732A75 Data Mining Lab-1

Anubhav Dikshit(anudi287) and Nahid Farazmand (nahfa911)

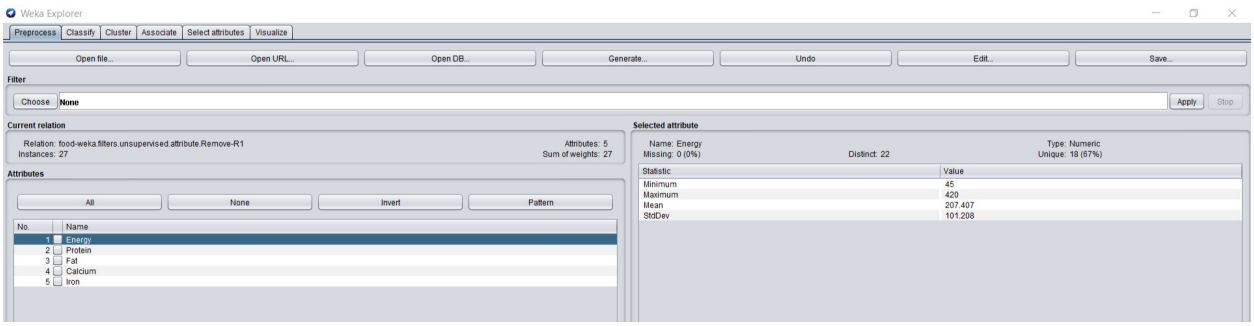
25 January 2019

SimpleKmeans:

Apply "SimpleKMeans" to your data. In Weka euclidian distance is implemented in SimpleKmeans. You can set the number of clusters and seed of a random algorithm for generating initial cluster centers. Experiment with the algorithm as follows:

1. Choose a set of attributes for clustering and give a motivation. (Hint: always ignore attribute "name". Why does the name attribute need to be ignored?)

Typically in clustering operations we only use numeric attributes, any attributes of type category or having too many levels are excluded. The reasoning is due to the fact clustering tends to be based on some distance measurment, and distance of attributes such as name, rownumbers are meaningless and may only come in the way of optimal solution.



2. Experiment with at least two different numbers of clusters, e.g. 2 and 5, but with the same seed value 10.

Weka Explorer

Preprocess | Classify | Cluster | Associate | Select attributes | Visualize

Clusterer

Choose **SimpleKMeans** -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1 -S 10

Cluster mode

- Use training set
- Supplied test set — Set...
- Percentage split — % 66
- Classes to clusters evaluation
- (Num) Iron
- Store clusters for visualization

Ignore attributes

Start | Stop

Result list (right-click for options)

09:31:42 - SimpleKMeans

Clusterer output

```
             Fat
             Calcium
             Iron
Test mode:    evaluate on training data


=== Clustering model (full training set) ===


kMeans
======

Number of iterations: 2
Within cluster sum of squared errors: 5.069321339929419

Initial starting points (random):

Cluster 0: 340,20,28,9,2.6
Cluster 1: 170,25,7,12,1.5

Missing values globally replaced with mean/mode

Final cluster centroids:
                        Cluster#
Attribute    Full Data          0          1
               (27.0)      (9.0)     (18.0)
=============================================
Energy        207.4074   331.1111   145.5556
Protein             19         19         19
Fat            13.4815    27.5556     6.4444
Calcium         43.963     8.7778    61.5556
Iron            2.3815     2.4667     2.3389



Time taken to build model (full training data) : 0.01 seconds

=== Model and evaluation on training set ===

Clustered Instances

0       9 ( 33%)
1      18 ( 67%)
```

Weka Explorer

Preprocess | Classify | Cluster | Associate | Select attributes | Visualize

Clusterer

Choose **SimpleKMeans** -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 5 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1 -S 10

Cluster mode

- Use training set
- Supplied test set — Set...
- Percentage split — % 66
- Classes to clusters evaluation
- (Num) Iron
- Store clusters for visualization

