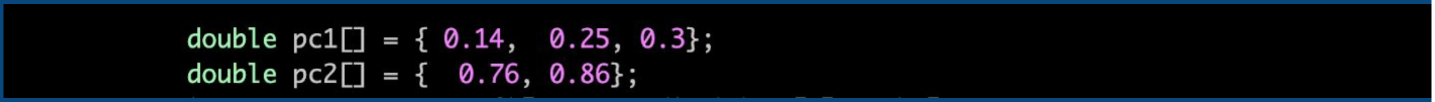
Topcoder

CMAP: DPeak Challenge

3rd Place Solution by mkagenius

High level idea of the solution

What finally worked was the ability to control the choices of random points selected during the initialization of K-Means algorithm.



```
double pc1[] = { 0.14,  0.25, 0.3};
double pc2[] = {  0.76, 0.86};
```

pc1[] are the choices for peak 1 while pc2[] are the choices for peak 2 in percentage terms. Earlier I was using python’s KMeans from scipy package but it does not allow you to choose the initialization points, so you have to try many more initialization points to arrive at a good one.

In addition, avoiding pandas and switching to pure c++ parsing of the file did help in the speed-up of the solution.

Details [train, deploy, run]

Train

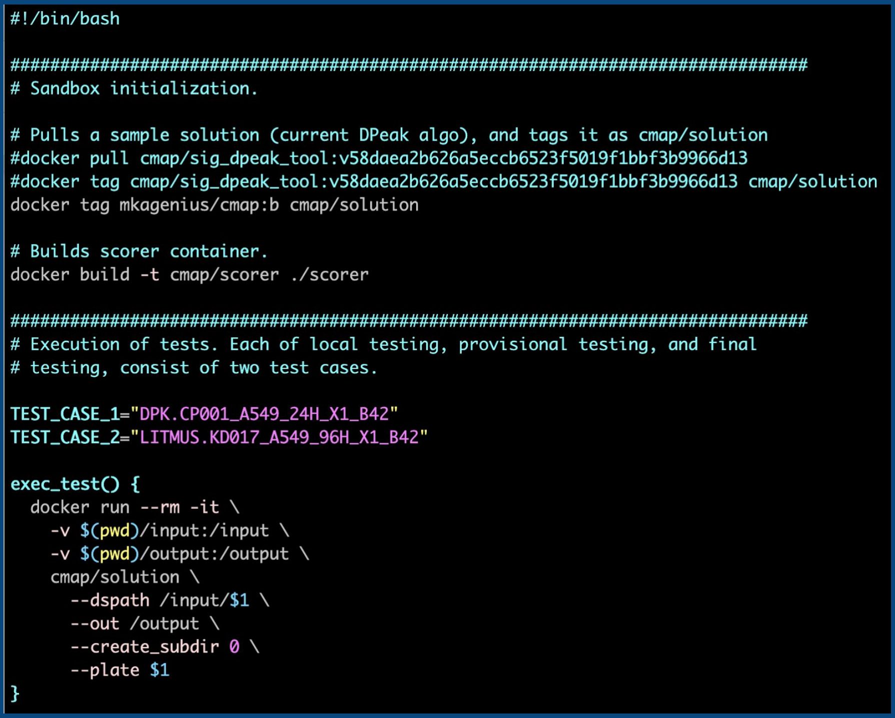
There is no training needed. No machine learning model here. Pure K-Means based on some initial split points.

Deploy

Method 1

We were required to dockerize the submission. The same dockerized solution can be used in the same as specified in the contest:

1. Unzip the dockerized final submission by me.
2. Go inside the directory where Dockerfile is located.
3. `$ docker build -t mkagenius/cmap:b .`
4. Now use the above tag name ‘mkagenius/cmap:b’ in the run.sh file provided by the contest organisers.
5. Snapshot of run.sh below:



