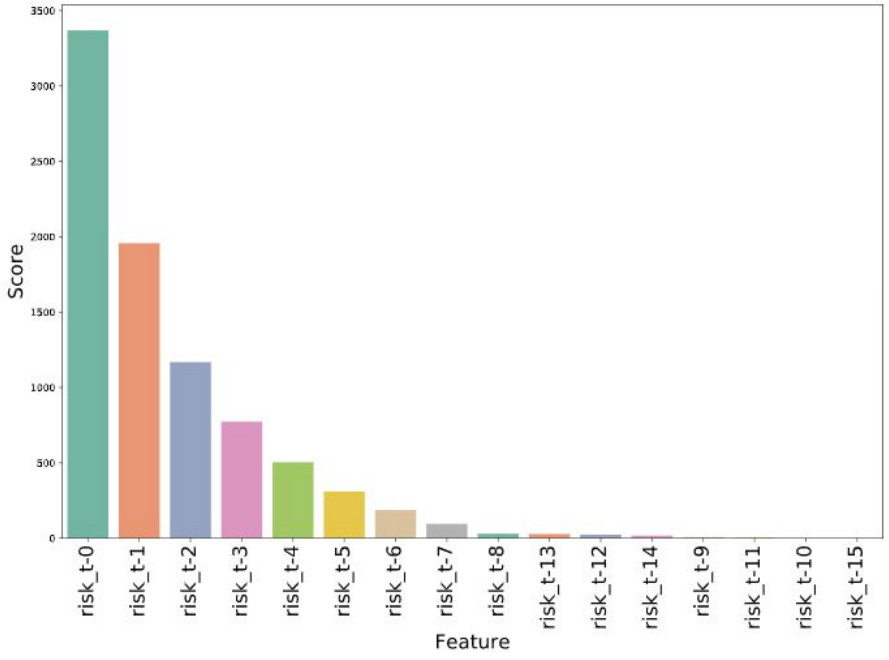

Figure 14 - Temporal Analysis Filtering technique

5. Feature Engineering

In this section, we discuss different ways of addressing feature engineering to get more meaningful information from the available data to ultimately enhance models' performance. We used three approaches: (i) brute-force feature engineering in which we generated as many features as possible using the best features; (ii) event information exploitation in which we extracted more information from the events; and (iii) domain-specific knowledge in which we used our understanding on the topic to generate new features.

5.1. Brute Force

Our first approach to feature engineering is based on leveraging the best features from the feature selection analysis to create thousands more using simple primitives. While this might not be the best approach as it lacks interpretability, domain knowledge, and might end up being useless, it also provides a great opportunity for data augmentation (much needed in this problem).

Around 2000 features were created thanks to Python libraries that make this process really straightforward. First, we selected the ten best features from the feature selection analysis. Then, we selected the primitives: summation, subtraction, multiplication, division, modulo, logarithm, square root, and exponentiation. Finally, we selected the wanted number of features, as trying to generate all combinations is incomprehensible.

We used this method in some recurrent neural network configurations described in Section 6. The results did not improve that much and the runtime went up too much to contemplate its use in other models.

5.2. Information on Events

The most important features in the dataset are undoubtedly the risk and time to closest approach (tca). In our second approach, we leverage their potential to generate more relevant information about each event in the dataset.

The complete list of features generated is the following: (i) 'event length' contains the number of timestamps (sequence length) that each event has; (ii) 'high risks' includes the number of high-risk timestamps that each event has; (iii) 'mean tca' computes the meantime jump of each event (useful as event sequences do not have constant periods); (iv) 'mean risk' calculates the mean risk jump of each event; (v) 'positive risk jumps' counts the number of times the risk goes up between two timestamps within the same event; and (vi) negative risk jumps' is the opposite of the last one, it counts the number of times that the risk goes down.

We assessed some of these features such as 'event length' using feature selection techniques from Section 4. The results were positive, ranking them among the best features in the dataset.

5.3. Domain-Specific Knowledge

The creation of this kind of feature is the most complicated because a great knowledge of the subject is needed to be able to create really useful variables.

In our case, although our knowledge was limited, we thought of using the features that indicate the relative speed of the chasing object with respect to the satellite even though these columns were discarded in the feature selection process. Although we were not able to calculate the relative trajectory of the object, we created some variables that indicate the distance the chaser would travel at that speed in the time remaining for the impact.

