Prototype selection for nearest neighbor

Lenord Melvix
A53092538

A short, high-level description of your idea for prototype selection.

For every label in the training data, the data points are sorted in ascending order of the euclidean distance from the mean value of training data for that label. From this sorted list of points, K unique data points are chosen (where K is computed by equation given below) such that these K data points are equally spaced along the sorted list - let this distance by θ. For each data point in K, the mean value of 2*θ data points in the list with the chosen data point as the middle value is computed and added to the prototype set. For explanatory purpose - this algorithm is term as **Hop Cluster** algorithm

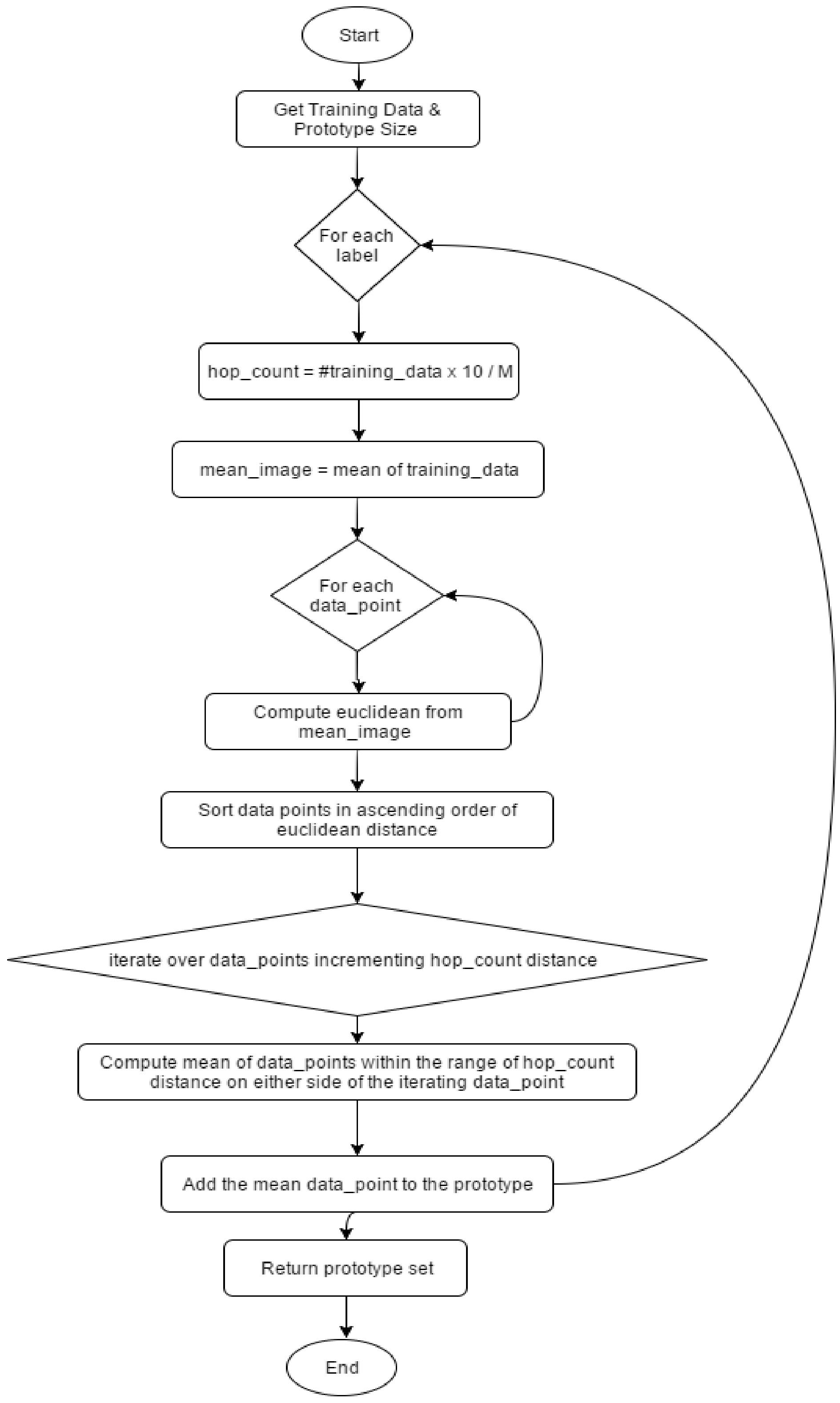
$$K = size\ of\ prototype / number\ of\ labels$$

