EECS 484, Fall 2015 Problem Set 2

Clustering of patterns from image data

The provided file training_data.mat contains 231 column vectors derived from transformations of color images. The images are snapshots of 7 shirts, from which color pixels are randomly sampled and transformed into 10 features. These features are the first 10 rows of the training data.

The last row of the training data is the "attribute"—a code from 1 to 7, corresponding to the 7 shirts. You should use only the first 10 rows of each column vector as features for creating clusters, and use the 11'th element for associating a shirt code with each resulting cluster.

A second set of data is contained in "validation_data_anon.mat". This file also contains 231 column vectors of random samplings of snapshots from the same set of shirts. Each column contains 10 features, ordered and transformed in the same manner as the training data. The 11'th element (attribute= shirt code) has been stripped off (set to 0).

Perform clustering on the training data. Then categorize the validation data using your clusters. Your result should be a row vector containing 231 integers from 1 to 7, corresponding to your best estimate of the shirt code for each of the test patterns. Per the starter code, this vector will be saved as the file "classification_results."

Assignment:
Complete the missing code in "test_using_clusters_incomplete." Debug your code by checking:

- The original training data gets classified identical to the original clustering results
- The validation data gets clustered with a good success score

Run your resulting code on the anonymous data, and include your classification results with your solution upload to Blackboard.

Choose some variation(s) to explore. Comment on your results.

Submit:

- Any code that you altered (at a minimum, test_using_clusters_incomplete")
- Your file "classification_results"
- Your report, including:
    - Description of your clustering results from the training data
    - Description of your classification results from the validation data
    - Description of what variations you attempted and your observed results

Code organization:

Process_image:

The Matlab script “process_image.m” uses the function “sample_colors_from_image.m”. This routine does not need to be altered. It is merely a tool to assist visualizing the images in color space. It can be run from a directory that contains the original RGB images, and it displays both the original images and sampled representations in two color spaces: RGB and HSV.

Create_training_data*:* You do not have to alter this function, but you are welcome to experiment with variations of it. The purpose of this function is to create the file “training_data.mat,” which is used for performing clustering. Please note that your (anonymous) validation data was generated consistent with the settings in this routine. If you create variations on the training data, you should also create a corresponding set of consistent validation data.

Create_training_data uses the following functions: save_subsamples, sample_colors_from_image, compress_training_data.m, compress_cubes, and compress_matrix.

Save_subsamples loads a named picture, converts to HSV color representation, and samples this picture at 10,000 randomly chosen pixels (set by the line `nsamps = 10000)`. This value was set arbitrarily without much experimentation, and it is one of the parameters you could explore. This function uses “sample_colors_from_image” to compress the HSV colors into “ndivs” divisions each, where ndivs is the color-cube resolution chosen for color classification. This has been set to ndivs=8, which creates an 8x8x8 set of smaller color cubes. This resolution could be tested with alternative choices. The default results in 512 color cubes used. Classification of a color within these 512 choices is expressed as a vector of 512 components, where a single sample has only a single component non-zero. For 10,000 samples, the 512x1 vectors get populated with an equivalent histogram of color-cube distribution for each image.

The function “compress_training_data” takes the 512x1 and compresses this data further, from 512 features to only 10 features (selected arbitrarily by setting ncubes=10 in this function). This compression process chooses the top-10 features to retain based on strength of variance across different shirt images. Note that this is a crude approach (selecting individual features to retain). Principal Component Analysis (PCA) could be applied to make a more intelligent choice of linear combinations of features to retain, achieving compression that is more discriminating.

The results are retained in two matlab files “training_data”, which contains the matrix “all_patterns”, and “best_cube_indicies”, which contains the vector “best_cubes.” The former is the set of all compressed data, consisting of 231 compressed samples from 21 images. Each 11x1 column vector is a compressed color representation from a sampling of a single shirt. The first 10 rows are (equivalently) histogram vales in each of the chosen 10 color-cube features, and the 11’th row is the shirt code (1 through 7).

You are provided with a set of training data to use, which is consistent with the provided set of validation data. If you choose to experiment with alternative representation and compression, you will need to generate consistent validation data as well.

Create_validation_data: This routine additionally uses save_validation_subsamples and compress_cubes_given_indices. The important difference here is that the abbreviated set of features retatined (10 color cubes) must be consistent with the features retained within the training data. This code thus reads in the feature-selection vector determined by create_training_data, and it preserves these features in the same order, thus creating logically consistent training data.

The results of this process are contained in the file “validation_data”, which contains the array “all_patterns”. (Note that this overwrites in RAM the array “all_patterns” contained in the training data, but does not affect the values preserved in the file “training_data”). The validation data has the same format as the training data: 231 compressed color samples taken from the 7 shirts. Note that this data does contain the shirt codes in the 11th row.

If you alter “create_training_data”, you should make sure that edit “create_validation_data” consistently, and generate new validation data that is consistent with your training data.

Kmeans_clustering: This is the routine that clusters the training data, arriving at a clustering result that hopefully is useful in classifying the validation data. The routine kmeans_clustering uses the functions: find_minmax_feature_vals, scale_all_feature_values, do_reclustering, and find_closest_cluster. The clusters are seeded with random selections from the training vectors, so the clustering results will be different each time this is run. Kmeans_clustering does the following:

- loads the training-data file: load training_data; %loads "all_patterns"
- strips off the attribute codes--saves them as "attributes"
- finds min/max of each feature value
- rescales all features, creating: feature_scaled_vals
- does clustering, creating matrices:
  nclusters=40 (you may choose to explore a different number of clusters)
  pattern_assignments (each pattern assigned to one cluster)
  cluster_populations (each cluster has a number of members)
  cluster_attributes (assign an attribute=shirt code to each cluster)
  cluster_centroids (average feature vector representing cluster centroid for each cluster)
  cluster_rosters (membership array--each pattern should belong to 1 and only 1 cluster)

Consistency of clustering is viewed (in part) by checking for assignments of the training data itself to the learned clusters. This test does not guarantee that the clustering results have predictive power, but it can be a factor in your decision of which clustering results to retain.

All of the variables from clustering are saved in the file “clusters_from_training.” This file gets overwritten each time you run kmeans_clustering.

Test_using_clusters_incomplete.: This routine loads the files “clusters_from_training” and “validation_data” OR “validation_data_anon”. You should debug/tune this code using the validation data, then run in on the anonymous data to generate your classification results to be submitted.

This routine should classify the selected data using the clusters found from the training data, as follows:

- strip off attribute codes from validation vectors

- scale the raw data using existing min/max values (from training data)
- for reach validation pattern:
- find_closest_cluster
- get the attribute associated with this closest cluster; call this "`kmeans_inferred_attribute`"
- compare this to the correct classification (attribute), if available
- compile results--number of successful matches (for validation data—will be unknowable for the anonymous data)
- print out statistics (to test/debug your classification using the classification_data file)

Ideally, the validation data is (mostly) correctly classified. This routine will save results in the file "classification_results", which you should upload for your classification attempt with the anonymous patterns.