When analyzing how good these new features were, filtering techniques told us that it was a good variable to predict risk. However, in the rest of the techniques, both wrapping and embedded, it was not a prominent variable.

For this reason, we ended up discarding these new features because they did not seem to provide relevant information that would make us obtain better results than with the previously selected variables.

6. Deep Learning Modeling

In this section, we discuss different deep learning models to solve this problem. First, we take a look at how we have prepared our dataset to fit the models: data splitting, data scaling, cross-validation, problem modeling, and hyperparameter tuning. Then, we describe how we have evaluated them. Finally, we comment on the design of each of them and their results.

6.1. Data Preparation & Considerations

6.1.1. Data Scaling

The first part was data scaling. To do so, we selected several techniques. The first one is minimum-maximum scaling or re-scaling. It is really popular due to its simplicity. In models that needed sequence paddings such as CNN, RNN, autoencoders and siamese networks, it was the preferred option as it eases the padding a lot which was really important.

The second technique is standardization, it is more robust than the former as it manages outlier better. However, padding was trickier using this one, and the results tended to overfit quiet easily in models that rely on padding.

The last scaling technique was quantile transformation. As the name implies, it transforms features using their quantiles. We selected it because the person who was third in the competition used as its main transformation. It did not work well so we ultimately discarded it.

Bear in mind that we only used the competition train set as the test was not disclosed. Aside from this, categorical features were transformed using one-hot encoding. We tried other approaches such as binary encoding to reduce the number.

6.1.2. Data Splitting

Splitting the dataset into train, validation, and test sets is considered a good practice. The key part is the validation set. Many people use only train and test and it leads to overfitting. Using a validation dataset while fitting the neural network alleviates this problem a lot. As the dataset is heavily unbalanced, we stratified and randomized the process to introduce as least bias as possible.

We tried many different percentages but finally stick to ~25% for the test, ~15% validation, and 60% train. We wanted a high test percentage because it reflected better the size of the test dataset from the challenge and the evaluation metric is a little less harsh with false negatives.

6.1.3. Problem Modeling

As explained in the following Section *6.2 Model Evaluation*, the evaluation metric has two parts a regression metric MSE and a classification metric f-beta. This widens the modeling possibilities of this problem.

First, we tried regression (and for the classification metric we simply used the high-risk threshold to categorize the continuous target feature). We used mainly MSE to train the regression as the evaluation metric used it. The results with this simple modeling were not great, so we searched for something better.

Then, we tried classification (and for the regression metric we fixed a high-risk value and a low-risk value to weight the categories of the target feature). This approach gave the best results as balancing the dataset was so much easier using classification-based techniques instead of regression-based ones. It is worth noting that the classification threshold (from which threshold of the prediction array distribution we consider a prediction a high-risk or low-risk event) was set to automatically give the best possible score.

Finally, we tried to model it as anomaly detection. The main reason is that we have two classes that are extremely unbalanced 100/1 ratio, which is the main characteristic of anomaly detection problems. The best part is that balancing classes is not needed as this kind of model assumes that the classes are unbalanced. We comment more on this in Section *6.3.4 Autoencoders*.

6.1.4. Hyperparameter Tuning & Cross-Validation

In deep learning models, the number of hyperparameters is usually really high. So, it is necessary to find ways to accelerate their tuning process. For doing so, we tried two approaches. First, we used *Random Search* thanks to Python libraries such as Keras Tuner that ease the implementation. It was tested in RNN Models but the results were not good enough to use it in the remaining models as the runtime increases exponentially. If we would have more time, we would have used it in the rest.

The second approach was simply trial and error. All of us had some previous experience in deep learning, hence we leverage our knowledge to tune the hyperparameters. It was the quickest method, and thus the preferred one as we developed many models that need at least a minimum tuning.

Finally, we address cross-validation. It is complicated to developed cross-validation in this kind of model as it takes too long to test every possibility. Our focus here was to train several times the best models to check if they were overfitting and rely on the randomization of the splitting and training to prove it.

6.2. Model Evaluation

6.2.1. Evaluation Metric

The first step to evaluate our models is to set the evaluation metrics. In our case, we simply used the metric given by the competition that was used in the leaderboards. They called it L and is the mean squared error of the high-risk events divided by the f-beta score with $\beta=2$.

