NASA Asteroid Data Hunter Marathon Match 2 res1 Solution Description

By Ryan Seghers
September 17, 2014
Revision 1

This document describes my solution to the NASA Asteroid Data Hunter Marathon Match 2 (MM2) coding competition.

1 Overview

The goal of the contest was to write a program to match the "truth" asteroid detections for a set of Catalina Sky Survey images on a testing set of images after having been trained on a training set of images.

My solution is about 14,000 lines of C++ code. I developed it in Visual Studio 2012 on Windows and wrote a program (BuildSource.exe) to combine all of the separate header files together and strip comments for submission to TopCoder.

There are no offline tools associated with my solution. The classifier is trained on-line, during the program execution.

My solution placed second with a final score of 329,686.

2 Terms

Some terms used in this document:

- **Blob**: A group of adjacent pixels in an image.
- **CA**: Candidate Asteroid: An object (possibly represented by a blob of pixels) that is being considered as a possible asteroid.
- **Detector**: Despite the name of the competition and the common use, in here by "detector" I mean the part of the code that finds the CA's, before classification.
- **FP**: False-positive
- **FN**: False-negative
- **Frame**: I call each of the 4 images a frame in the image set.
- **Image Set**: A set of 4 related images, usually 10 minutes apart.
- **RF**: Random Forest classifier

Also note that I use apostrophes in a non-standard way. Due to the way programming variable names and camel/Pascal casing interact I use apostrophes to indicate plurality whereas they would normally indicate possessiveness. For example I type CA's to mean plural CA instead of or possibly in addition to indicating possessive.

3 Design and Implementation

3.1 Overview

From a high level my approach was:

1) Use image diffing to detect changes between each of the 4 images vs their common signal. The changes are "blobs" of pixels in each of the diff images.
2) Examine blobs to find similar ones in linear motion between the 4 images and call these Candidate Asteroids (CA's).
3) Compute features on CA's.
4) Train the following RF classifiers on the CA's features:
    a. rfAsteroid: Produces pAsteroid the probability of being an asteroid. (Unfortunately named just "rf" and "pClass1" in the code.)
    b. rfNeo: Produces pNeo the probability of being a NEO.
    c. rfOverlap: Produces pOverlap the probability of being overlapped in the truth data.
5) Apply the classifiers and produce the results.

3.2 Processing

The following sections mirror the structure of the code, in processing order.

1) trainingData()
    a. Load and parse inputs.
    b. Skip image sets with 0 detections. (I found they didn't help me in the offline data set and did not have time to experiment with the real test data set.)
    c. ProcessImageSet(): the bulk of the processing, see below.
    d. Accumulate CA's and detection truth.
2) testingData()
    a. For the first call, train the rfAsteroid, rfNeo, and rfOverlap classifiers on the accumulated testing CA's and detection truth.
    b. ProcessImageSet(): the bulk of the processing, see below.
    c. Apply the classifiers to the resulting CA's.
    d. Accumulate the top pAsteroid CA's (up to the maximum number of answers allowed).
3) getAnswer()
    a. (Working on the CA's accumulated in testingData().)
    b. Set each CA's isNeo member based on pNeo as previously computed by rfNeo.
        i. This chooses very few to be NEO's, per the training data which indicates low prior probability.
    c. Insert duplicate CA's for select CA's as identified by rfOverlap.
        i. This just chooses all of the CA's with pOverlap over a particular threshold. And that threshold is very low.
    d. Do a final sort that combines pNeo with pAsteroid, to try to pull more likely NEO's up in the ranking.
    e. Convert to answer string format and return.

3.2.1 ProcessImageSet()

ProcessImageSet() is called from both trainingData() and testingData() and does the bulk of the processing. This function has these steps:

