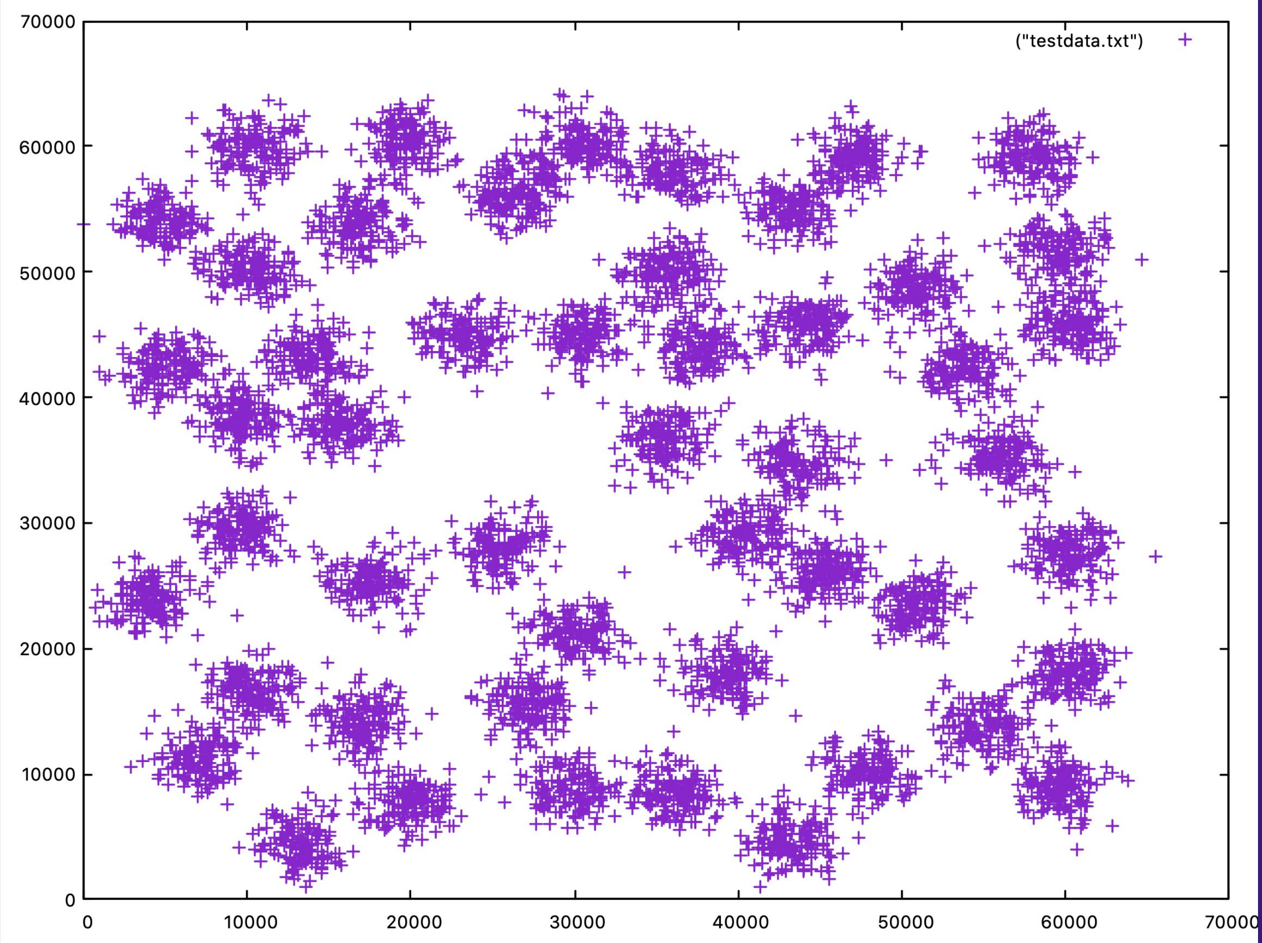
k-means clustering

david pojunas

k-means?