Ignore attributes

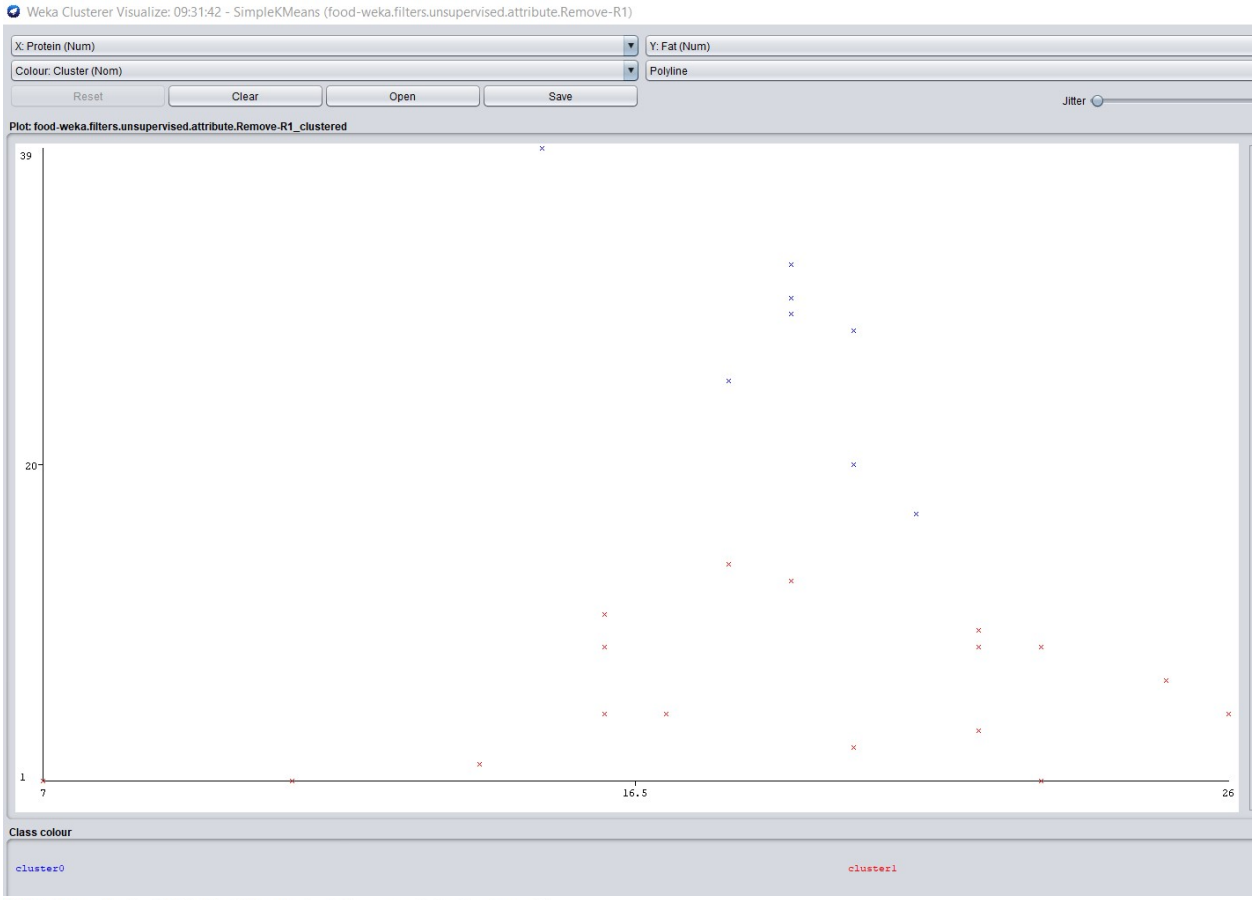
Start | Stop

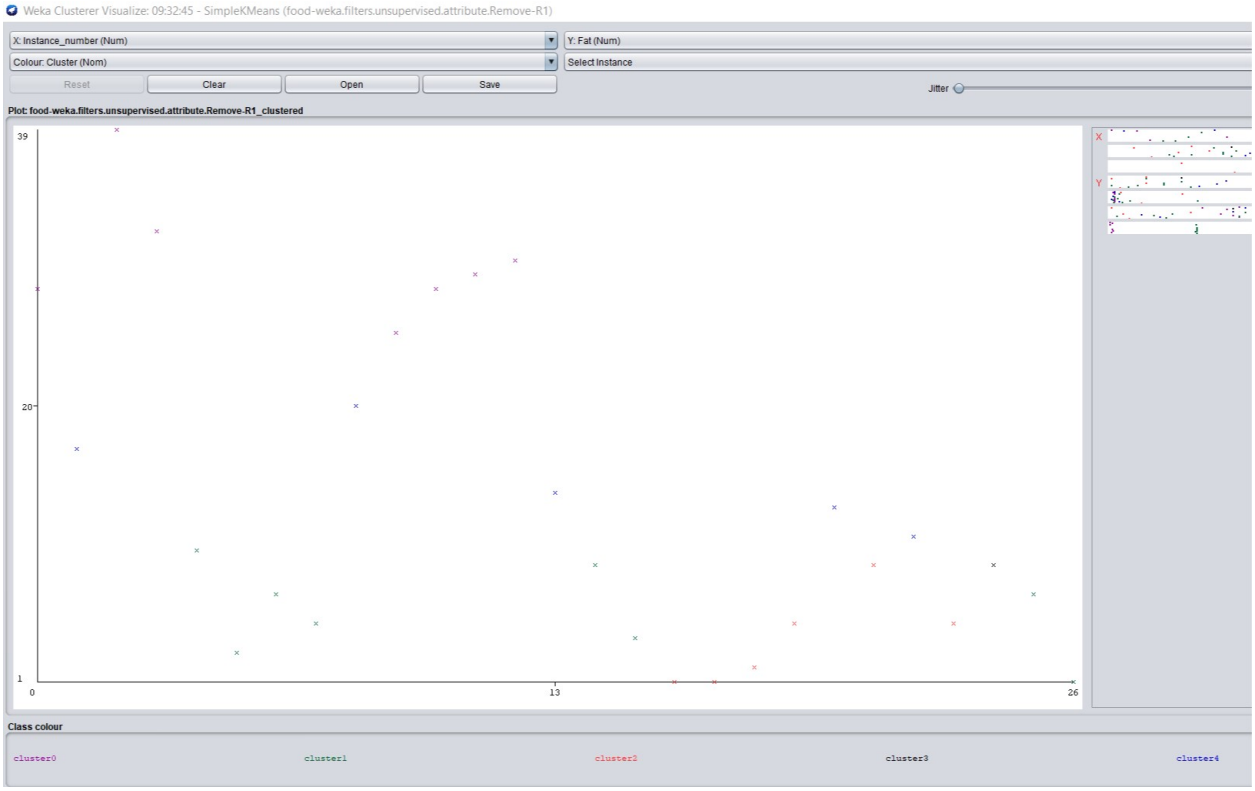
Result list (right-click for options)

