Asteroid Data Hunter MM 2

My solution of this Marathon Match is based on *random forest* classification method.

The basic structure of the algorithm is:

- with each training image set:
  1. in each of the 4 images, select *adepts* for asteroids;
  2. find 4-tuples of adepts lying on the line forming a *candidate* for a detection;
  3. assign features to each candidate;
  4. check for each candidate whether it actually is a detection; if yes, check whether it is NEO and/or double detection;
- build three separate random forests from all the candidates, one forest for detections, one for NEOs, one for double detections;
- with each test image set:
  1.
  2. *the same first three steps as with training sets*
  3.
  4. evaluate each candidate using the three forests, so to each candidate three values will be assigned: *detection score* (*DS*), *NEO score* (*NS*) and *double detection score* (*DDS*);
- Sort all candidates by *DS* in descending order and return all candidates as the answer; additionally, if *DDS* is over a specific threshold, return the candidate twice; moreover, mark a specific number of candidates with the highest *NS* as NEO's and also mark each candidate which does not precede any other candidate with greater *NS* as NEO.

More detailed description of the above steps follows.

Adepts Selection `[lines 1716 - 1737]`

- First, the blur filter [0, 1, 0; 1, 2, 1; 0, 1, 0]/6 is applied to the image. In the next text, I will denote the image data value of the pixel obtained after this step by *I*. [Blurring with this simple filter gives better score then no blurring; I did not have time to try another filters.]
- In the next step, in each area of the square grid with square side length equal to `const int MEDIANGRID=64`, the median of *I* is calculated. For each area, the average of the medians of that area and the neighboring 8 areas is calculated and taken as the *basic background value* (*BBV*) for that area.
- Then a simple shape detection is made: *Shape* is an 8-connected region composed of pixels with value more than `const int SHAPEOVERMEANLIMIT=200` over the *BBV* of the area in which the pixel lies. Each shape stores its *x*-scale (that is the length of the projection of the shape into the *x*-axis), *y*-scale (analogous definition) and *n*-scale (the total number of pixels in the shape).
- The algorithm looks for *peaks*, that is, the pixels which are local maxima of *I*. To each pixel, its *peak class* (*PC*) is assigned. If the pixel is 4-connected to a pixel with higher *I*, its *PC* is 0. Otherwise, its *PC* is at least 1. Consequently, the pixels in the neighborhood are tested and the algorithm checks whether the value of *I* decreases along the rays starting in the pixel. The value of PC is more or less the number of layers how far the value of *I* decreases. The code is self-explanatory `[lines 1579 - 1634]`.
- Now, the pixels of the image are examined:
  - If PC ≤ 2, pixel is not taken as an adept (is *discarded*).
  - If PC ≤ 4 and ($I - BBV < 46$ or $I - BBV > 8813$), the pixel is discarded.
  - If $I - BBV < -67$ or $I - BBV > 24875$, the pixel is discarded.

- Let the pixel be part of a shape from the shape detection. This shape has got some x-scale, y-scale and n-scale, which are now assigned to the pixel (if the pixel is not inside any shape, these values are set to 0).
  - If x-scale > 77, the pixel is discarded.
  - If y-scale > 474, the pixel is discarded.
  - If n-scale > 4642, the pixel is discarded.
  - If (x-scale + 1)/(y-scale+1) < 0.1621 or (x-scale + 1)/(y-scale+1) > 3 (that is, the shape is proportionally too tall or too wide), the pixel is discarded.

[All the parameters used to discard the pixels were set by exploring the provided 100 training sets. In the first iteration, the only criteria for discarding was PC ≤ 4. Later, I changed the criteria in such a way, that all the detections discovered in the first iteration will still fit into the constrains. But since I wanted to give a chance also to some pixels with PC ≤ 4, I let them pass if the value of $I - BBV$ was in the interval where 98% of the values of the detections discovered in the first iteration lay, i. e., the interval from 46 to 8813. This condition was necessary to avoid having very large number of adepts for further investigation.]

- Pixels, which passed the previous test, are taken as adepts. For each adept, the position inside of the image is recalculated from integer coordinates (x, y) into more accurate coordinates using the least square method: The values of I of the five points $(x-2, y)$, $(x-1, y)$, (x, y), $(x+1, y)$, $(x+2, y)$ are approximated with the parabola and the x-coordinate of the peak of the parabola is taken as the exact x-coordinate of the adept. The similar calculation for y-coordinate is made. These new coordinates are used for conversion into RA/DEC coordinates.
- To be able to get adepts from some area quickly, not only the list of adepts is stored (in variable `vector <Adept> adept[4]`), but also each pixel stores information what adept, if any, is around (variable `vector <int> getAdept[4]`). More over, the image is subdivided into the square grid with square side length `const int ADEPTGRID=50` and each area stores a list of adepts located in it (variable `vector <vector <int> > adeptList[4]`). For each of the four images, only some of these variables are being used later, but it was easier to implement it in such a way that for all four images everything is stored.