("testdata.txt")
0
10000
20000
30000
40000
50000
60000
70000
0
10000
20000
30000
40000
50000
60000
70000

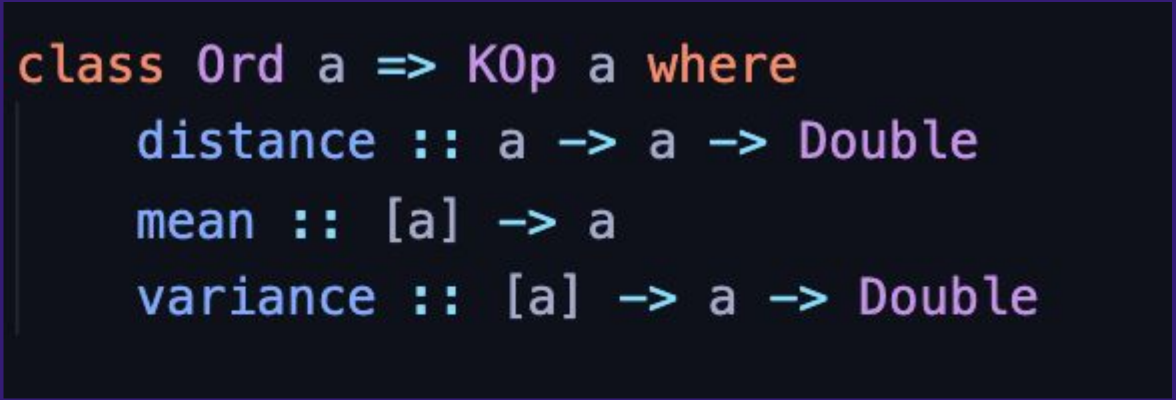
k-means algorithm

- collect some n-dimensional data!
- randomly pick k centroids
- cluster the data into k clusters
  - for each sample, take the minimum distance between the sample and all the centroids
- for each cluster calculate the mean and compare it to the centroids
  - if it's not the same we reiterate the clustering with new centroids (means)

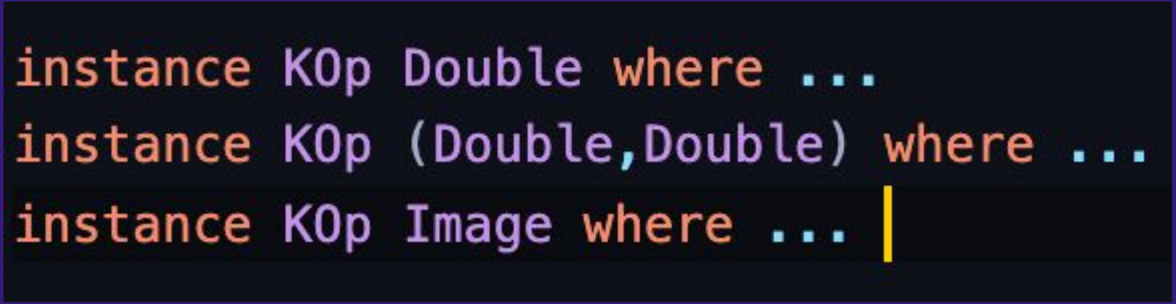
haskell implementation

- abstract out functions for our data types
    - distance function
    - mean function
    - variance function
- write a generic k-means algorithm for any type that implements those functions
- we also need to order our data

type classes are all we need



```haskell
class Ord a => KOp a where
    distance :: a -> a -> Double
    mean :: [a] -> a
    variance :: [a] -> a -> Double
```