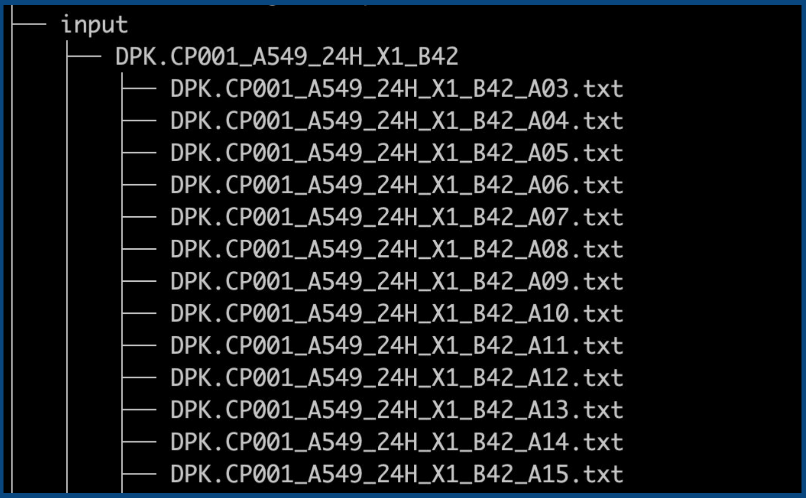
```
#!/bin/bash

################################################################################
# Sandbox initialization.

# Pulls a sample solution (current DPeak algo), and tags it as cmap/solution
#docker pull cmap/sig_dpeak_tool:v58daea2b626a5eccb6523f5019f1bbf3b9966d13
#docker tag cmap/sig_dpeak_tool:v58daea2b626a5eccb6523f5019f1bbf3b9966d13 cmap/solution
docker tag mkagenius/cmap:b cmap/solution

# Builds scorer container.
docker build -t cmap/scorer ./scorer

################################################################################
# Execution of tests. Each of local testing, provisional testing, and final
# testing, consist of two test cases.

TEST_CASE_1="DPK.CP001_A549_24H_X1_B42"
TEST_CASE_2="LITMUS.KD017_A549_96H_X1_B42"

exec_test() {
  docker run --rm -it \
    -v $(pwd)/input:/input \
    -v $(pwd)/output:/output \
    cmap/solution \
      --dspath /input/$1 \
      --out /output \
      --create_subdir 0 \
      --plate $1
}
```

6. After that you just run the `$ run.sh` command since run.sh file already has the parameters required.
7. Refer the competitor pack if you want: https://drive.google.com/open?id=1865JSqF2NqxOcEMcDV57a10k6YTWRm6m

Method 2

In case you do not want to depend on docker and just want to run the solution. Follow these steps.

1. Get my dockerized zip file and unzip it. It contains “script.cpp” file we will use.
2. Setup the input folders according to the following structure



```
├── input
│   ├── DPK.CP001_A549_24H_X1_B42
│   │   ├── DPK.CP001_A549_24H_X1_B42_A03.txt
│   │   ├── DPK.CP001_A549_24H_X1_B42_A04.txt
│   │   ├── DPK.CP001_A549_24H_X1_B42_A05.txt
│   │   ├── DPK.CP001_A549_24H_X1_B42_A06.txt
│   │   ├── DPK.CP001_A549_24H_X1_B42_A07.txt
│   │   ├── DPK.CP001_A549_24H_X1_B42_A08.txt
│   │   ├── DPK.CP001_A549_24H_X1_B42_A09.txt
│   │   ├── DPK.CP001_A549_24H_X1_B42_A10.txt
│   │   ├── DPK.CP001_A549_24H_X1_B42_A11.txt
│   │   ├── DPK.CP001_A549_24H_X1_B42_A12.txt
│   │   ├── DPK.CP001_A549_24H_X1_B42_A13.txt
│   │   ├── DPK.CP001_A549_24H_X1_B42_A14.txt
│   │   ├── DPK.CP001_A549_24H_X1_B42_A15.txt
```

3. Compile the script.cpp file, `$ g++ -std=c++11 -pthread -o myapp script.cpp`

4. Run the executable file ‘myapp’ with the four parameters: `--dspath PATH/TO/INPUT/DATASET/FOLDER`, `--out /PATH/TO/OUTPUT/FOLDER`, `--create_subdir 0`, and `--plate TEST_SET_NAME`.

