**Marathon Match - Solution Description**

1. **Introduction**

* Handle: nofto

1. **Solution Development**

After the first round, in which I finished 7th, I was sure I cannot beat wleite’s winning solution using my previous approach. Having some experience with wleite’s code from the Lung Cancer Round 2, it was natural to start with his solution also in this match. My idea was to improve it. Sadly, almost nothing I tried was beneficial. Here is the list of things which *did not* work (i. e., did not show improvement in the provisional score)

* Changing the returned shapes from convex hulls to non-convex
* Adding non-convex shapes to the convex hulls (that is, doubling the number of shapes which are processed in the second training phase)
* Training on multiple cities, that is, trying to build a single model for all cities.
* Adding new shape features (ratio of original area and convex hull area, distance from the image border, number of detected shapes in the current image, ratio of original perimeter and convex hull perimeter)
* Adding new “global” pixel features (the original methods rectStatFeatures() and text() applied to the entire image frame)

1. **Final Approach**

I will describe only the differences between the original wleite’s and my solution. All these are only minor changes (or changes implied by the difference in data format between both rounds).

* In the first round, there were 2 kinds of images, denoted by *3band* and *8band*. The problem statement says that in the second round, *RGB-PanSharpen* corresponds to *3band* and *MUL* corresponds to *8band*. However, I found out that better score is obtained when *MUL* is replaced with *MUL-PanSharpen* (around 10% gain). Since the resolution of *MUL-PanSharpen* images is the same as the resolution of *RGB-PanSharpen* (which was not the case for *8band* and *3band* images), it is necessary to adjust lines 28-29 in BuildingFeatureExtractor.java and lines 151-152 in PolygonFeatureExtractor.java.
* The original solution uses entropy as impurity function for splitting nodes in random forest. I replaced it with sqrt(x\*(1-x)) – it should be computationally less intensive and in the past I got better results in several problems with this function when compared to entropy and/or Gini impurity. In this problem, the results were only slightly better with my function (1% gain), and I only compared it on one city.
* The original solution uses 60% of data for the first phase of the training, 30% for the second phase of the training and 10% for offline testing. There is no sense to leave any data for offline testing in the final tests, so my split is 65/35/0 instead of 60/30/10.
* I believe there is a small bug on line 89 of the original Util.java (line 92 of my version) – there should be “buildingsPolyBorder2” instead of “buildingsPolyBorder”.
* I change the code so that it fulfils the requirements for final testing.

1. **Open Source Resources, Frameworks and Libraries**

My solution does not use any library or open source resource different from the one used by wleite in the first round. The jar files located in the “lib” directory are probably needed only to work with TIFF format, and I do not know its origin – I downloaded it as a part of the first-round winning solution.

1. **Potential Algorithm Improvements**

I have no clear idea how the approach may be improved. I tried several things (mentioned above) which did not help.

1. **Algorithm Limitations**

The results are good only if you test on the model which was trained on the same city. If you will detect buildings on a new city without training data, only with one of the four city models, the results will be poor.

1. **Deployment Guide**

Follow the guide from the wleite’s solution of the first match. I did not add any new source files, I only changed some of the original files.

1. **Final Verification**
2. Create directory structure as in the zip file in <https://www.dropbox.com/s/iov4wsgmutxt7ko/nofto-docker.zip?dl=1>
3. Execute “BuildingDetectorTrainer <directory>”, which will produce a serialized random forest files named “rfBuilding.dat”, “rfBorder.dat” and “rfDist.dat” in the corresponding models/city<n> directory.
4. Execute “PolygonMatcherTrainer <directory>”, which will produce a serialized random forest file named “rfPolyMatch.dat” in the corresponding models/city<n> directory.
5. Execute “BuildingDetectorTester <directory> <output file>”, which will produce the expected CSV file.