09:31:42 - SimpleKMeans
09:32:45 - SimpleKMeans

Clusterer output

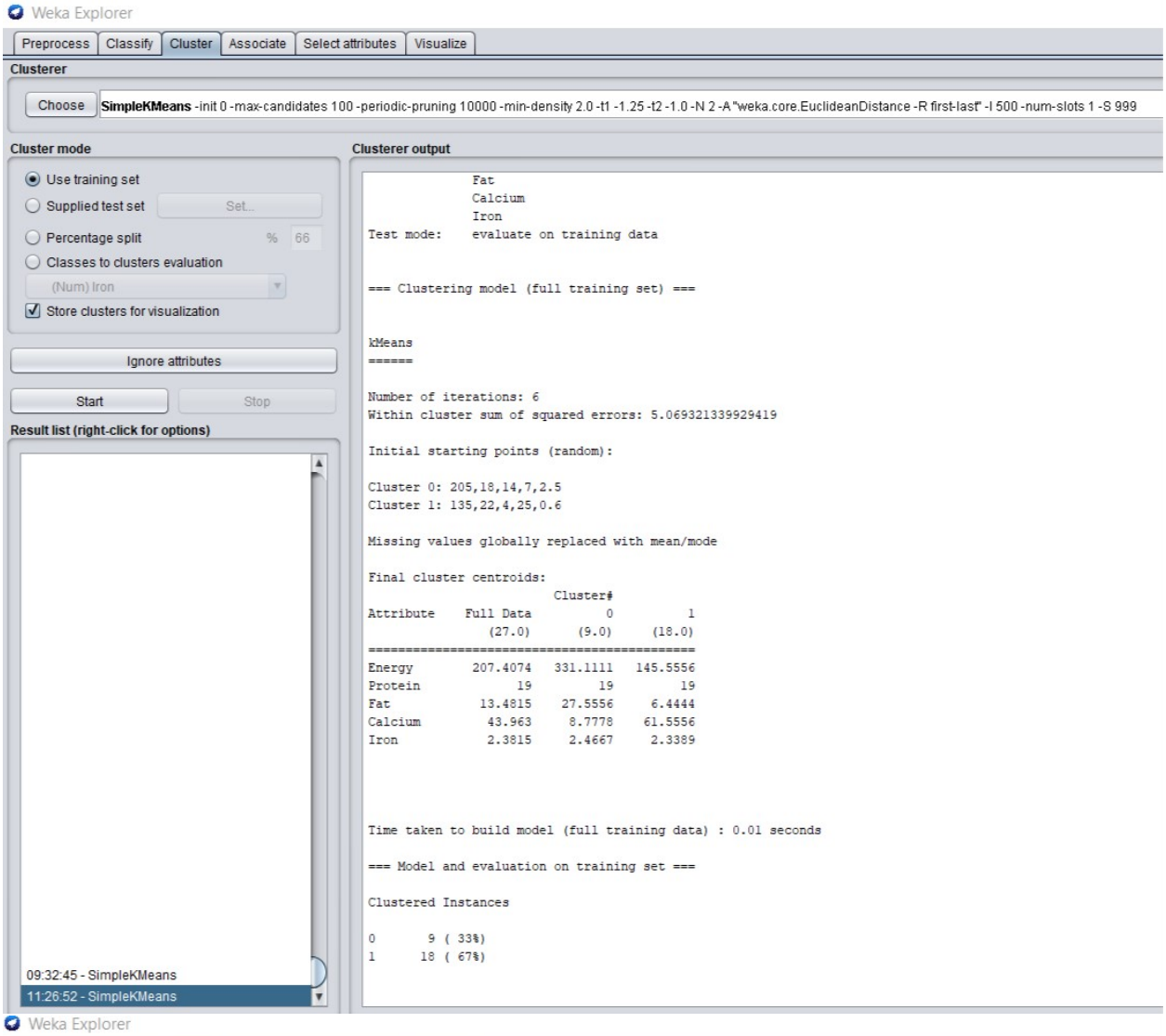
```
=== Clustering model (full training set) ===


kMeans
======

Number of iterations: 4
Within cluster sum of squared errors: 2.750432407251998

Initial starting points (random):

Cluster 0: 340,20,28,9,2.6
Cluster 1: 170,25,7,12,1.5
Cluster 2: 90,14,2,38,0.8
Cluster 3: 180,22,9,367,2.5
Cluster 4: 300,18,25,9,2.3

Missing values globally replaced with mean/mode

Final cluster centroids:
                        Cluster#
Attribute    Full Data          0          1          2          3          4
               (27.0)      (7.0)      (8.0)      (6.0)      (1.0)      (5.0)
=============================================================================
Energy        207.4074   352.8571    153.125      102.5        180        222
Protein             19    18.5714      23.25       13.5         22       18.8
Fat            13.4815    30.1429       5.75     3.8333          9         15
```

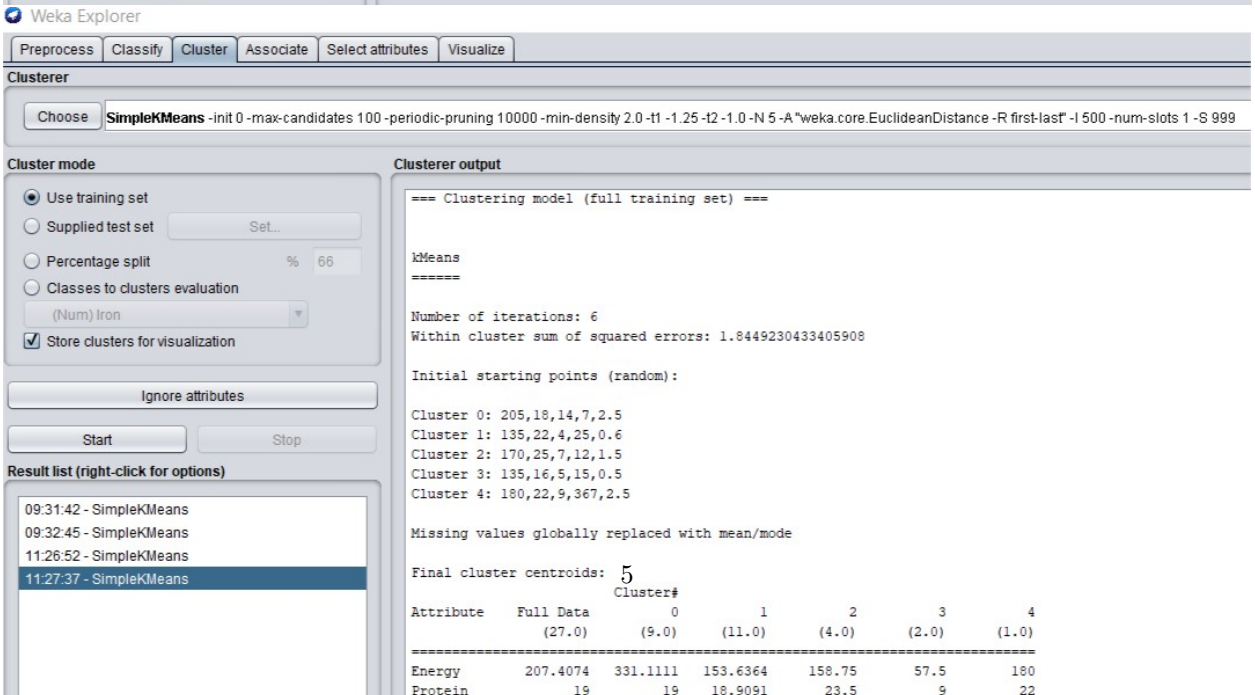

Weka Clusterer Visualize: 09:31:42 - SimpleKMeans (food-weka.filters.unsupervised.attribute.Remove-R1)
X: Protein (Num)
Y: Fat (Num)
Colour: Cluster (Nom)
Polyline
Reset
Clear
Open
Save
Jitter
Plot: food-weka.filters.unsupervised.attribute.Remove-R1_clustered
39
20
1
7
16.5
26
Class colour
cluster0
cluster1