(note: Essentially the same thing was being done by run.sh)

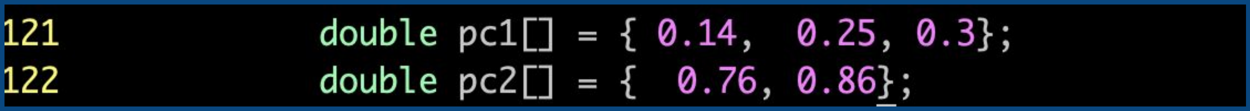
Run

Didn’t we run it already? Yes, the above step did run the solution as well. The output will be in the output folder specified in the parameters.

Tuning the script

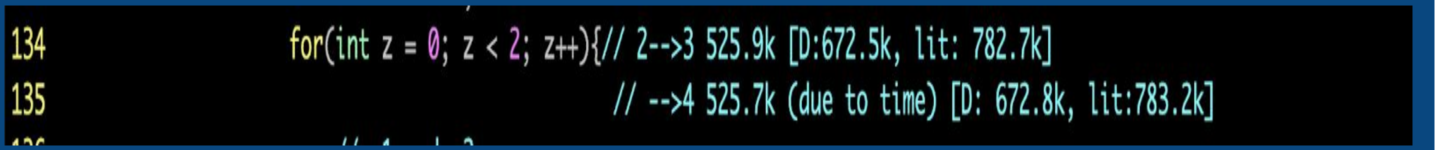
I want to highlight some aspects of the script which were tuned to get the final score.

1. Benefit of writing the KMeans algorithm was controlling the initial random points. I experimented with lot of other values and settled with the following which gave the best score locally -



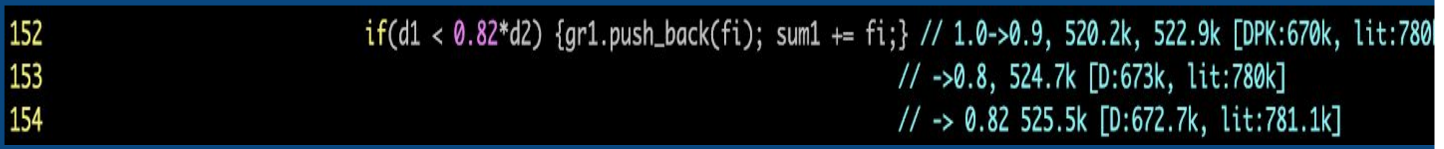
```
            double pc1[] = { 0.14,  0.25, 0.3};
            double pc2[] = {  0.76, 0.86};
```

2. Then to speed up your solution you can use less iteration to converge (only two was enough) -



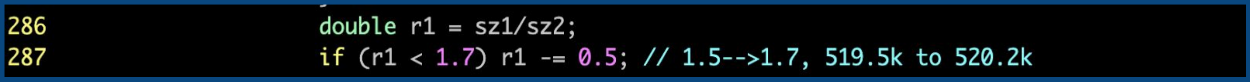
```
            for(int z = 0; z < 2; z++){// 2-->3 525.9k [D:672.5k, lit: 782.7k]
                                  // -->4 525.7k (due to time) [D: 672.8k, lit:783.2k]
```

3. You can also decide the threshold for deciding which group should a FI go into, playing with it did improve my local score -



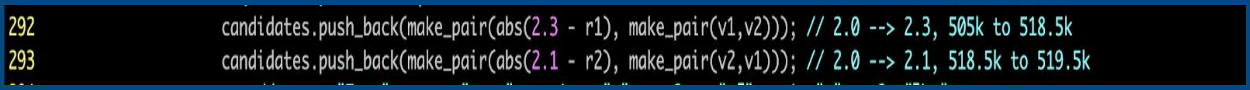
```
                if(d1 < 0.82*d2) {gr1.push_back(fi); sum1 += fi;} // 1.0->0.9, 520.2k, 522.9k [DPK:670k, lit:780
                                                                   // ->0.8, 524.7k [D:673k, lit:780k]
                                                                   // -> 0.82 525.5k [D:672.7k, lit:781.1k]
```

4. Impurity related adjustments: It seemed to me that sometimes the ground-truth was little-off which may be due to some impurities as discussed on the forums of topcoder. So, in cases where the ratio of memberships were < 1.7 , I decided to decrease it by some amount. It helped my local scores.