1) Crop input images and adjust the truth detections' pixel coordinates accordingly.
    a. I cropped to get to cache-aligned size 4096 wide for performance and also because the extra columns appear to be always garbage.
2) Align images using affine transform (without any shear), and adjust the truth detections accordingly.
    a. I adjusted the truth detections without rotation assuming that small rotations would fall within the relatively wide tolerances for determining correspondence between my CA's and truth detections. (And never checked this assumption.)
3) Image Fixup and Mask Creation
    a. Fix and/or create masks for artifacts in the images. There are magic and/or wrapped values in the image data. This function produces a mask to select "bad" pixels, a foreground mask to select "good" foreground pixels, a background mask to select "good" background pixels, a "dip" mask for apparent deadened camera pixel areas, and a fixed image with some things fixed.
    b. Create a super-bright object ("sbo") mask for a feature that measures proximity of a CA to a bright object that may create image artifacts (that I call scatter but there is certainly a better real term).
4) Create the "nsi" image for each frame. The nsi image is "number of stddevs increase" meaning it has non-zero pixels where the original image pixel was greater than its surroundings by at least 2 standard deviations. The nsi images are 8-bit.
5) Create the nsiDiff images for each frame. The nsi diff image represents the difference between this frame and the common signal.
    a. Create the common signal image. It is supposed to represent the unchanging background. It is created by a custom 4-image median function.
    b. Subtract each frame image from the common signal image.
    c. Filter the diff image. For example drop single isolated pixels, etc.
6) FindBlobsTop(): Find Blobs
    a. This does a standard connected-components labeling of the blobs in each frame's image, does some simple feature measurements of the blobs, drops some blobs based on those measurements, and creates a new image with just the surviving blobs.
    b. Compute "salience" scores/features which attempt to quantify how salient (how much they stick out) each blob is.
7) FindCandidatesSimple(): Find CA's
    a. For each adjacent pair of frames:
        i. For each blob in the first frame, look at all surrounding blobs in the other frame.
            1. For each pair found that is somewhat similar:
                a. Extrapolate its coordinates in the other frames and look there for similar blobs. Not all 4 frames must have blobs. Blobs get an "explained-missing" score based on masking or image edges.
                b. Define a new CA. Merge it into an accumulated set of CA's and avoid duplicates.

    b. For higher velocity asteroids, repeat the above after removing weaker blobs and with increased velocity thresholds.
    c. FindSlowMoversTop(): (This by itself is probably more like what other competitors did, and does not require the work to get a clean nsi diff image.)
        i. For slower moving asteroids, turn all nsi (not nsi diff) image blobs into objects. Do an analogous search to what is done above on blobs in adjacent frames.
8) PostProcessCandidates()
    a. Compute additional scores. For example a CA proximity score that compare CA's to its neighbors.
9) Drop weakest CA's by simple score (not a classifier score)
    a. (To keep the number accumulated down.)
    b. simpleScore is computed from metrics that try to assess:
        i. how similar the blobs are that compose the candidate
        ii. how round the blobs are
        iii. distance to super bright objects
        iv. how many neighbor CA's there are
    c. simpleScore is critical (because my method produces so many FP's) yet I did not have time to play with it.
10) Return the list of CA's.

3.3 Example Images

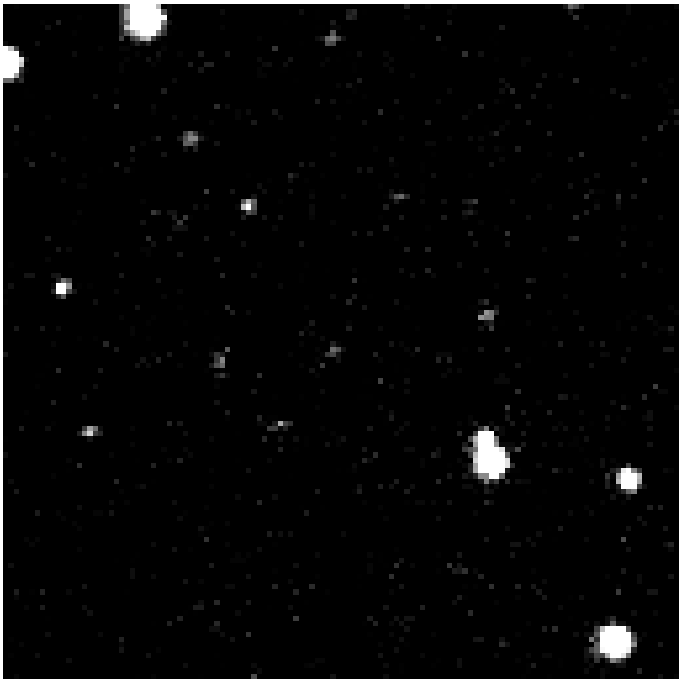

Figure 1 An example nsi image. The detection is at the center of the image.

Figure 2 The corresponding nsi diff image. The detection is at the center of the image.

Figure 1 shows an nsi image, which collapses the 16-bit input image down to 8 bits and makes pixel values relative to their neighbors. Figure 2 shows the signal left when the common signal from the 4 frames is subtracted, in this case leaving a probable asteroid (center of image).

