Topcoder Marathon Match – Quake Predictor - August/2015

Solution Description - Wladimir Leite (wleite) – 2nd place

Comments about the match

Before starting talking about the solution itself, I would like to note that I joined very late in the match (5 days before the deadline). In addition, I found very hard to deal with such huge amount of data (about 800 GB).

After downloading and unpacking the data files, which took me a while, each local attempt to run my solution required a lot of time, mainly because of I/O (reading data files from disk). I changed the provided visualizer in order to optimize things a little bit, but even with the changes, even a simple solution took few hours to complete the training/testing cycle in my environment.

I tried to use threads in training part, but that caused serious I/O concurrence, so I rolled back to use one single thread. Maybe a large enough SSD volume could help here (my SSD disk is too small and I used a regular HDD to store the provided data).

Probably that would have been a good match to use 3 or 4 weeks… I guess other competitors also had a hard time dealing with such huge data files.

Solution

I tried very few ideas, so my final solution is very simple and similar to the first attempt.

First, I randomly split the data files in two sets, about the same size, one for training and the other for testing. My idea was to train my final submission using all data files, but I didn't have enough time to do that.

Training phase extract features from each hour, for each site, and build a random forest of classification trees which output is the probability of an EQ event happen in one of the next 60 days or not happen at all, in the given site. In other words, it has 61 classes, 0 – means no earthquake in the given site in the next 60 days, 1 – EQ will happen in the next day, 2 – EQ in two days from now, and so on.

Initially I used a binary output, just to predict if an EQ will happen or not in a given site (my first submission). Then I divided the period in groups of few days. Finally, I used 1 day as the minimal unit. I didn't test shorter periods (6 or 12 hours, or even 1 hour), but I suspect the variance is so big that it would be useless.

About the selected features, I started with a very simple analysis of magnetometer signals: for each site and each channel, it counts the number of "peaks" within the hour. These peaks are only sequences where the raw data have 4 consecutives increases followed by 4 consecutive decreases. In the last days of the competition, I tried more complex peak-detection system, including the duration and amplitude of the signals, but everything gave much worse results. Probably I missed something here.

I also used a combined value, with the sum of 3 channels, and the planetary magnetic activity index at the given hour. So there are 5 basic features in my solution: 4 from the number of peaks (one for each of 3 channels and 1 combined) and the planetary activity.

To detect some change in the magnetic behavior, that could help predicting the EQ events, I compared values from the 5 basic features with the average values of the same feature in last *N* hours, at the same site. These differences, between the current value and the previously seen, are used as new features. The periods used to calculate previous averages were (in hours): 6, 12, 24, 48, 72, 120, 240 and 480.

Before doing anything with signal values, if the sample rate is 32, I converted it to 48, creating an extra virtual value between each pair of given values, with the average of them. That conversion target is to have similar frequencies (48 and 50) for all used data, but in practice, I could not see any difference, considering the simple set of features that used.

Testing phase uses the generated random forest to predict the probabilities for each site and each hour. For the first 720 hours, it just collects values that are used in averages calculation. Then, after hour 720, the output from prediction are added to previous returned value for a given site and hour, in a way that final values are averages of all predictions so far.

Implementation

I coded it in Java, using 3 source code files:

- **QuakeTester.java:** provided tester, with few changes to improve I/O performance, and to make testing and training through direct function calls.
- **QuakeTrainer.java:** code to build the random forest and other auxiliary functions to convert the generated trees to a binary file, and to convert that file to packed string that can be inserted as source code constant.
- **QuakePredictor.java:** this is the submission file, with the features extraction code and hard coded strings with a packed version of the pre trained random forest.

The steps to run the whole cycle is (given the data folder is in "d:/data"):

1) `java –cp bin QuakeTester train d:/data`

This first step will generate a binary file, with all random forest data, including the number of trees, splitting variable index and values and leaf output values, named "qfq.dat".

2) `java –cp bin QuakeTester convert d:/data`

The second step converts the "qfq.dat" to a packed string that can be included in the code (in a way that it can be used in TC server environment). The output of conversion process must be copied and pasted manually into QuakePredictor.java, and contains the value of three variables: "blen", "zlen" and "data". Internally it uses Inflater class from Java API and a custom encoding, in which each 4 bytes are converted to 5 characters using a 92-base system, with only printable characters. The number of trees was adjusted in this step. Final submission had only 14 trees, because of 1 MB submission limit.

3) `java –cp bin QuakeTester test d:/data`

This is the final step, used for local tests. Although the variance between test cases were huge, usually better local solutions gave me better provisional scores.

Conclusions

As stated in the initial comments, my solution is very simple, and my idea was to expand it, but as each new attempt took a lot of time, I didn't manage to go very far.

About the presented question about the highest score I could have reached without using the magnetometer signal, in my case doesn't make too much sense, since from my first attempt, I relied in magnetometer signals (although only a very simple peak count). I didn't used global quakes and site location information anywhere in my solution, didn't even tried to used them, and probably it would be possible to extract some information from them.