We notice from the very beginning that this two tweaks (mse only for a subset, and using f-beta instead of f-score) that aim to penalize false negative (not detecting satellite crashes) make the metric really flawed. Thus, we exploited this to boost our model's scores with a simple trick. For the MSE computation, we set all the low-risk prediction as close as possible

to the threshold (risk greater or equal to -6). This makes the scores improve a lot as the maximum difference while computing the MSE between two values is now -6 (the high-risk range) instead of -30 (the total risk range).

$$L(r, \hat{r}) = \frac{1}{F_2} MSE(r, \hat{r})$$

One of our main conclusions of this project was drawn from this. No matter how hard we tried with our models, having an evaluation metric that is too complex and thus prone to exploitation in someway seems to make the models not aim to a real solution. In few words, it seems that the models have been trained for something that does not reflect the real problem, and hence their usefulness is questionable.

6.2.2. Baselines

Best practices suggest having some baseline to asses the results of any data science or research project. In our case, we used two baselines, one given by the competition and another one that was harder to pass and reflects better the problem that we wanted to solve.

The former is simply predicting the constant risk value -5 that translate to in the real problem to the detection of all satellite crashes at the expense of always performing maneuvers. This baseline was easily beaten in the first stages of the project.

The latter was the real challenge. Two of us (Jaime and Sergio) have worked with time series forecasting problems for some time and a common baseline that we have used is predicting the last value of X in a sequence, which is called the naive model.

Time series models are prone to overfit and predict the last value given in the input array of the model. Normally, a time series that cannot be modeled to beat the naive model is called a random walk. This means that a time series modeling does not fit the problem.

There are many people that are not aware of this, and it is a huge and recurrent part of modeling time series. In this case, it was hard to beat and we could only do it at the final stage of the project after tuning the models for some time. We could not try these models with the competition test set since it ended halfway through the completion of the project, but there is a strong possibility that we would have ranked higher than we did.

As we mentioned in the last subsection, the metric does not reflect correctly this problem. So for us, beating the best scores does not mean always a better model. There are models with not so good scores that seem more robust than the others.

6.3. Deep Learning Models

6.3.1. FeedForward Neural Networks

As a first simple approach, it was decided to implement feed-forward neural networks using only the last available CDM (with *time_to_tca* > 2) of each event as input for the models. For this model to work properly it was essential to use data balancing techniques (such as

SMOTE) in addition to assigning a significantly different weight to the high risk and low-risk classes, during the training phase.

The challenge was first posed as a regression problem. However, the results were not satisfactory, as the metrics penalized heavily the failures made by the model's predictions. Hence, it was decided to approach the problem from the perspective of a classification problem, where the model must predict whether an event will result in high risk or low risk.

Since the evaluation metric is designed for a regression problem, when our model predicts that an event is a high risk, we set the value of *risk* at -5 (baseline proposed by the competition organizers). In contrast, when the model predicts that the event is low risk, we set the *risk* value very close to -6.

This has been done to improve the test results with the proposed metric, although as explained before, it is a trick that despite improving the results, it does not indicate that the models are better or more robust.

The best results have been obtained with the parameters specified in the following table. An intensive search for hyperparameters was not performed due to a lack of time, which could have further improved the results.

Parameter	Value	Parameter	Value
Layers	5 Dense	Optimizer	RAdam
Neurons	256-128-64-16-4	Batch Size	32
Activation	ReLU	Epochs	60
Regularizer	L2=0.001	Scaler	Min-Max
25 best features from feature selection		Class weights	90 / 2