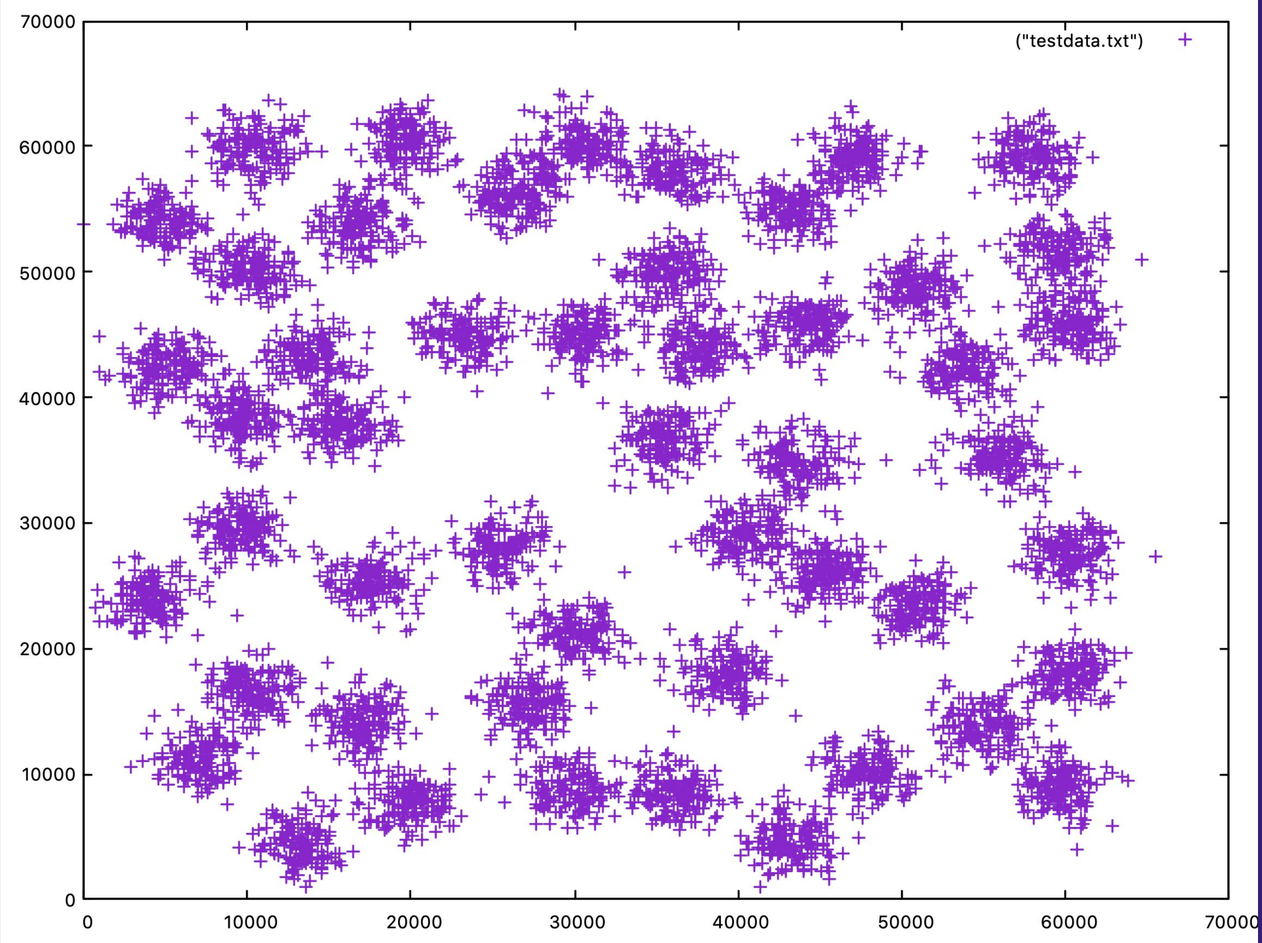
```haskell
instance KOp Double where ...
instance KOp (Double,Double) where ...
instance KOp Image where ...
```

tackling randomness

- randomness requires IO ()
- wanted to avoid IO () in the basic implementation of k-means
- ideally, I wanted the function as follows
    - *kmeans :: KOp a => Int -> [a] -> [(a, [a])]*
- to provide a starting point we needed
    - *kmeans :: KOp a = Int -> [a] -> [a] -> [(a, [a])]*