Candidates selection `[lines 1739 - 1867]`

Up to this step, we have dealt with the images separately. We are left with the four lists of adepts and we wish to combine them to form candidates for detections.

- First, the algorithm looks for the "stars", that is, adepts, which seem not to "move". The RA/DEC coordinates of each adept from the first image are projected into x/y coordinates of the remaining 3 images and the algorithm looks for the adepts in some neighborhood of these projections. If there is an adept in each image with the RA/DEC distance less than `const double STARLIMIT=0.0005`, then the adept from the first image and the corresponding adepts from the other images are labeled as "stars" and are prohibited to be combined as candidates.
- Each of the remaining adepts from the first image is paired with each adept from the last image which lies in the RA/DEC distance at least `const double MINDIST=1e-3` but no more than `const double MAXDIST=0.03`. These two adepts form a segment in the RA/DEC space. Let P_2 and P_3 be the points which divide this segment into thirds. If there is an adept in the second image with distance from P_2 less than `MAXRD` and there is an adept in the third image with distance from P_3 less than `MAXRD` (where `const double MAXRD=0.001`) then these four adepts are "checked" by the `checkCandidate` method. This method inspects whether the four adepts are suitable for candidate. Denote the four points in RA/DEC space where the four adepts lie as A_1, A_2, A_3, A_4. These points form three segments A_1A_2, A_2A_3 and A_3A_4. To pass the test, the ratio of the shortest to the longest among

these segments must be at least `const double NORMRATIOLIMIT=0.4`. Also, for each of the three segments, its directed angle is calculated. To pass the test, the difference between any two of these three angles must be less than `const double ANGLEDIFLIMIT=0.8` radians. In case the 4-tuple passes the test, it is selected as a candidate for the detection. The maximum difference of the three mentioned angles – *angleDif* – is stored as one of the features.

Features `[lines 1168 - 1199]`

In the previous step, the list of candidates for the detection was created, that is, the list of 4-tuples of adepts. To each candidate, the following features are assigned, using the notation from the preceding text (together 32 features, 0-based numbering; "average" means arithmetic mean of the values corresponding to the 4 adepts; "span" means the difference of the maximum and minimum of these 4 values):

0. length of the segment A_1A_4;
1. average of RA's;
2. average of DEC's;
3. average of x-coordinates;
4. average of y-coordinates;
5. average of PC;
6. average of $I - BBV$;
7. average of x-scales [not used in the final submission];
8. average of y-scales;
9. average of n-scales;
10. average of (x-scale + 1)/(y-scale+1) [not used in the final submission];
11. sum of the squares of perpendicular distances of A_1, A_2, A_3, A_4 from the line obtained by fitting these 4 points with the total least squares method;
12. slope (as an angle between $-\pi/2$ and $\pi/2$) of the line obtained in the previous step;
13. feature[11] divided by the square of the length of the segment A_1A_4;
14. directed angle of the segment A_1A_4;
15. directed angle of the mirror image of the segment A_1A_4;
16. span of PC;
17. span of $I - BBV$;
18. span of x-scales [not used in the final submission];
19. span of y-scales;
20. span of n-scales;
21. time difference between the 1st and the 4th image (extracted from headers);
22. length of the segment A_1A_4 divided by feature[21], i. e., speed of the object;
23. modified julian date (extracted from headers) modulo 365.25, i. e., "season" of the year;
24. average of zenith distances of the first and the last image (extracted from headers);
25. minimum of the lengths of the segments A_1A_2, A_2A_3 and A_3A_4 divided by the length of the segment A_1A_4;
26. maximum of the lengths of the segments A_1A_2, A_2A_3 and A_3A_4 divided by the length of the segment A_1A_4;
27. distance of A_2 from P_2;
28. distance of A_3 from P_3;
29. feature[27] divided by the length of the segment A_1A_4;
30. feature[28] divided by the length of the segment A_1A_4;
31. *angleDif*.

Detection Check `[lines 1200 - 1233]`

The detection is considered to be a "double detection", if there is another detection which differs in each of the four x- and y-coordinate by less than `const int NEARLIMIT=8` `[lines 1889 - 1898]`. During the training phase, each candidate for the detection is checked whether it actually is a detection, or even NEO and/or double detection. Each candidate is compared to each detection, with the code equivalent to the code in the provided visualizer. Additionally, if it passes this test with some detection, another test is examined: Let the detection corresponds to the four points D_1, D_2, D_3, D_4 in the RA/DEC space. To pass the test, the difference of the oriented angles of segments A_1A_4 and D_1D_4 must be less than $\pi/18$ (=10 degrees) and the ratio of the lengths of the segments A_1A_4 and D_1D_4 must be between 0.8 and 1.2. If this test is passed, the candidate is marked as a detection (and possibly NEO and/or double detection – according to the detection via which it passed). [The extra test was added because otherwise often several candidates pass via the same detection – the official test was simply too generous.]