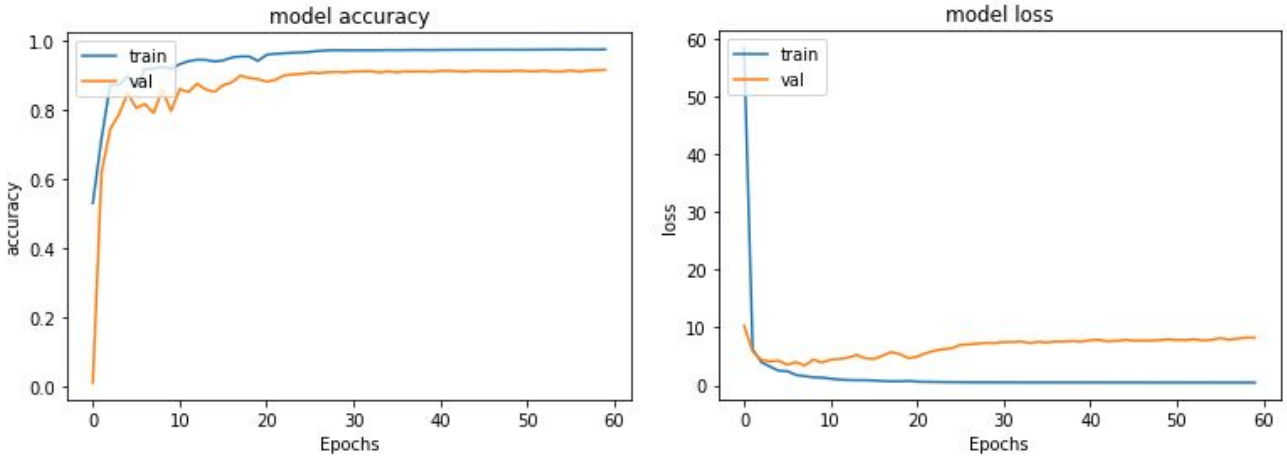

Figure 15 - Model accuracy and loss, FeedForward Networks

6.3.2. Recurrent Neural Networks

As we have stated before, this is a time series forecasting problem in which we predict future values of the risk feature. If there is one deep learning architecture that suits a time series is recurrent neural networks. This is because these networks have the ability to learn information of entire sequences to predict their future values.

We tried almost everything of the previously described steps in our RNN models: different sequence padding, imputing strategies, class balancing, feature engineering, feature selection, different scaling techniques, random search, trial and error, classification, regression, etc. Moreover, we spent quite some time tweaking the model's hyperparameters to get the best score and avoid issues such as overfitting.

As it would be excessive to explain all the configurations and results, we comment only on two models from which we can draw meaningful insights. The first model is the "best" one since it got the best score (even among all proposed architectures). Here are its hyperparameters and configuration:

Parameter	Value	Parameter	Value
Layers	4 LSTM + 1 Dense	Optimizer	Adam
Neurons	50-25-12-5-1	Timesteps	8
Dropout*	0.1	Batch size	256
Recurrent dropout*	0.2	Epochs	100
Regularizer*	L2=0.001	Class weights	300 / 1
Batch normalization* & stateless units*		Scaler	Min-Max
8 best features from feature selection		High-risk Threshold	-6.0

*for each LSTM Layer

The problem with this model is that the training was not as stable as others that score worse. This could lead to weaker models the most you try to improve the score. The main reason was the class weights, the ratio was too high. However, removing them makes the model useless, it needs some kind of balancing. To alleviate the training instability we tried to constrained as much as possible.

One solution to better test the model would be to have access to the full test set of the competition. Sadly, the committee did not want to disclose it. Another solution would be to have a better evaluation metric as discussed in section 6.2.1.

On the other hand, the second model did score worse than the first one, but the training seems to be smoother. This indicates that is also more robust. Here, we move the high-risk threshold a little bit to increase the number of high-risk events (although we gave up on values from -6 to the new threshold). Thus, we can reduce class weight ratio. We also

constrained even more (dropout, regularizer…). The model's configuration and learning curves of both models are as follows:

Parameter	Value	Parameter	Value
Layers	3 LSTM + 1 Dense	Optimizer	Nadam
Neurons	32-16-8-1	Timesteps	17
Dropout*	0.15	Batch size	64
Recurrent dropout*	0.3	Epochs	75
Regularizer*	L2=0.01	Class weights	5 / 1
Batch normalization* & stateless units*		Scaler	Min-Max
8 best features from feature selection		High-risk Threshold	-8.0

*for each LSTM Layer.

