L2 Experiments

There are 2 performance problems to solve in:

- KMeans.
- Memory management mechanism.

I did not change the KMeans implementation, however:

- It clearly has a performance problem.
- I did not check the topologies that it generates and this can be important for some optimization decisions, e.g., we can have 2/3 clusters with most of the features and 20+ with a very small number of features, etc.

I changed the memory management mechanism, in particular, I use the Euclidean distance (l2) instead of the pcl's KdTreeFLANN method to find the distances between features. L2 is not normalized.

I did the following experiments:

- Test the memory management mechanism using the L2 distance with no distances matrix.
- Test the memory management mechanism using the L2 distance with a distances matrix implemented using a dictionary, more precisely: *unordered_map<string, double> lookupTable*.
- Compare the results with the original implementation.

When using L2 the distances values magnitude changes when compared to the original implementation, see Table 1. As a consequence, it becomes hard to find appropriate values for the new implementation that map directly to the original ones.

KdTreeFLANN	**L2**
0	0
<u>0.000108092</u>	**<u>0.0103967</u>**
0.00819727	0.0905388
0.0153168	0.123761
0.0446929	0.211407
0.0495284	0.22255
0.0535045	0.23131
0.0618347	0.248666
…	…
0.133249	0.365033

Table 1 – Distances between a feature and other features obtained using 2 different distance methods: KdTreeFLANN and L2.

I used the following values for the new implementation, see Table 2.

	Original	**New**
Search Radius	0.05	0.55
Redundancy Threshold	0.000126	0.011

Table 2 – Search Radius and Redundancy Threshold values for the original and the new implementations, respectively.

I started by using the value 0.22 for the Search Radius but I realized that bigger values seem to stabilize KMeans and give better results, so I used 0.55 instead. The Redundancy Threshold follows from the results in Table 1, see underlined row. If such a mapping can be considered valid.

The results are shown in Table 3. The tests are using the some categories and views and the views are seen in the same order.

In Table 3, Redundancy Filter refers to the search domain used to check redundancy:

- All - means all features in memory
- Cluster - means only the features in the same cluster to which the feature being checked belongs to.

	Original	**Test 1**	**Test 2**	**Test 3**
Nº Categories	5	5	5	5
Nº Seen Objects	100	100	100	100
Redundancy Threshold	0.000126	0.011	0.011	0.011
Search Radius	0.05	0.55	0.55	0.55
Distance Method	Pcl	L2	L2	L2
Redundancy Filter	All	All	Cluster	Cluster
Distance Matrix	No	No	No	Yes
Distance Matrix Size	0	0	0	25936164
Nº Clusters	37	39	38	38
Accuracy	82.50%	92.50%	92.50%	92.50%
Features in Memory	M: 20271 R: 3593 D: 0 F: 3593	M: 20101 R: 3763 D: 0 F: 3763	M: 20151 R: 3713 D: 0 F: 3713	M: 20151 R: 3713 D: 0 F: 3713
Time	8:46	11:21	7:2	9:41
Test Filename	t_41	t_39	t_37	t_36

Table 3 - Tests' results.

Test	**Description**
Original	Test using the unchanged original code.
Test 1	Test using the L2 distance. Redundancy is checked against all features in memory and no distances matrix is used.
Test 2	Test using the L2 distance. Redundancy is checked against the features that belong to the cluster to which the feature being checked belongs to and no distances matrix is used.
Test 3	Test using the L2 distance. Redundancy is checked against the features that belong to the cluster to which the feature being checked belongs to and a distances matrix is used.

Symbol	**Description**
M	Final number of features in memory.
R	Number of redundant features found.
D	Number of features forgotten by applying the memory decay threshold.
F	Total number of forgotten features.

Conclusions

Tests 1, 2 and 3 give better accuracy results than the original test and they are finding more redundant features. This could be related to the new search radius value but I did not look into it. This was a surprising result.

The number of clusters is very similar. I stress that the search radius value seems to stabilize the KMeans.

When checking redundancy it seems that there isn't a big difference between checking redundancy against all features in memory and checking redundancy against the features in the cluster to which the feature being checked for belongs. BUT THIS MAY BE DEPENDENT ON THE CONFIGURATION, e.g., the search radius value, etc. That is, it can be dependent on the KMeans 'stability', 'on its topology being well-formed'. Checking redundancy at the cluster level seemed an interesting optimization and it yields the best results. However, a poor configuration, 'an ill KMeans topology', will find less redundant features and the accuracy results may decay.

Finally, the use of a distances matrix seems to have a major problem. The number of entries is huge. In Test 3, with only 20151 features in memory the matrix has 25936164 entries and these are not all the distances because I'm checking redundancy at the cluster level. This implies, intuitively, that there will be an explosion on the number of entries as the number of features grows. I did not test the use of the distances matrix with more features but I'm considering doing a test on my computer at IEETA with more categories and settings to induce memory decay.

I send the source code and the files containing the tests' output.

Directory	**Description**
SI_CODE_L2	Source code using the L2 distance without a distances matrix.
SI_CODE_L2+DistanceLookupTable	Source code using the L2 distance with a distances matrix.

File	**Description**
t_36	L2 + distances matrix output test.
t_37	L2 with no distances matrix and checking redundancy against the features in the cluster to which the feature being checked for belongs output test.
t_39	L2 with no distances matrix and checking redundancy against all features in memory output test.
t_41	Original test output.