fold fold fold

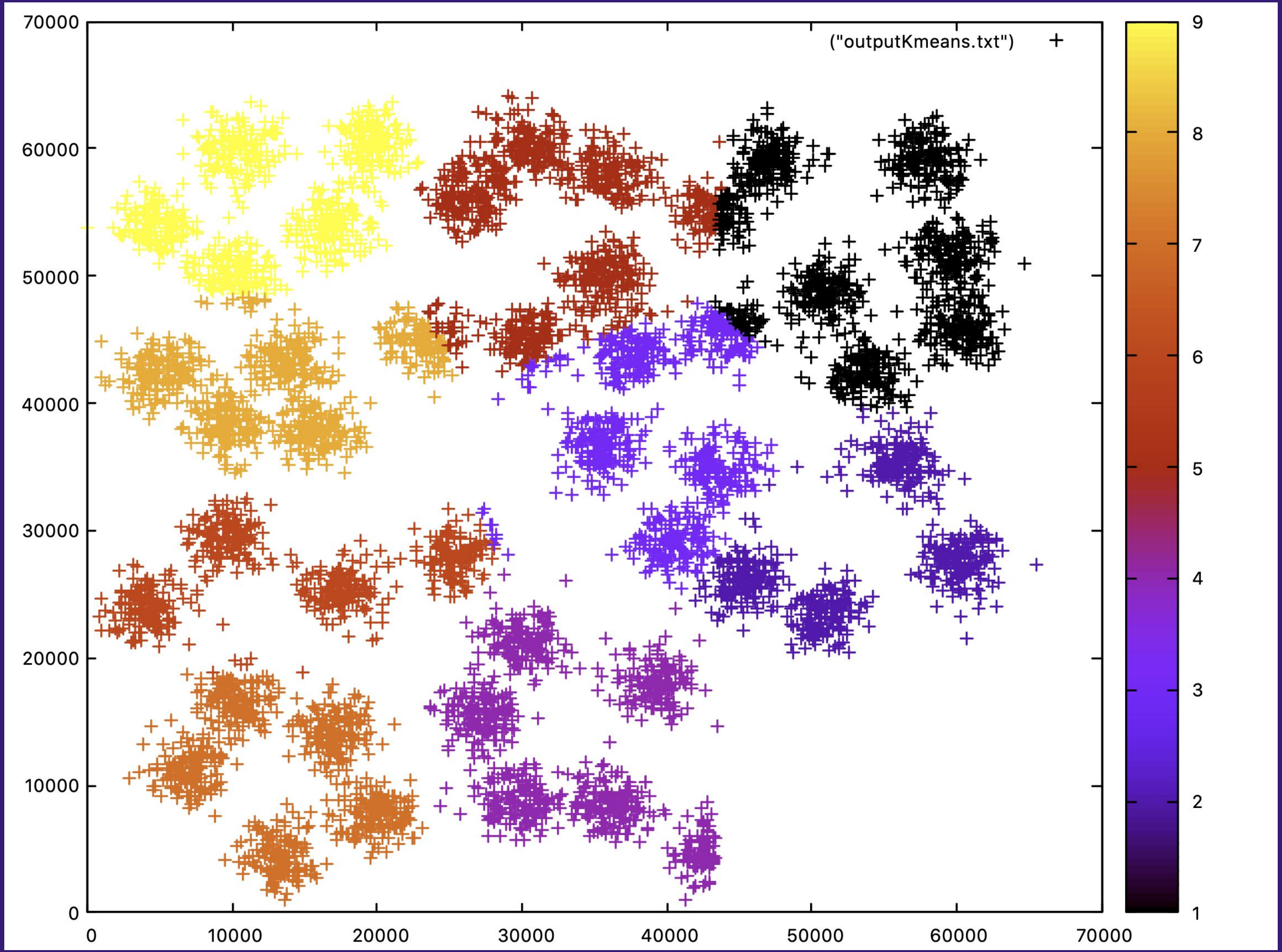
- used maps for the data and the clusters
- for any function of a map, I just folded
    - clustering
    - reclustering
    - calculating means and distances
- never really needed Functor or Monad
- did a little bit of IO for reading and writing data
- used gnuplot for plotting my data...