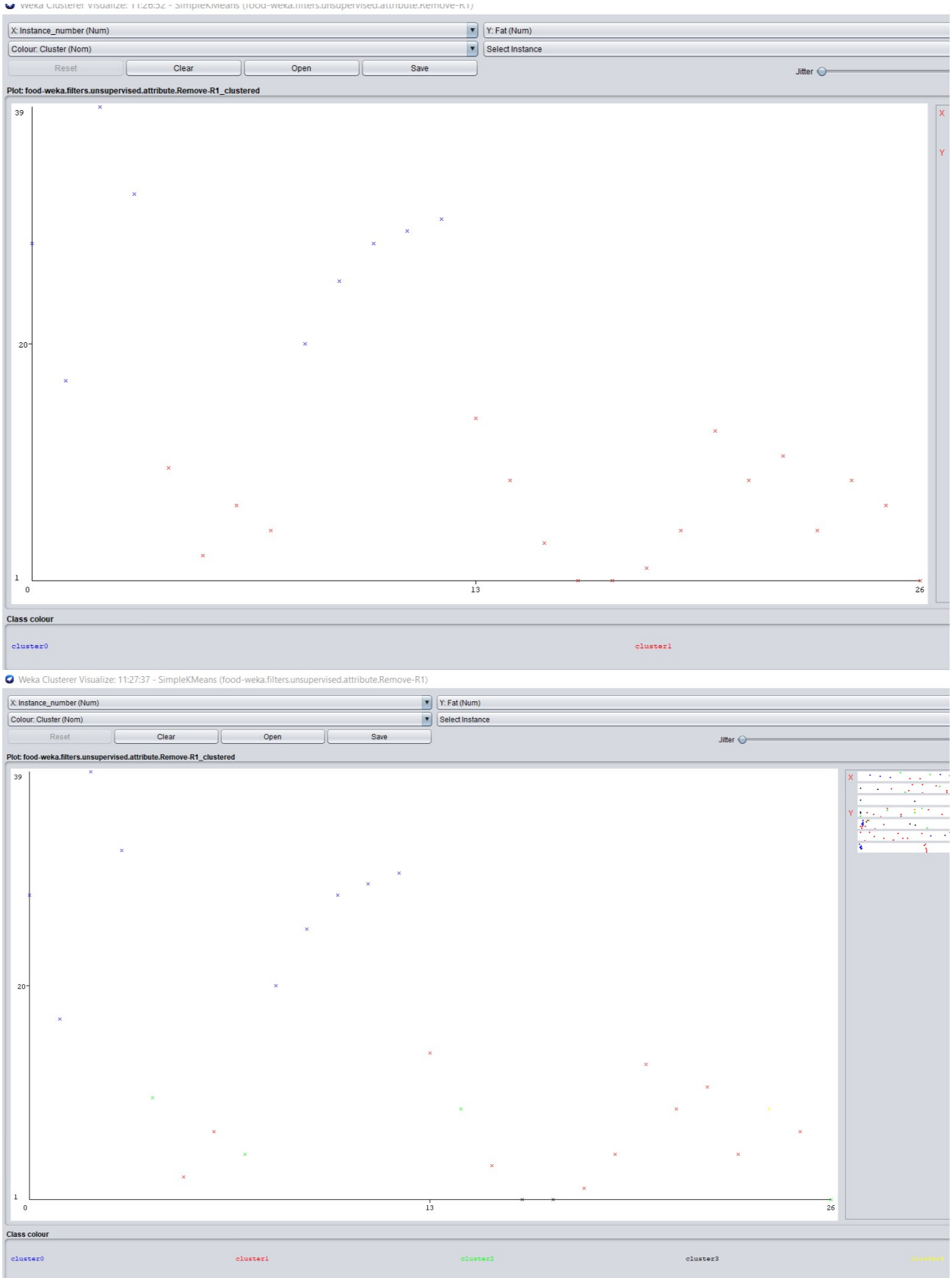

Weka Clusterer Visualize: 09:32:45 - SimpleKMeans (food-weka.filters.unsupervised.attribute.Remove-R1)
X: Instance_number (Num)
Y: Fat (Num)
Colour: Cluster (Nom)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot: food-weka.filters.unsupervised.attribute.Remove-R1_clustered
39
20
1
0
13
26
Class colour
cluster0
cluster1
cluster2
cluster3
cluster4

The more clusters we have, the more subdivisions we get and due to the nature of k-means (parition based algorithm) a cluster will never be empty, there will be no overlap and finally all data points will be covered. This leads to the possibility that there are some clusters which have so little data points (k=5) that they are not meaningful. Thus care must be taken to ensure that we specify k carefully and visually inspect if they make sense.

3. Then try with a different seed value, i.e. different initial cluster centers. Compare the results with the previous results. Explain what the seed value controls.







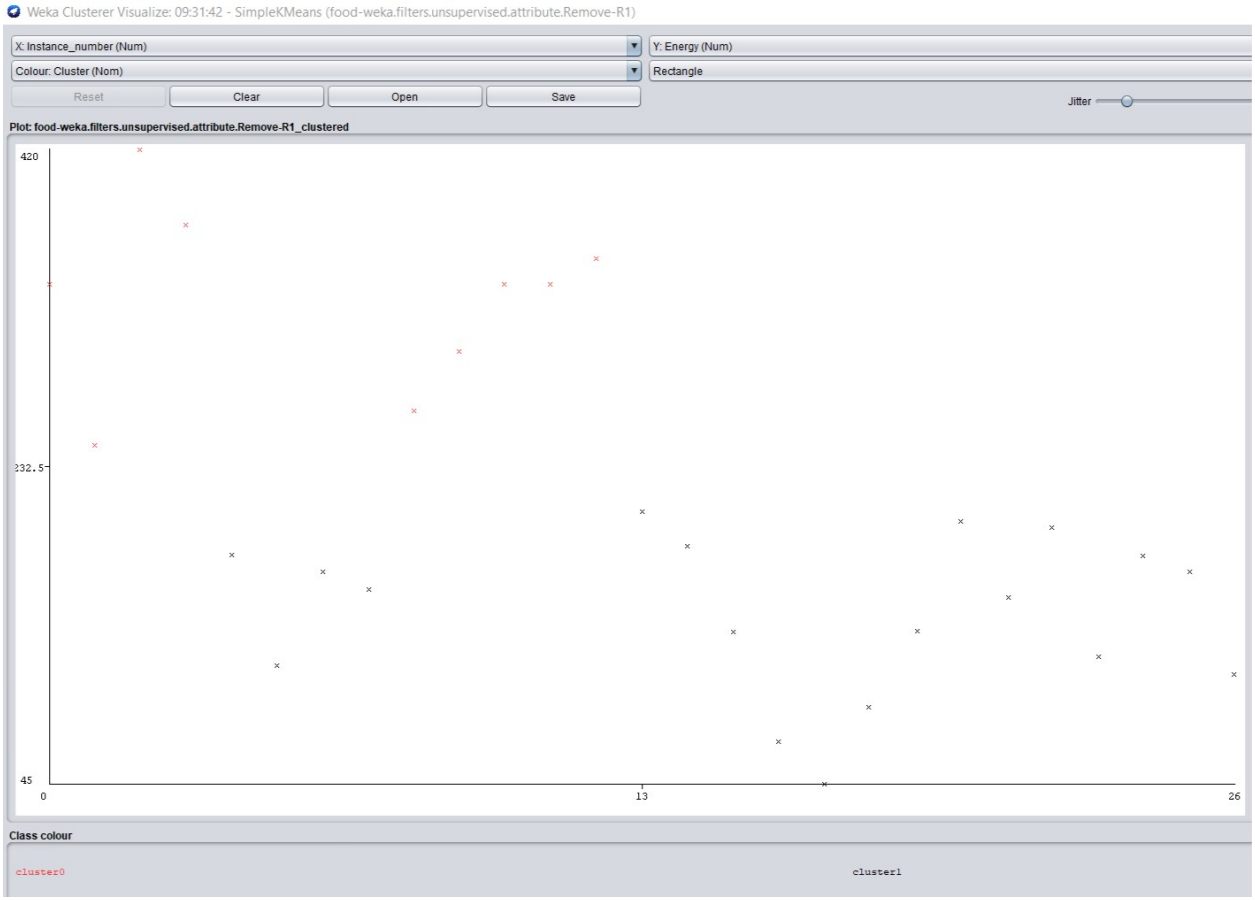
The seed value controls the starting points of the cluster centroids. If we change the value of the seed, the