Concise and unambiguous pseudocode

Pseudocode :

```
prototype_selection (INPUT : S - training_dataset, M - prototype_size) {
       initialize P - prototype_dataset
       for each s_i in S {          //where s_i is training dataset of label i
              hop_count  = (sizeof(s_i) * 10)/M
              mean_image = mean(s_i)
              Add mean_image to P
              sort s_i in terms of euclidean distance from mean_image
              for index in range(0, sizeof(s_i), hop_count):
                     mean_image = mean(s_i[index-hop_count]::s_i[index+hop_count])
                     mean_label = s_i[0].label
                     Add mean_image to P
              end
       end
       return P
end
```

Flow Diagram of Hop Cluster :

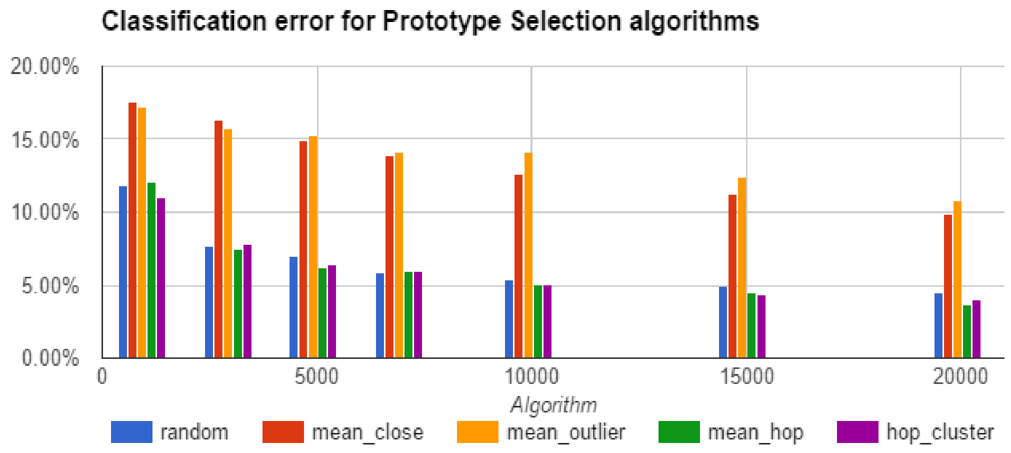

Start
Get Training Data & Prototype Size
For each label
hop_count = #training_data x 10 / M
mean_image = mean of training_data
For each data_point
Compute euclidean from mean_image
Sort data points in ascending order of euclidean distance
iterate over data_points incrementing hop_count distance
Compute mean of data_points within the range of hop_count distance on either side of the iterating data_point
Add the mean data_point to the prototype
Return prototype set
End

Experimental results:

For evaluation purposes, in addition to the Hop Cluster algorithm - I also implemented the following set of algorithms to evaluate the relative performance of the classifier. Each experiment was conducted 5 times and the results are averaged.

