COMP 652 - Homework 5

Assigned: Nov 4, 2008
Due: Nov 12, 2008
Late: Nov 17, 2008

For the following two questions, consider the data in the file "HW5_Q1_X.txt". This file contains a large number (210,012) of length 3 vectors, each on one line. Each vector is actual the red, green, and blue intensity values of one of the pixels in the image shown at the right. For you information, the image has 516 rows and 407 columns. The pixels in the file are listed row by row from top to bottom, and within each row from left to right. For example, the first pixel in the file is the uppermost left pixel in the image. The second line of the file contains the pixel to the right of that one, and so on. In this assignment, we will explore clustering methods, applying them in particular to the problem of dividing the pixels of the image into a small number of similar clusters.

1. K-Means (10 pts.)

Consider the K-means clustering algorithm, as described in class. In particular, consider a version in which the inputs to the algorithm are:

1. The set of data to be clustered. (I.e., the vectors $\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \ldots$)
2. The desired number of clusters, K.
3. Initial centroids for the K clusters.

Then the algorithm proceeds by alternating: (1) assigning each instance to the class with the nearest centroid, and (2) recomputing the centroids of each class—until the assignments and centroids stop changing.

There are many implementations of K-means publicly available, as well as available in software packages such a MATLAB. However, please implement K-means clustering yourself, from scratch! Then, use your implementation to cluster the data in the file mentioned above, using $K = 8$, and the initial centroids:

R	G	B
255	255	255
255	0	0
128	0	0
0	255	0
0	128	0
0	0	255
0	0	128
0	0	0

Report in all of the following:

- How many clusters there are in the end. (Recall that a cluster can "disappear" in one iteration of the algorithm if no vectors or closest to its centroid.)
- The final centroids of each cluster.
- The number of pixels associated to each cluster.
- The sum of squared distances from each pixel to the nearest centroid after every iteration of the algorithm.
- And turn in your code.

Of course, it's fun to visualize your result by replacing each pixel with the centroid to which it is closest, and displaying the resulting image. This is readily done in MATLAB. But as it may not be easy in whatever programming language you are using, I leave this as an optional exercise.

2. Hierarchical clustering (10 pts.)

In the lecture discussion of hierarchical clustering, we mostly focussed on agglomerative methods, which build a tree from the bottom up. In this question, however, we will look a divisive clustering, which builds a tree from the top down.

Professor X proposes the following recursive algorithm for constructing a hierarchical clustering. Start with all the data at the root node of the tree. Find the two data points that are farthest from each other according to Euclidean distance. Call these points i and j. Then divide the data in two, by separating all the points that are closer to i from all the points that are closer to j. These two subsets of the data define the left and right children of the root node. Proceed recursively.

Professor Y has a similar idea. But instead of finding the two points, i and j, that are farthest apart when making a split, he proposes to: (1) compute the centroid of the data. (2) Find the point i that is farthest from the centroid. (3) Find the point j that is farthest from point j.

A. (5 pts.) Is the hierarchical clustering produced by Professor X's algorithm monotone with respect to the complete-linkage cluster dissimilarity metric? That is, suppose the examples at a node are divided into sets A and B. And suppose that at the next level down in the tree, A is divided into C and D, while B is divided into E and F. Is $d(A, B) \geq d(C, D)$ and $d(A, B) \geq d(E, F)$, where d denotes the complete-linkage dissimilarity between sets? Prove your answer.

B. (5 pts.) Professor X's method (and all the standard agglomerative methods) are hard to apply to the image data, because they require computing the distances between all pairs of pixels. (The number of pairs of pixels is very large, even for modestly sized images.)

Professor Y's method, however, only takes linear time in the number of data points to compute each split. Implement Professor Y's method and apply it to the image data. You need not compute a full hierarchical clustering. Rather, compute a clustering only to depth three (so there are 8 leaf nodes). Report:

- The number of data points in each cluster and its centroid, for both the leaf clusters and the internal-node clusters.
- Compute the sum of squared distances from each pixel to the centroid of the leaf cluster into which it falls. How does this value compare to the result of the K-means clustering?
- Turn in your code.
- And if you so please and are able, you may want to recreate the image with each pixel replaced by its leaf cluster centroid.

Optional/Bonus: Is Professor Y's method monotone with respect to the complete-linkage dissimilarity metric?

3. Perceptrons and margin (10 pts.)

Consider the data in files "HW5_Q3_X.txt" and "HW5_Q3_Y.txt". The files describe a two-input binary classification problem, similar to data sets in the perceptron and SVM lectures. The data is linearly separable.

A. (5 pts.) Implement the perceptron learning rule (see slide 5 of lecture 12), except please initialize $\mathbf{w}$ and w_0 to zero. Use your implementation to find a $\mathbf{w}$ and w_0 that separates the data. Show the data along with the decision boundary, report the obtained values of $\mathbf{w}$ and w_0, and turn in your code. Also, compute the margin of your decision boundary:

$$\text{margin} = 2 \min_i y_i \left(\frac{\mathbf{w} \cdot \mathbf{x_i}}{\|\mathbf{w}\|} + \frac{w_0}{\|\mathbf{w}\|} \right)$$

B. (5 pts.) The perceptron rule is guaranteed to find a separating hyperplane if one exists. But that plane may not have maximum margin. Consider the following scheme, which is similar to the perceptron rule, but has a different criterion for choosing on which examples to update.

- Initialize $\mathbf{w}$ and w_0 to zero.
- Repeat the following:
  - Find the data point i that minimizes $y_i \left(\frac{\mathbf{w} \cdot \mathbf{x_i}}{\|\mathbf{w}\|} + \frac{w_0}{\|\mathbf{w}\|} \right)$
    (In the first round, $\|\mathbf{w}\| = 0$, and you can choose an arbitrary i for your update.)

– Update $\mathbf{w}$ and w_0 just as in the perceptron learning rule:

$$\begin{aligned} w_0 &\leftarrow w_0 + \alpha y_i \\ \mathbf{w} &\leftarrow \mathbf{w} + \alpha y_i \mathbf{x_i} \end{aligned}$$

where α is a step-size parameter.

Implement the approach described above, and apply it to the data set. It is up to you to choose an appropriate step size and number of iterations or stopping criterion. Show the data along with the decision boundary obtained, report the values of $\mathbf{w}$ and w_0 obtained, and turn in your code. Also, report whether the decision boundary correctly sperates the data, and if so, what margin was obtained.