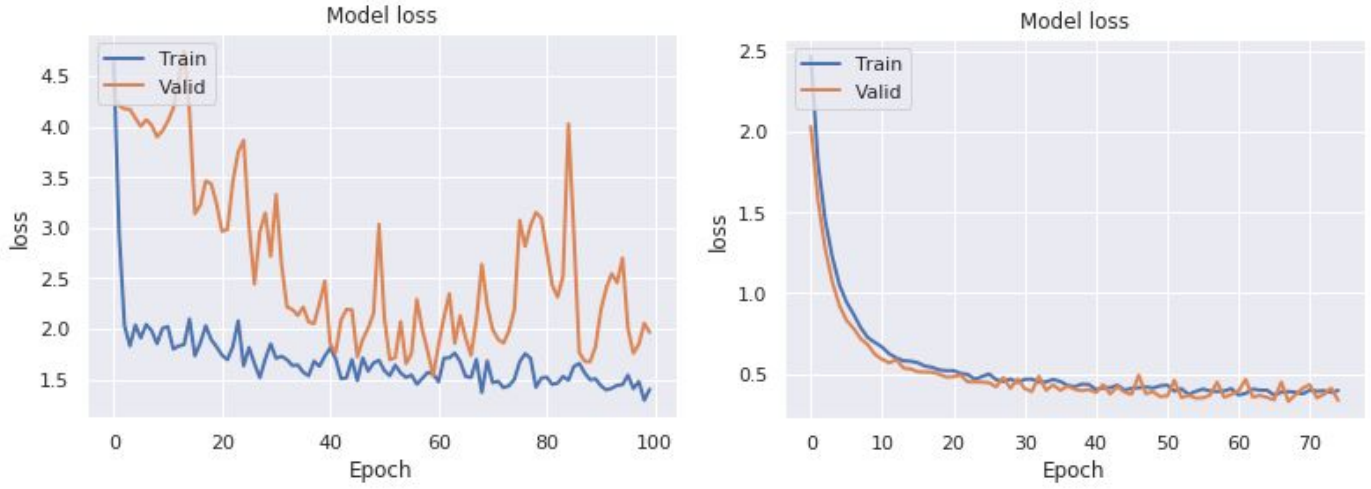

Figure 16 - The "Best" RNN model (left) and the robust RNN model (right).

6.3.3. Convolutional Neural Networks

Another approach to time series forecasting using deep learning is convolutional neural networks with one-dimensional filters. CNNs are great to reduce the number of network parameters by leveraging local spatial coherence. There are many complex architectures for this type of network, but we designed some simple models that resemble VGG16-like blocks with 1D convolutions instead of 2D ones.

One important part of the configuration that we wanted to try was different data shapes. As explained in Section 2.3., we created two versions of the dataset. First, a three-dimensional data frame whose shape is events, timesteps, and features, respectively. Second, a two-dimensional data frame whose shape is events, and timesteps times features. Since 1D convolutional layers need three dimensions to work we reshaped the latter to add a third and 'dummy' dimension.

This helped us to assess whether to use the features in the third dimension (the channels or later activation maps) or the second dimension (the timesteps that are convolved). After tuning both architectures, the second one that encodes the features alongside the timesteps produced better scores. However, it suffers the same stability problem of the recurrent model when trying to beat the best scores. Here is the full list of hyperparameters that were used for the best model:

Parameter	Value	Parameter	Value
Architecture	2 Blocks + 2 Dense	Optimizer	Nadam
Block layers	2 Conv1D + MaxPool. + Dropout	Timesteps	8
		Batch size	64
Conv1D act. maps	[1] 10, [2] 20	Epochs	50
Conv1D filter size	8	Class weights	100 / 1
Pool size	6	Scaler	Min-Max
Dropout	0.5	High-risk Threshold	-6.5
Dense neurons	8, 1	-	-

[1] first block, [2] second block

6.3.4. Autoencoders

Perhaps, the main problem to model this problem is the unbalanced classes. Instead of trying to balance the classes with weights or statistical inference data augmentation, we can model it as an anomaly detection problem. Among anomaly detection solutions, there are some based on deep learning such as autoencoders.

Autoencoders work as a nonlinear dimensionality reduction technique. Their goal is to reconstruct the input in the output, by first shrinking the dimensions (encoder) and then expanding them to the original number (decoder). This simple idea is used in many fields.

As for anomaly detection, first, the algorithm is trained using only data from the dominant class. Then, it is tested with data from both classes. The hypothesis is that the reconstructed error (e.g. MSE) is going to be much higher for the non-dominant class (anomalies) as it was only trained for the dominant class that includes many more data. Thus, you can separate them with a threshold.

We did implement and tuned LSTM autoencoders, but the results were always negative. Even though the learning curves followed a smooth trend after tuning the model with strong foundations **[3]** instead of trial and error, the distribution of the reconstructed error for both classes was always similar. Therefore, they were indistinguishable. The following figures show an example of these distributions, similar in range and shape, from a model that was trained using small overlapped windows of each event to perform data augmentation.