("testdata.txt")
0
10000
20000
30000
40000
50000
60000
70000
0
10000
20000
30000
40000
50000
60000
70000

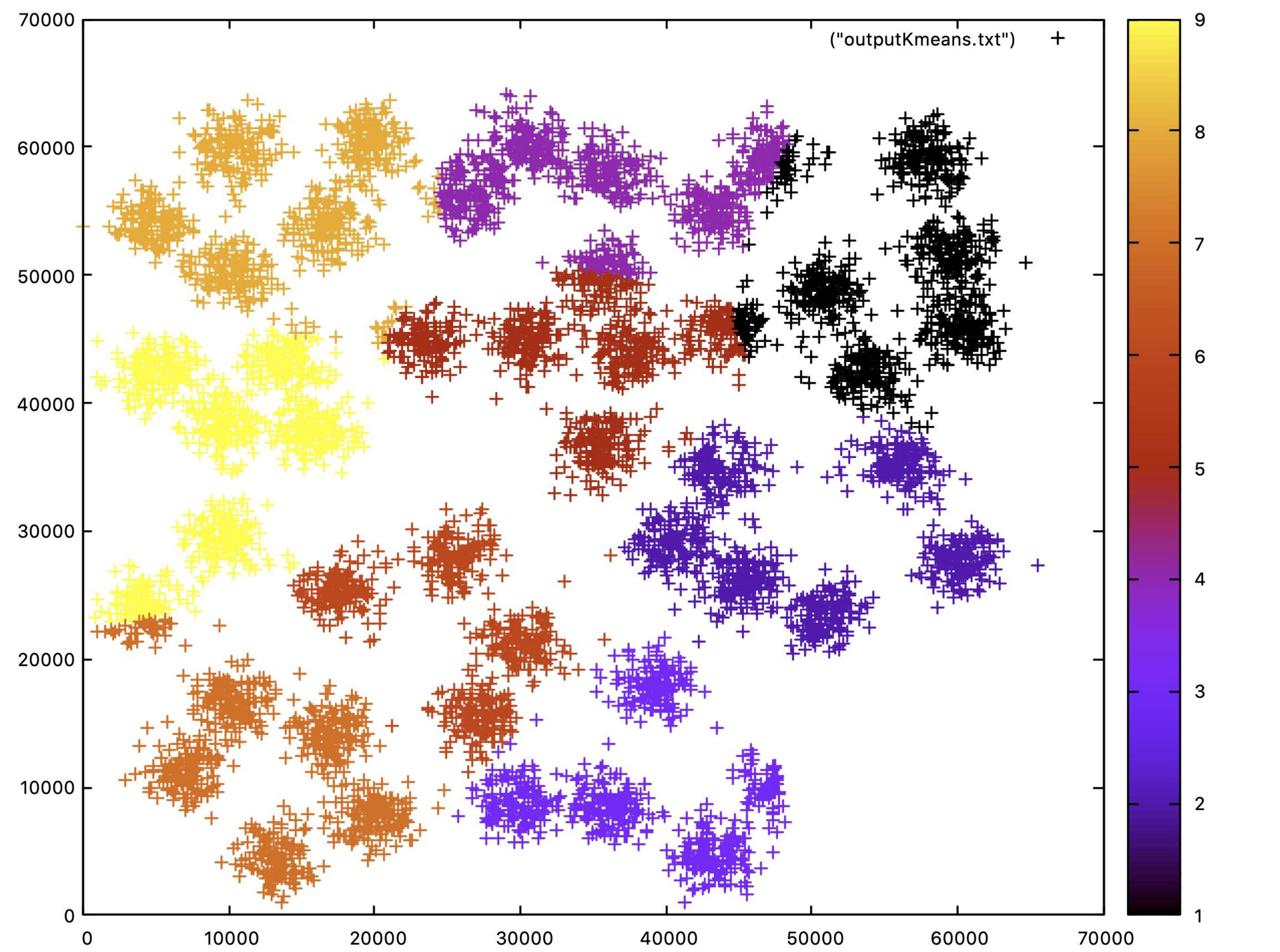
we can do better than just basic k-means