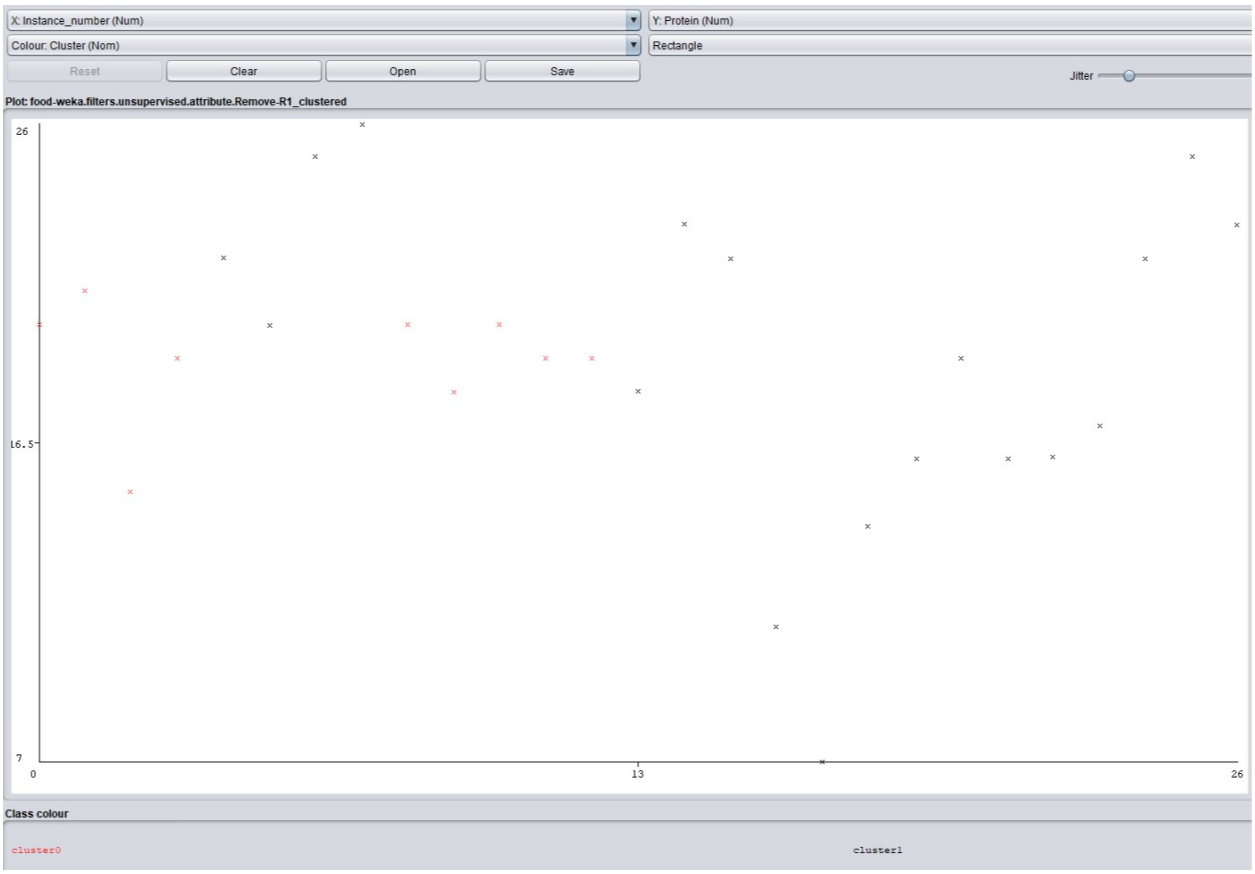
algorithm seems much more unstable for larger values of k, producing quite different clusters, which is likely also due to the fact that we have such a small dataset. However, if k is small, the clusters produced are quite similar for different seeds. With $k = 2$ for instance, the results are quite similar, evaluating by the centroids of the clusters, however for $k = 5$, much less so.