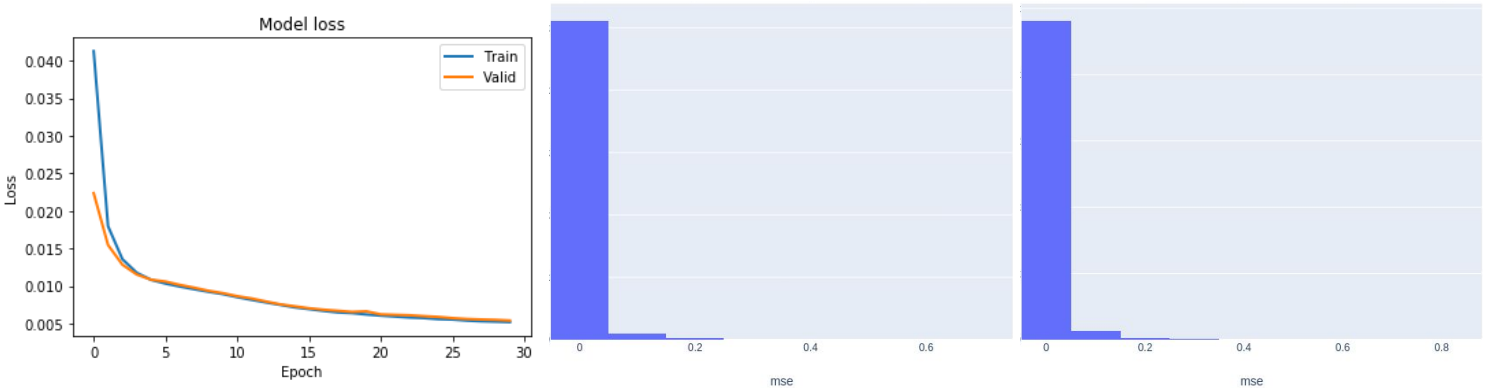

Figure 17 - Autoencoder learning curve, low-risk event and high-risk event distributions.

Parameter	Value	Parameter	Value
Layers	4 LSTM	Optimizer	Adam
Neurons	32-16-16-32	Timesteps*	10
LSTM activation	SELU	Stride*	1
Stateless		Batch size	64
Loss function	MSE	Epochs	30

*timesteps is the window size and stride is the overlapping step.

6.3.5. Siamese Neural Networks

We also developed more advanced architectures such as Siamese Neural Networks (SNNs). The inspiration came from the third-best participant in the competition. Although, he used it to detect events that change from low to high-risk events instead of to just classify low-risk and high-risk events. In our case, we tried the latter approach.

On top of that, he used a few shot learning approaches to generates siamese pairs to avoid overfitting. In our model, we tried that approach and also a data augmentation approach in which we generate as many pairs as possible.

In a few words, siamese networks use two inputs to compare how similar these are, both are processed using the same model, hence the siamese part. This simple design enables an important data reduction to train the models and avoid overfitting, as now the objective is to assess similarities and not the classes like in regular classification approaches.

Since there might be many classes or many samples of some classes to generate all possible pair combinations, the usual approach is to utilize Few Shot Learning and generate a moderate number of pairs. These models have a huge potential in many areas

In our case, we have implemented Manhattan LSTMs (MaLSTM). This model uses LSTMs (recurrent neural networks) to process the pairs, and the Manhattan distance to compare the two outputs (asses their similarity). As we mentioned, first we tried generating a moderate number of pairs randomly (always keeping class balanced and usage). However, it did not

work well. Learning curves were noisy and the scores very low. Hence, we discarded this approach.

At this point, it was clear that the problem training the models is the scarcity of high-risk events in the dataset and the way to augment them or balance the classes. Siamese networks present an opportunity to exploit this little data as much as possible by generating all the possible pairs, so we did.

In our second approach, we generate all combinations of the high-risk class which is equal to the square of its size. This was the first quarter of the MaLSTM input dataset. Then, we generate the second quarter (same number of pairs as the first) by sampling randomly from the low-risk class as it has many more data. Finally, for the third and fourth quarters, we sample for each high-risk event a fixed number of low-risk events equal to the size of the high-risk event data, first for the left input (third quarter) and then for the right input (fourth quarter). The following table summarizes the resulting input dataset and the target feature.