- we can run the clustering multiple times
  - for each iteration we calculate the sum of the variances for each cluster
- then we take the clustering with the smallest variation between each cluster

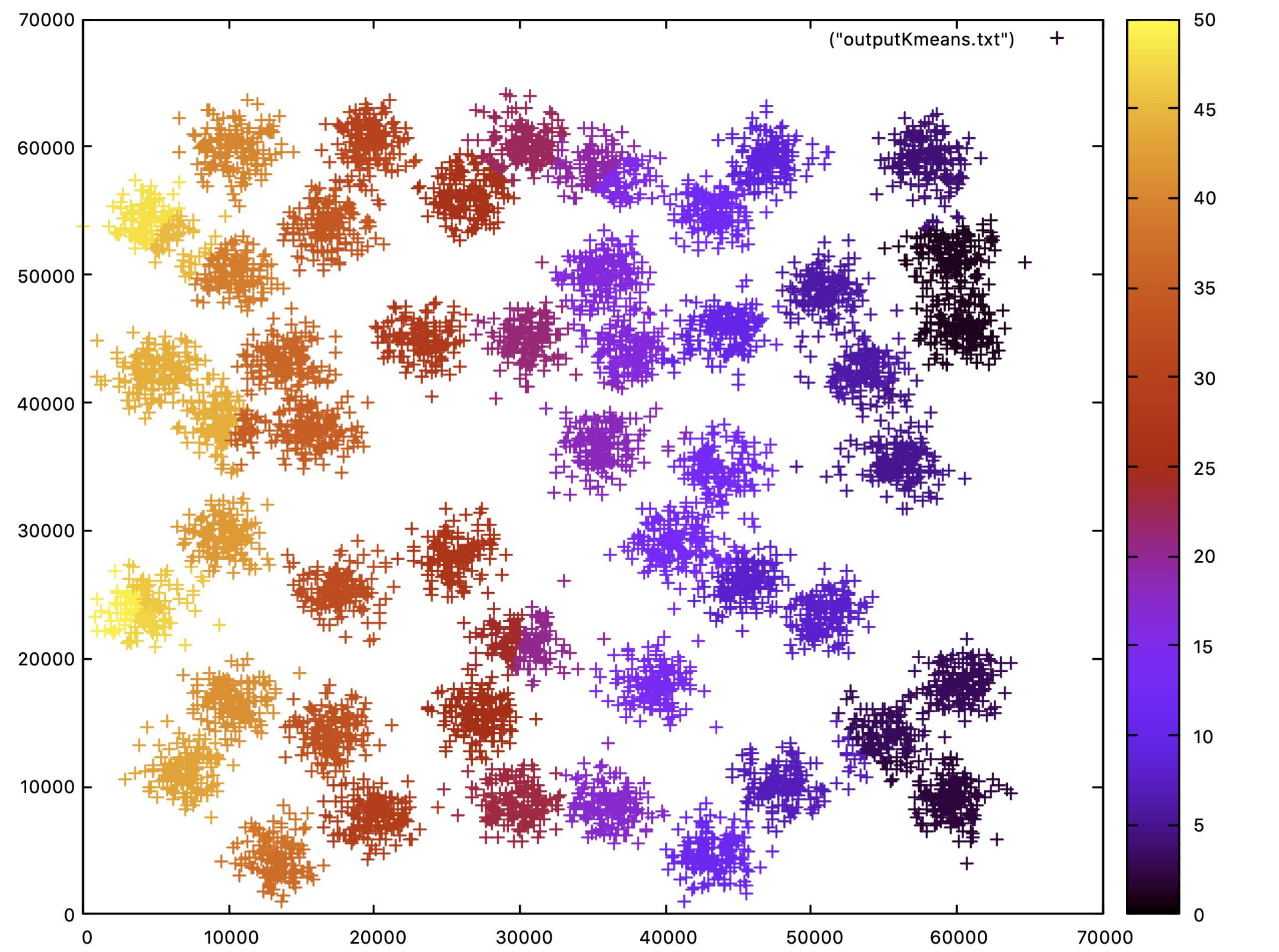
Iter : 1


("outputKmeans.txt")
0
10000
20000
30000
40000
50000
60000
70000
1
2
3
4
5
6
7
8
9