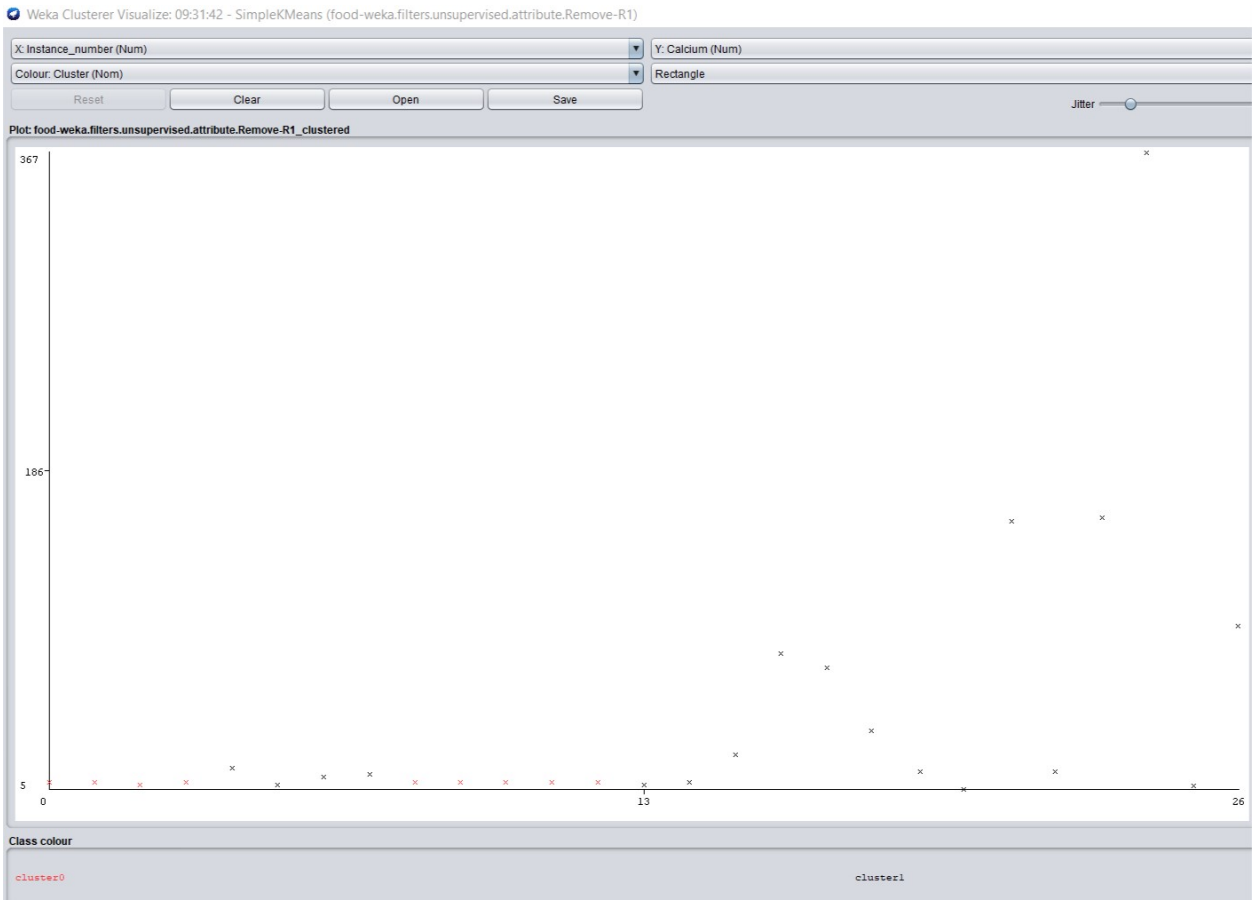
We chose these values in order to compare how meaningful the clusters when there are many vs when there are few. The number of data points in the clusters may vary too with the seed. This, along with the fact that the algorithms converges after many less iterations that the maximum we supply of 500, shows that k-means does not generate solutions corresponding to the global optimum, because otherwise starting by only changing the seed values would result in the same solutions. Instead, local optima are reached and then the algorithm stops. The choice of the seed values is somewhat arbitrary as they just influence a random starting point.

4. Do you think the clusters are "good" clusters? (Are all of its members "similar" to each other? Are members from different clusters dissimilar?)

We guage the goodness our clusters on what attributes we are looking at. Since there are many variables, it is hard to visualize them all at once, visualization of each variable against itself to see if similar elements get into similar clusters is a helpful way to examine different clusters.

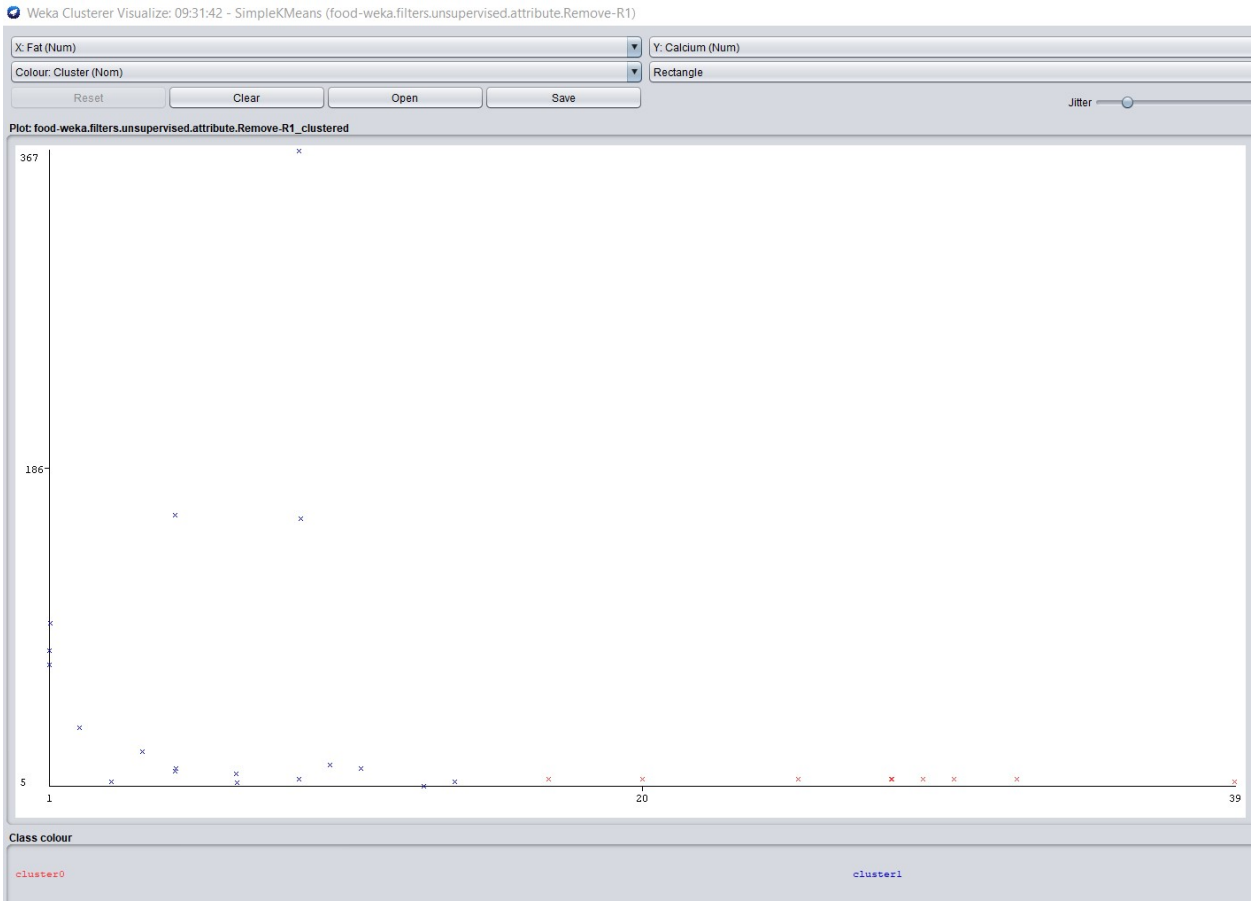



X: Instance_number (Num)
Y: Protein (Num)
Colour: Cluster (Nom)
Rectangle
Reset
Clear
Open
Save
Jitter
Plot: food-weka.filters.unsupervised.attribute.Remove-R1_clustered
26
16.5
7
0
13
26
Class colour
cluster0
cluster1



We see that clusters seperate some variables very well (Energy) but not so well on other variables(Calcium). In a very done clustering the elements in a cluster are closer to each other. When we view cluster vs. Energy we can see that elements are sort of tightly packed.