4 Discussion

4.1 Image vs Object Space

My solution may be relatively different from other approaches (though I haven't read much of their code yet) in that I attempted to use image diffing: I looked for objects after diffing (subtracting) out signal common to the 4 images. I stayed in "image space" longer than other competitors, I think. My approach is intuitively appealing because it lets me see what is happening, however noise and artifacts and image variations make it difficult.

The alternative is to get out of image space more quickly, for example by using convolution to find kernel matches (Gaussians?) in each image, and then starting to compare between images.

My approach involved quite a bit of work, lines of code, and CPU cycles. The result of the competition doesn't prove (nor disprove) that this method is inferior because it might just be my implementation, or other factors like the features chosen or the classifier, etc. However I think the competition does prove that it was too much work for the time frame.

Note that image diffing inherently sacrifices slow moving objects (because they are not very different between images), which I noticed late in the contest. So image diffing cannot be the sole candidate object detector.

Image diffing may also have more trouble with conjoined objects, where a CA blob encounters a stationary blob, but both are still identifiable.

4.2 Follow-Up

Some things I would have liked to do:

- Add a blob classifier: to reduce FP's early on, it might work to have a simple, fast classifier to disqualify blobs that are not realistic. For example some blobs are not remotely round (nor ellipsoidal).
- Experiment with simpleScore, which is an early prioritization score used to reduce the number of CA FP's. Maybe it should be an actual (offline trained) classifier, like a Naïve Bayes or a previously-trained and serialized RF.
- Finish image contour handling, where low frequency variations across images causes problems for the image diff approach. I tried a couple approaches but ran out of time. A DFT/DCT probably makes sense (maybe leaving out the very bright objects) but I didn't have time to find one that I could include in my solution.
    - *Note: I tried tiling and a wide-area filter approach. Tiling worked relatively well, but I removed that to do the wide-area filter approach. Then had trouble with that, commented it out, and forgot to ever re-enable tiling. So my final submission has no contour handling at all. Oops.*
- Feature selection: I didn't do any work on feature selection other than adding features that seemed useful. There are many more features to try, and having bad features can hurt classifier performance.
- Compare my detector to the convolution approach (using the same features and classifier to isolate differences).
- Compare either detector to a combined approach.

    - It may make sense to have several flavors of detector all finding candidates, and combine those candidates in some way, with the source detector and maybe features specific to that detector as feature inputs to the classifier.
- Investigate FN's and FP's
    - Look at high pAsteroid FP's to try to identify features or other steps to take to eliminate them.
    - Look at low pAsteroid FN's that are bright according to the truth detections to look for ways to improve.
- Hill climb classifier parameters: Once features are good, one can tune classifier performance by adjusting its parameters.
- Lots of other details: reject double-exposure images, use a distance transform for super-bright-object proximity, increase initial nsi blob sensitivity, clean up very bright spike artifacts, etc.

4.3 Caveats

Some caveats about my solution:

- I felt like I was about 1/3rd done when the deadline hit. So it is kind of a mess and not very close to finished.
- I did not have time to prune unused code. There is a lot of it. I started from my MM1 codebase and did not do much cleaning.
- My implementation is heavily stamp-coupled, which is a design anti-pattern that causes complexity. Stamp coupling means lots of the code touches common, shared data structures (like ImageSet, CA, FitsImage) and the touches are interdependent. The alternative would have more data structures and lower performance (because you copy data structures around rather than referring to a single one) but would decrease complexity.
- Despite the solution being speed limited in a score-affecting way, I never got close to doing a speed analysis, and I know there are egregiously slow things going on in there.
- There are a lot of hardcoded constants that I just took a guess on.

4.4 Contest vs Reality

It might be worth pointing out for someone who wants to do real asteroid detection that there are some aspects of my solution and others that one should be aware of:

- There are duplicates/overlaps in the truth detections. So I (and probably other top scorers) artificially introduced my own detection duplicates.
- The winner (at least, probably others, and I should have but forgot) included features like date of image capture in the classification. I think this may not be good for a real asteroid detector because it may be a type of over-fitting that happens to work for the contest and not in reality.
    - Similarly giving the ra/dec to the classifier may work because there are patterns to asteroids' locations, I'm not sure about the ramifications of allowing that prior probability to affect a real detector: an object that looks more like an asteroid but is in an uncommon region of space will be rejected in favor of one that looks less like an asteroid but is in a region where asteroids are more common…
- I'm not sure what the source(s) of the truth detections is/are, but if the sources use a particular type of detection (e.g. convolution) then that detection mechanism may work better for the contest and not necessarily in reality, another type of over-fitting.