Random Forests Setup

The possible parameters for the random forests are:

Parameter description	Value in final submission
Number of trees	200
Number of random features used to split nodes	6
Stopping criteria	`MINNODE=1, MAXLEVEL=50`
Cost function used to split nodes	$\sqrt[3]{x(1-x)}$

`MINNODE=1` means that the tree will grow up to the nodes of size 1, which means there is no stopping criteria based on node size. `MAXLEVEL` is the maximum depth of the tree. I did not check if this level was ever reached, so maybe this stopping criteria is redundant. I did not optimize these parameters.

I did not use different setup for each of the three separate forests. That is, the same set of features was always used as well as the same parameters described in the table above. [For each forest and each feature, the algorithm displays "feature score" in the standard error stream, that is, each time a feature was used in some node splitting, its score is increased by the size of the node. Seeing these scores during developing the solution, I decided to remove three features from the list, as is marked in the feature section.]

Getting the Final Answer `[lines 2028 - 2104]`

- Each candidate obtained during the testing phase is evaluated on each tree of each random forest. The score is the percentage of the positive answers appearing among all the answers (that is, a number between 0 and 1). The candidates are sorted by DS in descending order.
- The threshold 0.04 for the value of *DDS* is used to determine whether the candidate will appear twice in the answer. [I just looked at these scores for the one test case which was provided and discovered that using this threshold, the number of double detections would be approximately such as desired.]
- The number of expected NEO's (E) is calculated as the number of found NEO's among the training candidates multiplied by the ratio of the sizes of testing candidates set and training candidate set. E candidates with the highest *NS* are marked as NEO's and also each candidate which does not precede (with respect to *DS*) any other candidate with greater *NS* is marked as NEO.

Overview of impementation

Lines of code	Description
1 to 22	Includes and preprocessor definition of the cost function $G(x)$ used to split nodes and some other functions
32 to 64	Definition of constants and parameters; the values for these parameters were selected by several trials and observing the results (on the one provided example test), no special science was used for optimizing them
66 to 106	Declaration of global variables
109 to 155	General methods like random generator, splitting the text by space, measuring time
157 to 174	Total least squares method
176 to 188	Method generating a random sample of features
190 to 224	Definition of the classes used in shape detection
225 to 315	Method performing the shape detection
317 to 501	RA/DEC to XY and vice versa convertor (translated from the provided visualizer)
504 to 593	Relic method for peak detection, not used in final submission
595 to 609	Method for blurring the image
611 to 663	Method for calculating the *BBV*
665 to 727	Relic methods, not used in final submission
728 to 730	Method for getting the *BBV* value at the specified position
732 to 793	Method only returning the array of *BBV*'s for every pixel (it was also used for a different approach of calculating *BBV*, so there is some code unused in the final submission wrapped in the `if(MEANING){...}` part)
796 to 981	Methods used for offline visualizations, not used in final submission
983 to 1055	Definition of the `class Detection`, used to read the lines containing information about detections and to store the detections; it contains also methods for offline visualizations, not used in final submission
1057 to 1076	Definition of the `class Adept`, used to store the adepts
1077 to 1093	Method for final checking of the 4-tuple of adepts
1094 to 1100	Method for calculating of the span of four values
1101 to 1253	Definition of the `class Candidate`, used to store candidates, assign features and check whether the candidate is a detection
1254 to 1426	Ugly implementation of comparing methods, used to sort a sample at a specific node by a specific feature, to easily find the best splitting
1427 to 659	Definition of the `class Sample`, used to store a sample of detections at a specific node when building a tree
661 to 1454	Definition of the `class Node`; each tree of the forest is a collection of these nodes; each node stores the id's of the left and right node below it, feature used to split this node, value of that feature used to split this node, level of this node and number of positive and negative answers among the candidates in this node

1455 to 1549	Definition of the `class Tree`; including recursive method `clusterAtNode` used for calculating the answer of a tested candidate on a particular tree and the recursive method `divideNode` used to build the tree. When splitting a node, the sample is sorted by every tested feature and split by every possible intermediate value. The best splitting is stored and performed at the end. To ease the implementation, all the features are stored as doubles, even if some of them are naturally only int's.
1551 to 1571	Function for building one tree; i.e., it selects random sample of candidates, creates a tree with one node and call the recursive method `divideNode` described in the previous cell
1573 to 1679	Method returning the list of adepts in the given image
1681 to 1697	Method used for extracting the header information
1699 to 1872	Method used for extracting the candidates both in training and testing phase
1874 to 2105	Definition of the `class AsteroidDetector`. In first call of the `trainingData` function, some constants used for random forests generation are calculated `[lines 1906 - 1972]`. In first call of the `testingData` function, the random forests are grown `[lines 1980 - 2017]`.