spotGEO solution description

Max Halford

August 2020

1 Overview

The goal of the spotGEO challenge was to determine the locations of satellites on grayscale images. I took the straightforward approach of classifying whether a given pixel was a satellite or not. The issue with this approach is that the number of samples becomes very large. Moreover, the large majority of pixels are not satellites. Therefore, considering every pixel would induce an extremely unbalanced learning problem. I thus devised a method to sift through most of the pixels and only consider those that I deemed *interesting*. I then collected features for each remaining pixel, trained a bagged version of LightGBM [1], and performed some valuable post-processing treatment.

2 Looking for interesting pixels

Most pixels in the provided telescope images do not contain satellites. On the contrary, most of the pixels are dark. Satellite sightings are noticeable because they look brighter than their surroundings. For almost all pixels, it's quite obvious that they are not satellite sightings. Therefore, I decided to write a script to distinguish between pixels that had no chance at all of being satellites and those that did. The goal was to devise a method that filtered out the points that were "obviously" not satellites, therefore greatly reducing the amount of samples to consider.

I did this in an unsupervised manner, by looking at the brightness of the region surrounding each pixel. I determined the brightness of each surrounding region by applying a median filter [2]. I defined each region as the rectangle of width 16 and height 12 that surrounds a pixel. I only kept the pixels whose brightness was larger than the median brightness of the associated region by a margin of 6. Increasing the margin means that less pixels are deemed interesting, whereas lowering it is less conservative. For the training set, I labeled the interesting pixels with the provided annotations by framing it as a linear sum assignment problem. I only kept the assignments for which the Chebyshev distance with the closest interesting pixels was lower or equal to 2.

[1] `https://github.com/microsoft/LightGBM`
[2] `https://www.wikiwand.com/en/Median_filter`

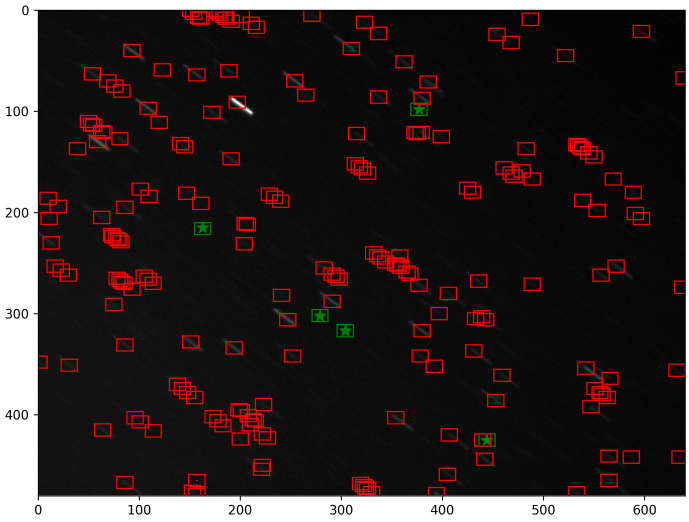

Figure 1: Interesting pixels found for frame 1 of sequence 1069. Each box is a region that surrounds a pixel deemed interesting. The red boxes are pixels that are not close enough to any of the satellites sightings provided in the dataset. The green boxes are interesting pixels that are close to one of the sightings in the dataset. The green stars represent the sightings from the dataset. In this case, all the sightings are accounted for, and 210 pixels are deemed interesting.

I ended up with around 2.8 million pixels that were deemed interesting. This represents 0.1% of the 1.9 billion pixels that were present in the provided dataset. However, this subset of interesting pixels doesn't include all of the sightings that were provided by the competition organisers. Indeed, I only managed to assign 88% of the sightings in the training set to the interesting pixels I extracted. If the training set and the test set images follow the same distribution, then there is a significant chunk of sightings that the rest of my pipeline is simply ignoring. In order to include more sightings, I would have to devise a pixel selection process that was less conservative, but this would probably have required me to process more samples.

3 Extracting features and learning

I extracted features for each interesting pixel. My feature extraction process is very straightforward. I computed descriptive statistics of square regions of different widths around each pixel. I flattened each region into a vector and

computed the latter's mean, variance, maximum, minimum, entropy, skew, and kurtosis. I looked at squares of width 3, 7, 11, and 15.

At this point I've framed the problem as a straightforward binary classification problem. I thus decided to use LightGBM, which is probably the best off-the-shelf learning algorithm for tabular data. I didn't bother a lot with hyperparameter tuning. I used a learning rate of 0.1 and used 64 leaves in each tree. The only fancy thing I did was to perform a bagging process on top of LightGBM, which is the standard thing to do in Kaggle competitions.

4 Post-processing

My pipeline at this point was decent, but nowhere near as performant on the public leaderboard as other participants. My intuition was that I was not exploiting the sequential aspect of the problem. I was instead looking at each image in a vacuum by itself. Therefore, I thought of a post-processing step to execute after having made my first set predictions. In fact, this post-processing step could be applied to the solutions of other participants.