Dataset	Left Input	Right Input	Target
1st Quarter**	High-Risk	High-Risk	True
2nd Quarter	Low-Risk	Low-Risk	True
3rd Quarter	High-Risk	Low-Risk	False
4th Quarter	Low-Risk	High-Risk	False

**this quarter sets the size of the remaining ones.

Thanks to this approach, we were able to vastly boost the number of samples to train the model while giving the same importance to both risk classes and maintaining the balance of the similarity classes. Note that this data augmentation process does not rely on any statistical inference or padding technique.

It only uses real data which makes it more robust. This produced around 7K training samples, roughly the same number in the other regular models. However, if we sacrifice some samples (and thus score) by moving the high-risk threshold from -6 to -7 the number of pairs grows exponentially, setting the number around ~55K pairs.

This makes the model even more robust. Aside from that, siamese models usually need a lot of tweaking, so we carefully tuned them. Here is the full configuration and learning curve for the best model:

Parameter	Value	Parameter	Value
Train pairs	~55K	High-risk Threshold	-7
Layers	2 LSTM + Distance	Optimizer	Nadam
Neurons	32-16	Gradient clipping	1.0
Dropout*	0.25	Learning rate	0.0005

Recurrent dropout*	0.5	Timesteps	17
Batch normalization* & stateless units*		Batch size	512
Scaler	Min-Max	Epochs	20

*for each LSTM Layer.

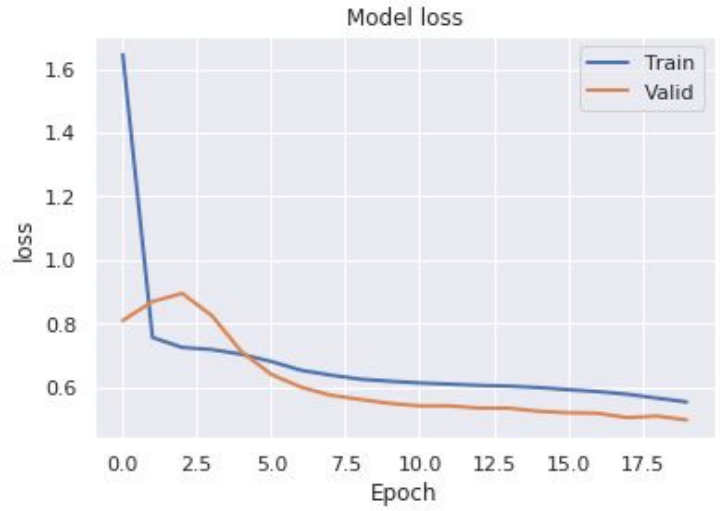

Figure 18 - Best MaLSTM model learning curve.

From its learning curve, we observe a smooth and stable training. Note that the validation has better loss since it is tested without dropout whose value now is really high. To make the process even more robust, we thoroughly design the testing procedure. First, we tried to evaluate each event in the test set by comparing it with the entire train set before classifying it. So, for each test event, we got a distribution of the results of comparing it with all train events. Then, a threshold is set to determine whether the test event is a high or low-risk event. Due to a known memory leakage in the Keras *predict* function, it was impossible to compare each event with the entire train set, so we decided to use a smaller sample of it, while always using all high-risk events.

The results of this model did not score as good as RNNs or CNNs. However, the siamese networks tend to predict correctly most of the crashes while getting wrong more low-risk events. So, it seems that is the most robust or safe model of the ones tried in this project for a real scenario, but not for score better in the competition. It is also worth considering that the evaluation metric or we are not able to capture the real trade-off between maneuvering and crashing, it all comes down to the real value each event has for the ESA.

6.3.6. Deep Ensembles

In order to improve the results obtained in the other DL models, it was decided to implement ensemble methods that would improve, not so much the accuracy of the models, but their robustness. In general, ensemble learning consists of learning how to combine the predictions of multiple models to obtain a better model at the end. There are several flavors of ensemble learning such as bagging, boosting or stacking among others.

In this work, due to lack of time, we decided to implement only one flavor of ensemble learning, the stacking method. These consist of building a meta-model that obtains as inputs the solutions of the base models. In this way, and if the base models have enough heterogeneity, the meta-model learns to combine the solutions of these in a non-deterministic way. To implement this solution we have used the Python library *DeepStack*.

As input, 5 FFNs models have been trained varying their structure, optimizers, activations and other hyperparameters. The meta-model consists of a Random Forest Classifier, which has been given different weights for each class (300/1). The obtained results were quite satisfactory, however, due to the lack of time it has not been possible to test this stacking method with other types of DL models or even combining them with more classical ML models. It is proposed as a future line of work.