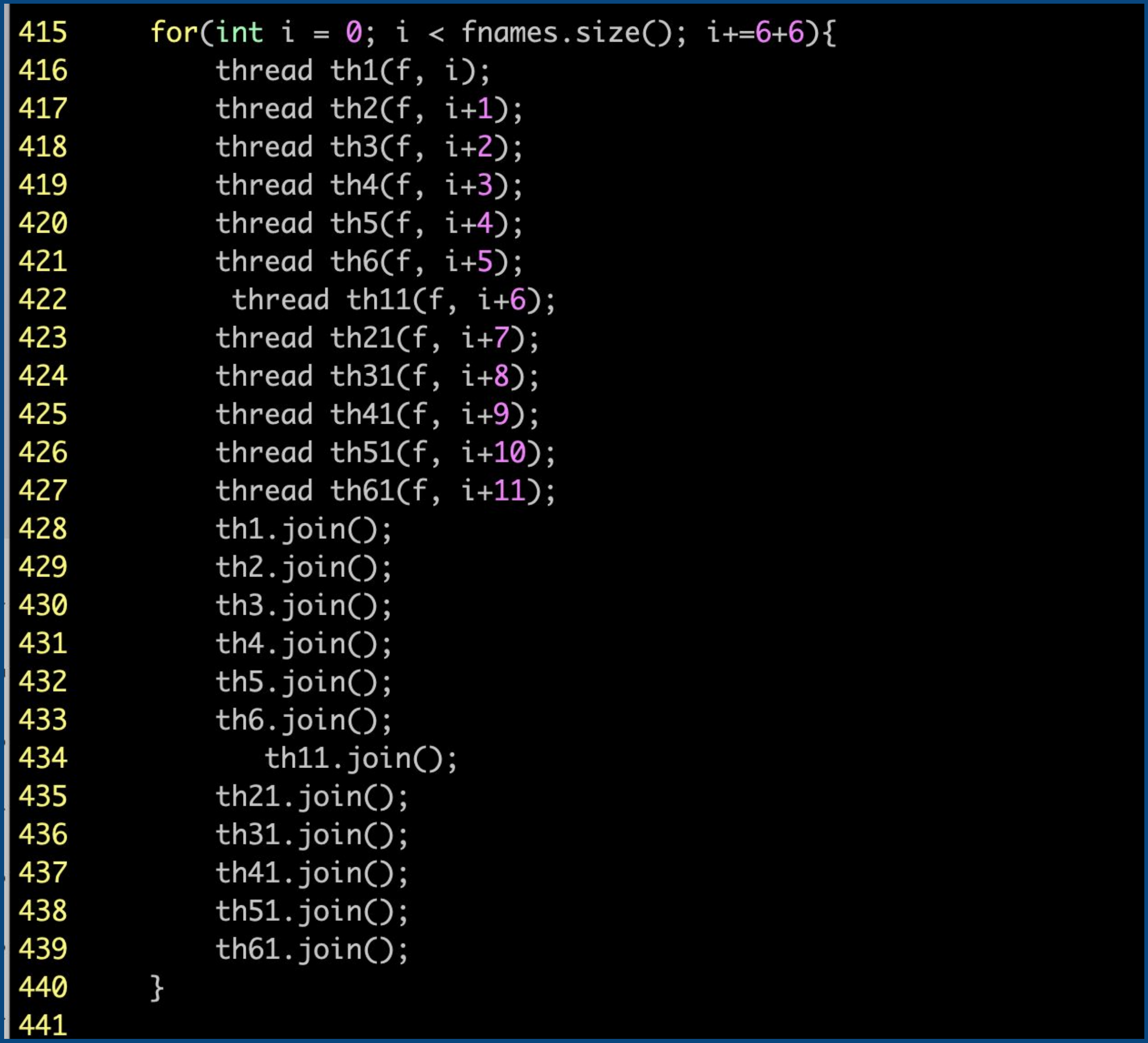
```
double r1 = sz1/sz2;
if (r1 < 1.7) r1 -= 0.5; // 1.5-->1.7, 519.5k to 520.2k
```