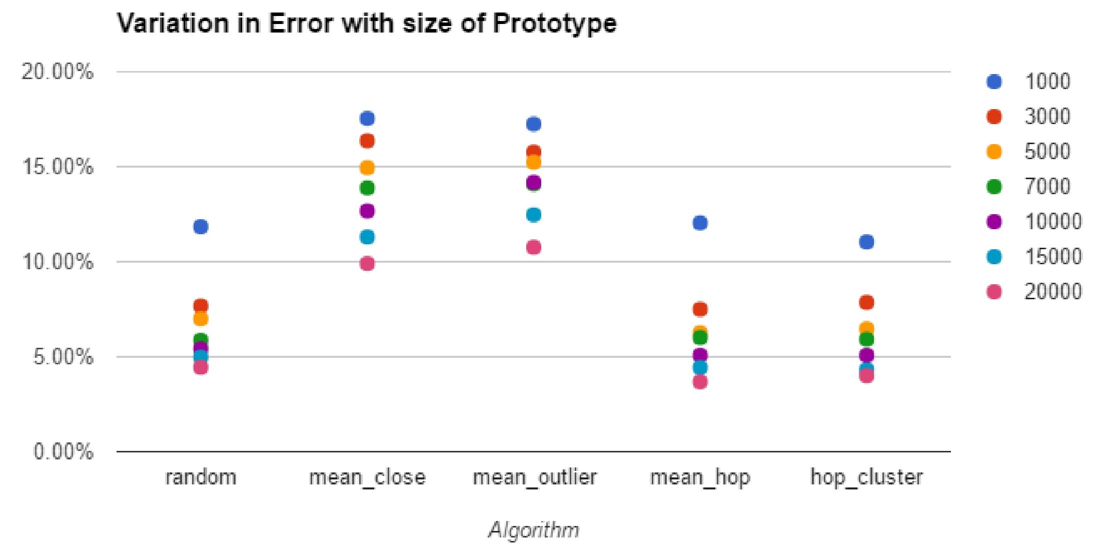
Random : Pick M random data points from training set
Mean + Close : Pick mean data point and M/10 points closest to mean for each label
Mean + Outlier : Pick mean data point and M/10 points farthest from mean for each label
Mean + Hop : Like Hop cluster, but pick only the data point at hop distance not mean of cluster

	Prototype Size						
Algorithm	**1000**	**3000**	**5000**	**7000**	**10000**	**15000**	**20000**
random	11.84%	7.67%	7.00%	5.87%	5.42%	4.98%	4.45%
mean_close	17.53%	16.36%	14.94%	13.88%	12.67%	11.30%	9.90%
mean_outlier	17.24%	15.76%	15.23%	14.09%	14.17%	12.47%	10.76%
mean_hop	12.04%	7.50%	6.26%	6.01%	5.08%	4.44%	3.69%
hop_cluster	11.04%	7.86%	6.47%	5.92%	5.08%	4.32%	4.01%



Critical evaluation:

Performance for small prototype sizes: Experimental results show that Hop cluster out performs random prototype selection significantly for small prototype sizes in range of ~1000. This can be attributed to reason that random selection would have picked training data disproportionately thereby resulting in poor performance for some of the labels. While hop-cluster performs 1 percent point better than random due to uniform distribution of training data points in the prototype set - selected proportionately across all the labels. However, the overall performance of both algorithms are poor due to insufficient amount of training data to build a reliable model. It can also be noted that mean+close & mean+outlier algorithms perform worse than random algorithm because each label would have insufficient training data to encompass wide of same labelled data in the test set.



Performance for sufficiently large prototype sizes: Experimental results show that performance of all the classification algorithms improve with the increase in prototype size owing to the fact that there is a significant increase in number of training data to build a reliable model that can classify the system with higher confidence. Hop cluster continues to perform marginally better than random algorithm due its clustering nature to pick mean data points across various clusters of data point in the same label thereby increasing the probability to correctly classify test data even more effectively for outlier cases. And this approach to encompass outliers is more effective than mean+outlier case as clustering of outlier data set take more number of outlier images into consideration while building the prototype while mean+outlier only picks most extreme cases leaving out the outliers that lie in the mid range.

Scope for improvement: There is scope for improving the hop cluster model by picking clusters based on the distribution of images in that label varying the cluster sizes based on the density of each cluster instead of uniformly picking points along the sorted dataset. This could improve the performance to a certain extent as the underlying distribution of images can be more accurately modelled in this fashion.

What next: I would take principal component analysis approach to classification to more effectively filter out images and resonate on the features that have more significance in making the classification decision. Other potential ways of improvement could be to explore EigenFaces and Fisherfaces approach to image recognition.