6.4. Results Summary

In order to compare all the models, we have summarized all the best ones in the following table. As the number of samples used in the test was not the same for all of them, the scores cannot be compared directly. To address this, we added the time series baseline of each model (the other baseline was beaten by a mile) and computed its improvement with respect to this value.

Bear in mind that as we have stated before, a better score might not lead to a better or robust model. It only means that the model ranks better in the competition. So, these scores should be taken with a grain of salt. Also, since the autoencoders did not work, they have been removed from the table.

Model	Score	Improv.	Model	Score	Improv.
FNN	1.16	**-34.8%**	CNN	0.534	**+12.4%**
Baseline	0.86		Baseline	0.600	
Best RNN	0.429	**+39.9%**	Robust RNN	1.375	**-12.4%**
Baseline	0.600		Baseline	1.223	
Siamese	1.535	**-357%***	Ensemble	1.13	**-31.4%**
Baseline	0.429		Baseline	0.86	

*It might seem high but the competition baseline score was usually between 6 and 10.

7. Conclusions

There are many insights that can be drawn from this project. First, the topic of the competition was not easy to grasp at first sight. We spent quite some time together trying to figure it out. Our knowledge on the matter was limited to a few lectures on satellites during the first year of our master's degree. So, it was more challenging than the typical data science competition.

Once we put our hands on the dataset, we found out that it was really raw and has many problems. We spent a long time discussing and implementing approaches to process the data in different ways, most of them critical to the performance of the deep learning models. The list of problems was significant: variable periodicity and length in the sequences, hugely unbalanced classes, lots of missing values, useless features, etc.

We came up with many approaches to solve all these problems: two dataset versions, several imputation techniques, different padding values for sequences, oversampling and undersampling methods, five types of feature selection techniques for two completely different types of analysis, three approaches to feature engineering, several types of data scaling techniques, problem modelings, hyper-parameter tunings, siamese pairings, etc.

Aside from this thorough data processing, we utilized many Deep Learning architectures and tested many different configurations such as FeedForward Neural Networks, Convolutional Neural Networks, Recurrent Neural Networks, Autoencoders, Deep Ensembles, and Siamese Neural Networks. Some of them were easier to design and implement than others.

Although we beat the two baselines with Recurrent Neural Networks and Convolutional Neural Networks and ranked among the best in the competition (19th out of roughly 100 teams), we think that the evaluation metric used in the competition is flawed and easily exploitable, and hence it might not produce the best models for real contexts. AAnalysing our results, we found out that models that seem to have much more stable learning processes or more robust data augmentation techniques scored worse than the best and more volatile models. Unfortunately, we could not test this idea as the competition ended halfway through the development of this project.

In any case, we did achieve a deep understanding of the challenge and obtained great results with some of the approaches and deep learning models that were proposed. For more information on this project, please visit our Github accounts **[5][6][7]**, the full implementation will be published there shortly.

8. References

[1] ESA, “Space debris by the numbers,” [Online], Available: https://www.esa.int/Safety_Security/Space_Debris/Space_debris_by_the_numbers **[Accessed 2 January 2020]**

[2] ESA, “Collision Avoidance Challenge,” [Online], Available: https://kelvins.esa.int/collision-avoidance-challenge/home/ **[Accessed 2 January 2020]**

[3] Towards Data Science, “Build the right Autoencoder — Tune and Optimize using PCA principles. Part II,” [Online], Available: https://towardsdatascience.com/build-the-right-autoencoder-tune-and-optimize-using-pca-principles-part-ii-24b9cca69bd6 **[Accessed 10 January 2020]**

[4] K. Merz, B. Bastida Virgili, V. Braun, T. Flohrer, Q. Funke, H. Krang, S. Lemmens, J. Siminski, “Current Collision Avoidance service by ESA’s Space Debris Office”, 7th European Conference on Space Debris, 2017. **[Accessed 11 January 2020]**

[5] Github, "Sergio Pérez Morillo Profile," [Online]. Available: https://github.com/spmorillo **[Accessed 11 January 2020]**

[6] Github, "Jaime Pérez Sánchez Profile," [Online]. Available: https://github.com/jaimeperezsanchez **[Accessed 11 January 2020]**

[7] Github, "Carlos Andrés Ramiro Profile," [Online]. Available: https://github.com/carlosaramiro **[Accessed 11 January 2020]**