5. Finally when the candidates are sorted to choose the closest one to 2.0 value of ratio of membership, I did a small tweak there, in cases where v1 was the lower valued frequency, I assumed the tail/spread of intensity distribution to be longer and hence subtracted it from 2.3 while in case of v1 being higher valued intensity I subtracted it from 2.1 assuming the spread of the distribution to be lower in this case.



```
candidates.push_back(make_pair(abs(2.3 - r1), make_pair(v1,v2))); // 2.0 --> 2.3, 505k to 518.5k
candidates.push_back(make_pair(abs(2.1 - r2), make_pair(v2,v1))); // 2.0 --> 2.1, 518.5k to 519.5k
```

6. Speed up using mutliple threads, I used 12 but you can use more by changing this piece of code:



```
for(int i = 0; i < fnames.size(); i+=6+6){
    thread th1(f, i);
    thread th2(f, i+1);
    thread th3(f, i+2);
    thread th4(f, i+3);
    thread th5(f, i+4);
    thread th6(f, i+5);
     thread th11(f, i+6);
    thread th21(f, i+7);
    thread th31(f, i+8);
    thread th41(f, i+9);
    thread th51(f, i+10);
    thread th61(f, i+11);
    th1.join();
    th2.join();
    th3.join();
    th4.join();
    th5.join();
    th6.join();
       th11.join();
    th21.join();
    th31.join();
    th41.join();
    th51.join();
    th61.join();
}
```

3. Please answer the following questions:

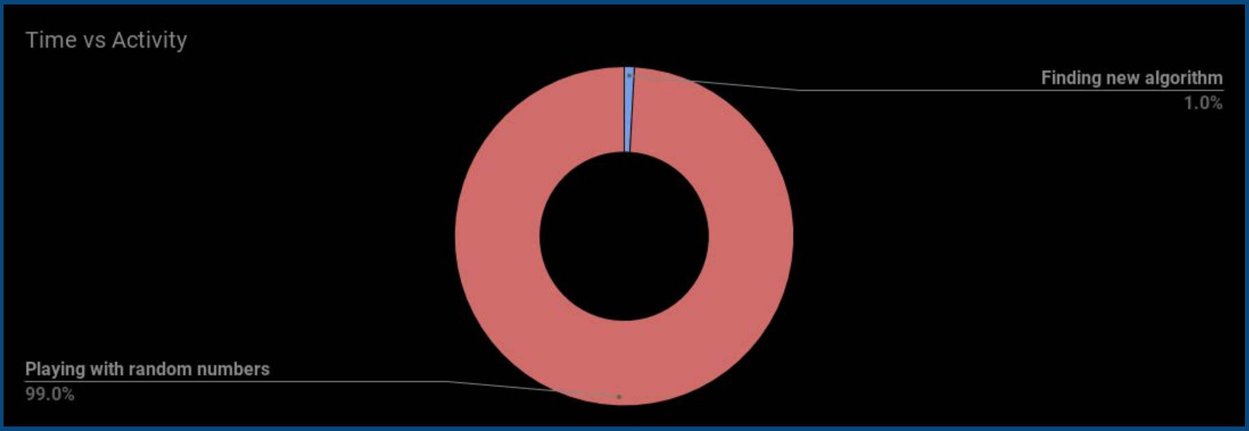
- Country of residence - India

- Highest academic degree achieved- Undergraduate, B.Tech

- If college educated, what was your major - My Major was in Information Technology

- Main motivation for competing in this challenge- My wife told me to be a little productive. Just kidding.

- How difficult/enjoyable the problem and actual competition of this MM was relative to standard MMs on Topcoder?- This is how I spent my time on this contest, See below:



4. Feedback

Very responsive co-pilot - Good. Leaderboard had a little glitch.