5. What does each cluster represent? Choose one of the results. Make up labels (words or phrases in English) which characterize each cluster.



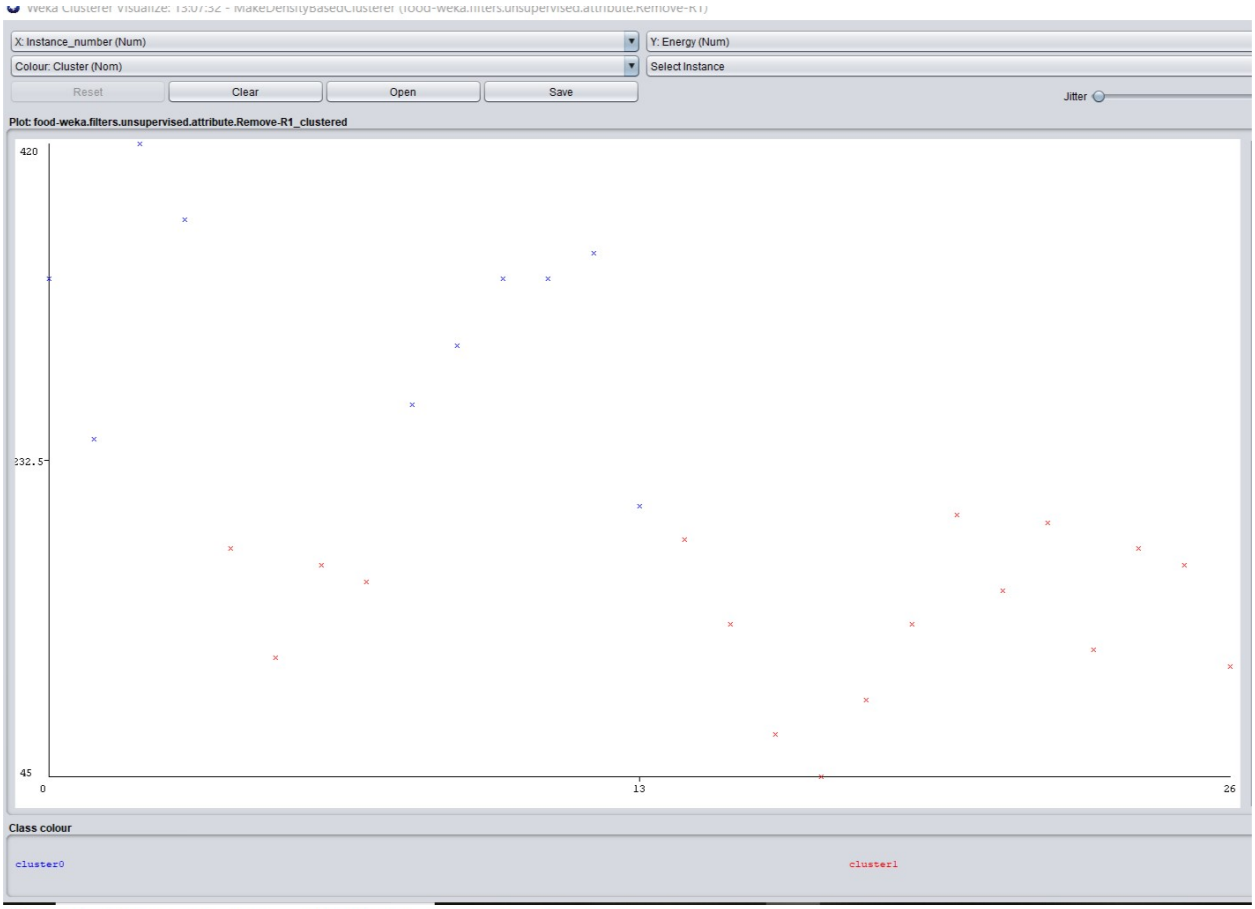
We chose $k = 2$ with seed 10. As can be seen in the plots above, the clearest separation is in terms of fat content. In particular, we can see that very fatty foods contain very little calcium. The blue cluster contains meats that are high in fat and contain very little calcium, while the red cluster contains meats that are low in fat.

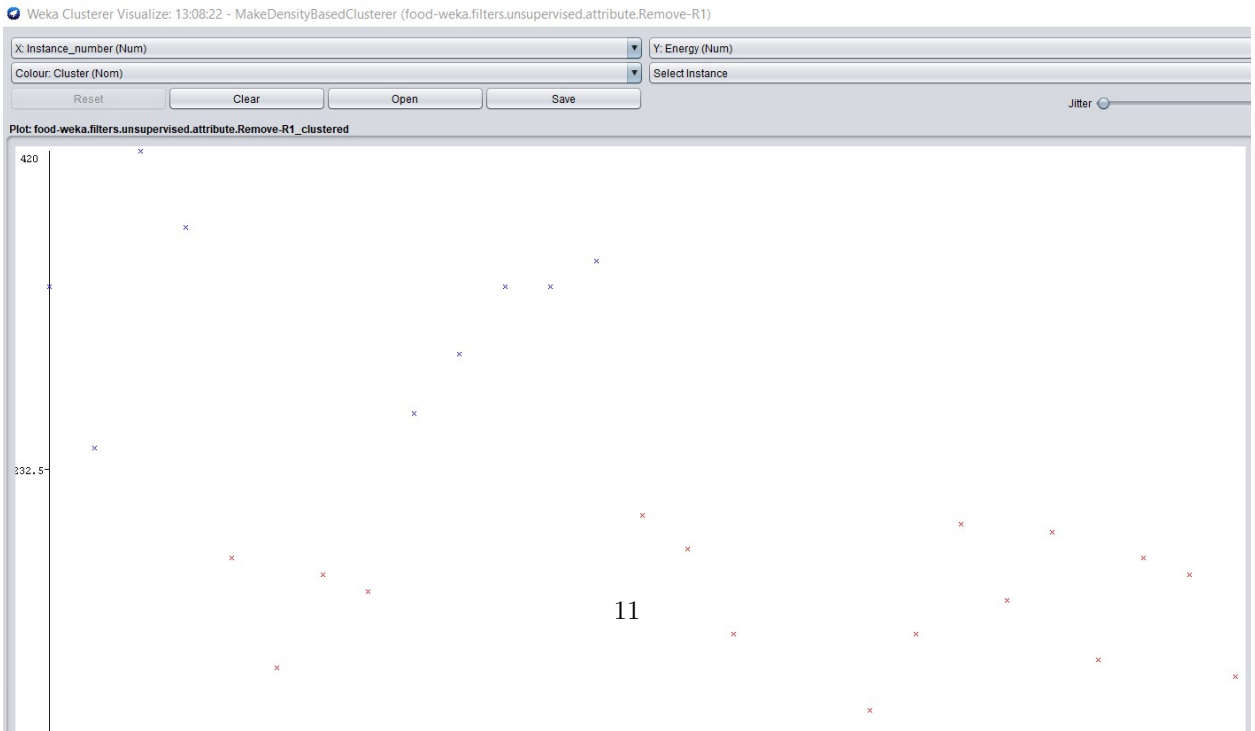
MakeDensityBasedClusters:

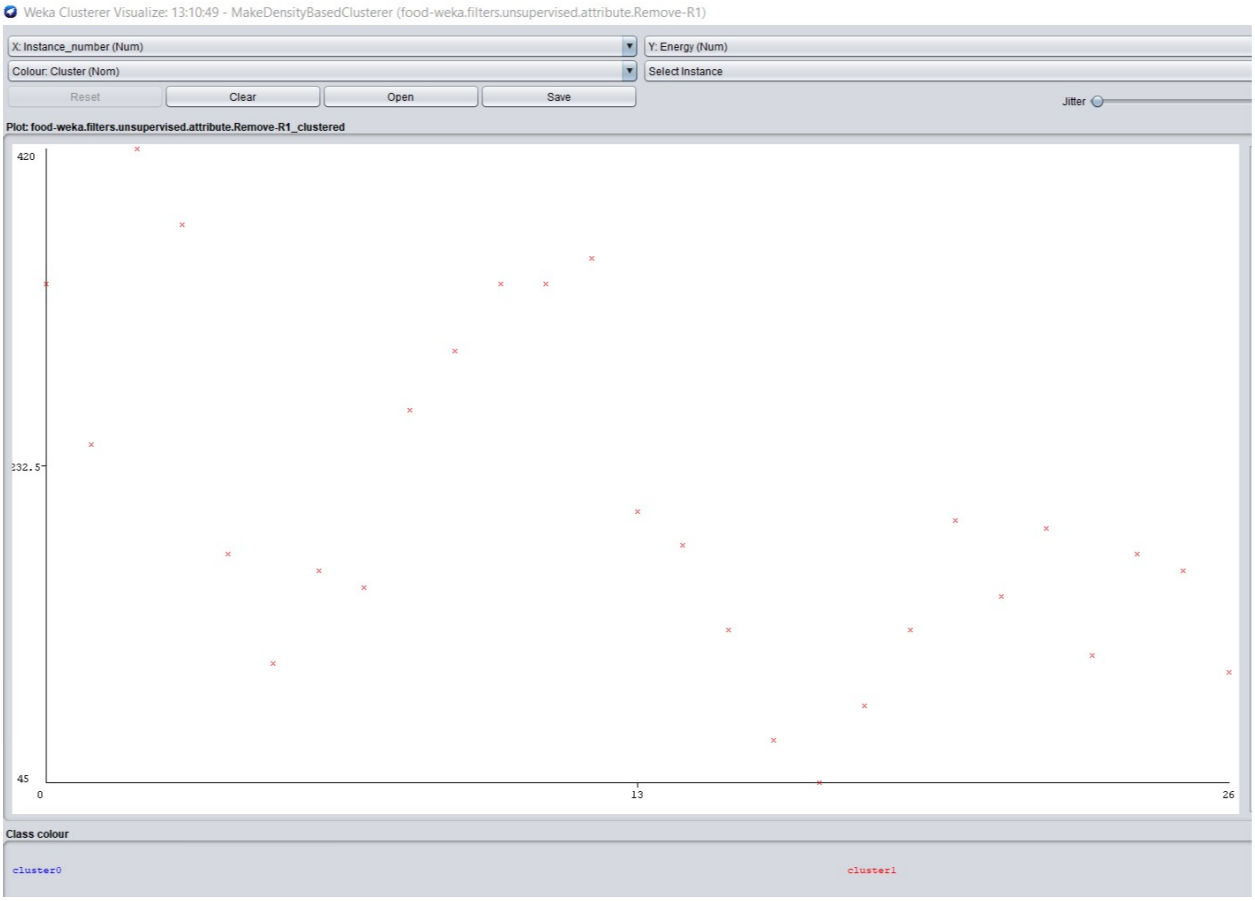
Now with MakeDensityBasedClusters, SimpleKMeans is turned into a denstiy-based clusterer. You can set the minimum standard deviation for normal density calculation. Experiment with the algorithm as the follows:

1. Use the SimpleKMeans clusterer which gave the result you haven chosen in 5).

2. Experiment with at least two different standard deviations. Compare the results. (Hint: Increasing the standard deviation to higher values will make the differences in different runs more obvious and thus it will be easier to conclude what the parameter does)







The standard deviation parameter influences the size of the final clusters. First, the algorithm produces clusters according to k-means, and then adjusts them with the minimum standard deviation. If we have a sufficiently small minimum standard deviation, we are guaranteed that we will have as many clusters as we originally asked for, in our case $k = 2$. However, if the standard deviation is too large we will end up including points that are dissimilar among each other, or even reducing the number of clusters, as more and more points need to be included to satisfy the minimum standard deviation requirement. The larger the standard deviation of an attribute for a given cluster in relation to the other attributes, the less that attribute serves to characterize the cluster. Taking cluster-means into account we need to consider the inter-cluster distance between the means for a given attribute in relation to the standard deviation of said attribute in order to evaluate its significance in characterizing a cluster.

With $k = 2$ we first used the default minimum of $\sigma = 1.0E - 6$, and we ended up with two clusters with the same data distribution as with the *k-means algorithm*. For clustering with $\sigma = 10$ we, as with $\sigma = 1.0E - 6$ got two clusters, even if two data points shifted cluster assignment. With $\sigma = 1000$ all elements end up being in one cluster as a result of no data points being outside this constraint on the minimum standard deviation.