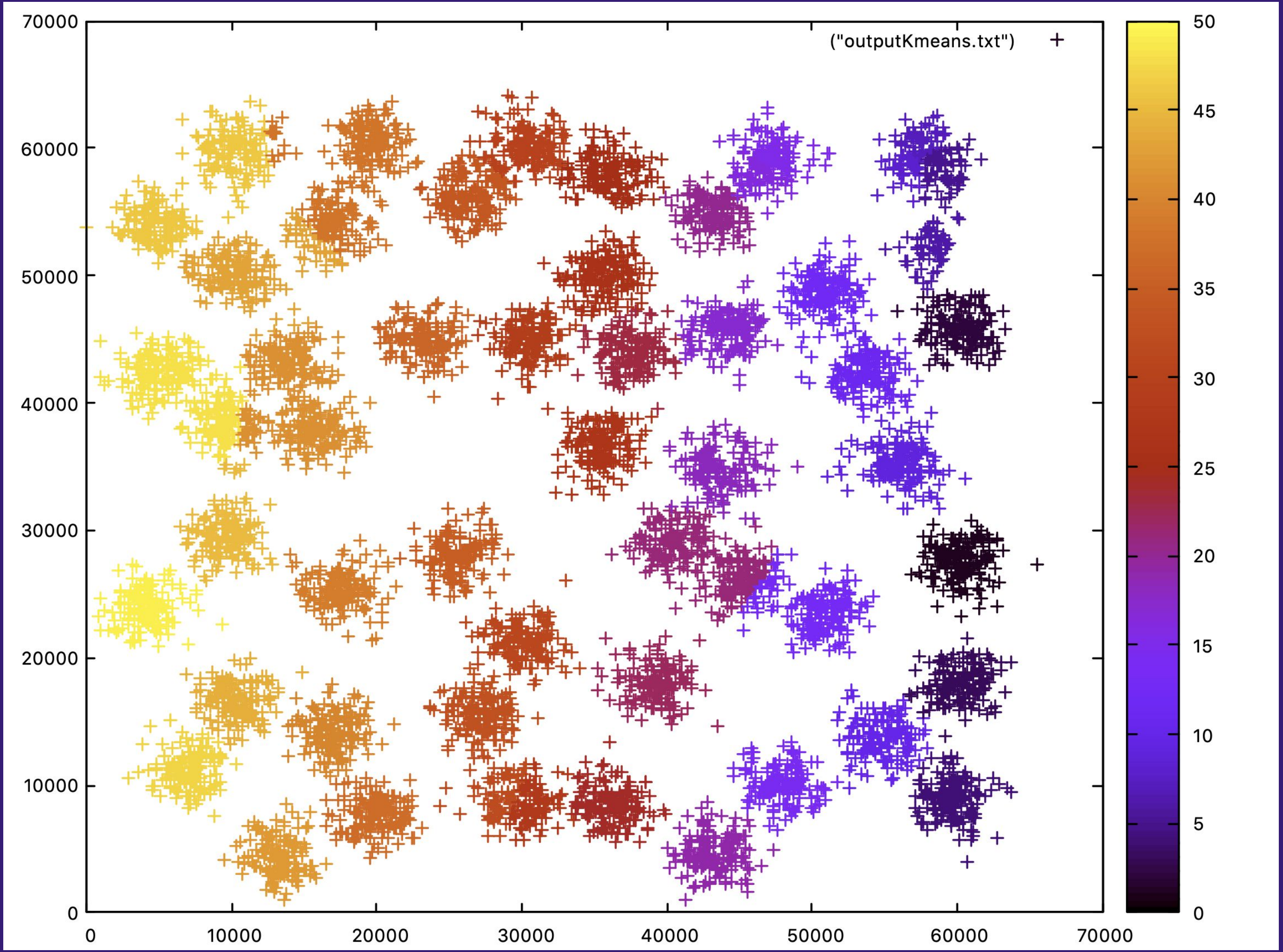
Iter : 3


("outputKmeans.txt")
70000
60000
50000
40000
30000
20000
10000
0
0
10000
20000
30000
40000
50000
60000
70000
9
8
7
6
5
4
3
2
1

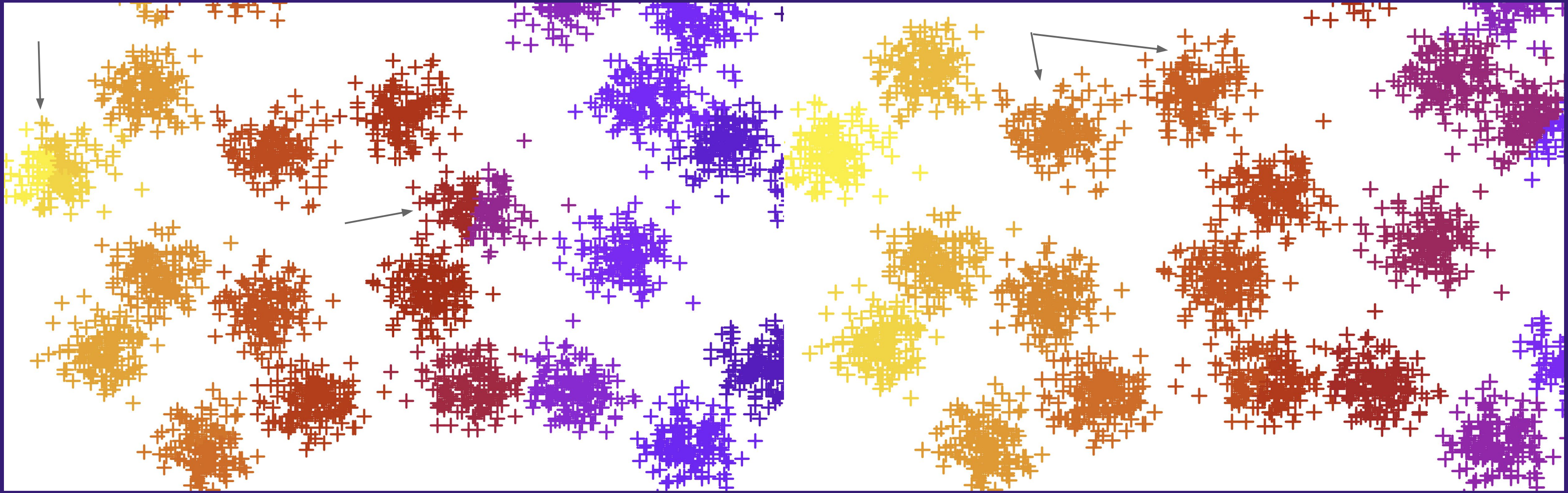
Iter : 1


("outputKmeans.txt")
0
10000
20000
30000
40000
50000
60000
70000
0
10000
20000
30000
40000
50000
60000
70000
0
5
10
15
20
25
30
35
40
45
50

Iter : 3


("outputKmeans.txt")
70000
60000
50000
40000
30000
20000
10000
0
0
10000
20000
30000
40000
50000
60000
70000
50
45
40
35
30
25
20
15
10
5
0

a closer look



Iter : 1

Iter : 3

ending thoughts

- surprised it works well
- folds do everything
- monads are great but not for everything
    - randomness
    - could be used in a future implementation
- fully make a generic type class for k-means
    - distance :: a -> a -> Double
    - distance :: a -> a -> b
- fix the missing points