By overlaying the satellite sightings from each frame on top of each other, it's quite apparent that each satellite follows a straight path. Moreover, the gap between each pair of consecutive positions seems to be constant for every satellite. Therefore, if the coordinates of a satellite are known for two frames, then the positions for the three other frames can be interpolated. The challenge is thus to cluster predictions from different frames into trajectories.

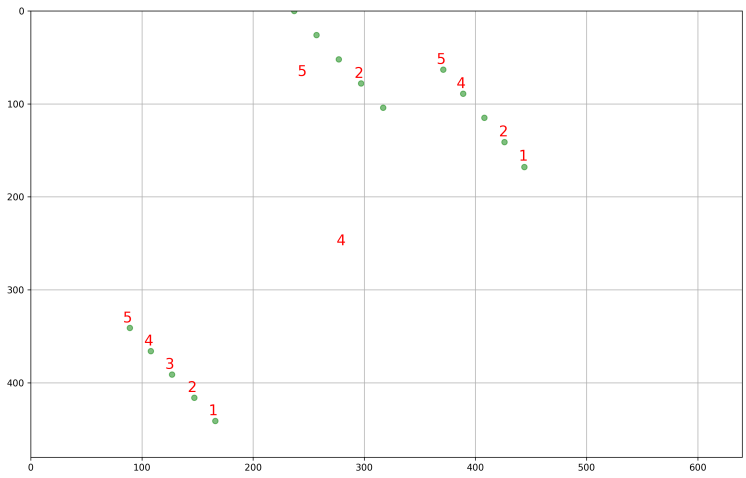

Figure 2: Predictions and ground truths for sequence 32 before post-processing. The ground truths are indicated by the green dots. The predictions are indicated by the red numbers, where each number stands for the associated frame. The predictive algorithm managed to find 4 out of the 5 positions for the upper-right trajectory. It's quite apparent that the position for the third frame can be inferred from the existing predictions. The only challenge is to design some algorithm that can correctly identify the fact that those predictions belong to the same trajectory.

I identified satellite trajectories from different frames by fitting straight lines to the coordinates. I did this in a iterative fashion. First, I considered all combinations of five predictions. For each combination, I attempted to fit a straight line. If the fit looked good, then I put that combination aside and labelled them all as part of one trajectory. I then applied the same process for every combination of four predictions from different frames. If I found a good line that went through four points, then I interpolated the position of the fifth point from this line. I then applied the exact same process for combinations of three points.

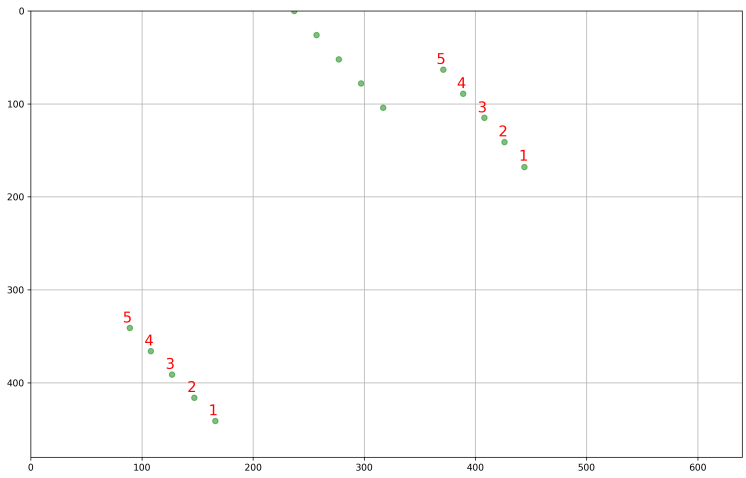

Figure 3: Predictions and ground truths for sequence 32 after post-processing. The third frame position of the upper-right trajectory got correctly interpolated. Meanwhile, the predictions that were by themselves got discarded because there were not other predictions from different frames to match them with.

This post-processing step worked really well for me. It lowered my public F1 score by about 0.2. One of the advantages is that allowed me to be less conservative with my predictive algorithm. Indeed, I got a boost by lowering the classification threshold. I classified each prediction as a satellite sighting when the predicted probability was higher than 0.29 instead of the 0.5 default. Indeed, weak true positives got kept if there were part of a trajectory, whereas false positive got discarded because no other predictions from other frames got assigned to them.

5 Conclusion

I'm quite satisfied with my performance with respect to the time that I put into it. I participated on my own. I also had a busy summer and not a lot of time to dedicate to the competition. I think that my biggest source of improvement would come from designing a better filtering process. Indeed, I think that I've ignored too many pixels. The upside is that I'm happy with the performance of the rest of the process. In particular, I would be curious to see what my post-processing step could bring to other solutions.