Image compression with *K-*means Clustering and Principal Component Analysis

Introduction

The *K-*means algorithm is a method to automatically cluster similar data examples together. *K-*means is an iterative procedure that starts by guessing the initial centroids, and then refines this guess by repeatedly assigning examples to their closest centroids and then recomputing the centroids based on the assignments.

Principal Component Analysis (PCA) performs dimensionality reduction by finding a direction (a vector) onto which to project the data so as to minimize the projection error.

Dataset:

Dataset is based on a cropped version of the labeled faces in the wild dataset.

1. *K-*means Clustering

Goal: implement the *K-*means algorithm and use it for image compression. Starting with an 2D dataset and use the *K-*means algorithm for image compression by reducing the number of colors that occur in an image to only those that are most common in that image.

* 1. Implementing *K-*means

For a training set , group the data into a few cohesive clusters.

The *K-*means algorithm is as follows:

% Initialize centroids

centroids = kMeansInitCentroids(X, K);

for iter = 1:iterations

% Cluster assignment step: Assign each data point to the closest centroid. idx(i) corresponds to cˆ(i),

%the index of the centroid assigned to example i

idx = findClosestCentroids(X, centroids);

% Move centroid step: Compute means based on centroid assignments

centroids = computeMeans(X, idx, K);

end

The inner-loop of the algorithm repeatedly carries out two steps:

(i) Assigning each training example *x(i)* to its closest centroid, and

(ii) Recomputing the mean of each centroid using the points assigned to it.

The *K-*means algorithm will always converge to some final set of means for the centroids. But the converged solution may not always be ideal and depends on the initial setting of the centroids. Therefore, *K-*means algorithm is run a few times with different random initializations. And to choose between these different solutions from different random initializations is to choose the one with the lowest cost function value (distortion).

1.1.1 Finding closest centroids

In the “cluster assignment” phase of the *K-*means algorithm, the algorithm assigns every training example *x(i)* to its closest centroid, given the current positions of centroids.

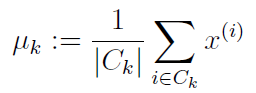
Specifically, for every example i :



where *c(i)* is the index of the centroid that is closest to *x(i)*, and  is the position (value) of the *j*th centroid.

1.1.2 Computing centroid means

Given assignments of every point to a centroid, the second phase of the algorithm recomputes, for each centroid, the mean of the points that were assigned to it. Specifically, for every centroid k:



where Ck is the set of examples that are assigned to centroid k.

Possible improvement: use a vectorized implementation that does not use loop so the code may run faster.

1.2 *K-*means on example dataset

Initial trial run the *K-*means algorithm on a toy 2D dataset

1.3 Random initialization

A good strategy for initializing the centroids is to select random examples from the training set.

Improvement:

% Initialize the centroids to be random examples

% Randomly reorder the indices of examples

randidx = randperm(size(X, 1));

% Take the first K examples as centroids

centroids = X(randidx(1:K), :);

Randomly permute the indices of the examples. Then, select the first K examples based on the random permutation of the indices.

1.4 Image compression with *K-*means

|  |  |
| --- | --- |
| Apply *K-*means to image compression.  A 24-bit color representation of an image, each pixel is represented as three 8-bit unsigned integers (ranging from 0 to 255) that specify the red, green and blue intensity values. For image contains thousands of colors, goal to reduce the number of colors.  Using the *K-*means algorithm to select K colors, K = 16, that will be used to represent the compressed image. | D:\SJCE\MTechCE\II Semester\Main Seminar\mlclass-ex7-005\mlclass-ex7\bird_small.png  Figure 1: 128x128 image [1]. |

Treat every pixel in the original image as a data example and use the *K-*means algorithm to find the 16 colors that best group (cluster) the pixels in the 3-dimensional RGB space. Use the cluster centroids in the image by replacing the pixels in the original image.

1.4.1 *K-*means on pixels

Load the image, and then reshapes it to create an *m* x 3 matrix of pixel colors (where *m* = 16384 = 128 X 128), and call *K-*means on it.

After finding the top *K* = 16 colors to represent the image, assign each pixel position to its closest centroid.

The original image requires 24 bits for each one of the 128 X 128 pixel locations, resulting in total size of 128 X 128 X 24 = 393,216 bits.

The new representation requires some overhead storage in form of a dictionary of 16 colors, each of which require 24 bits, but the image itself then only requires 4 bits per pixel location. The final number of bits used is therefore 16 X 24 + 128 X 128 X 4 = 65,920 bits,

which corresponds to compressing the original image by about a factor of 6.

1.5 Tests on more images, and vary *K* to see effects on the compression

2 Principal Component Analysis

Goal: Use principal component analysis (PCA) to perform dimensionality reduction. First by experiment with an example 2D dataset, and then use it on a bigger dataset of 5000 face image dataset, to find a low-dimensional representation of face images.

2.1 Example Dataset

Starting with a 2D dataset which has one direction of large variation and one of smaller variation and use PCA to reduce the data from 2D to 1D.

2.2 Implementing PCA

PCA consists of two computational steps:

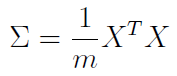
First, compute the covariance matrix of the data.

Then compute the eigenvectors U1; U2; . . . , Un; which will correspond to the principal components of variation in the data.

Before using PCA, normalize the data by subtracting the mean value of each feature from the dataset, and scaling each dimension so that they are in the same range.

After normalizing the data, run PCA to compute the principal components of the dataset.

Compute the covariance matrix of the data:



Where *X* is the data matrix with examples in rows, and *m* is the number of examples.

After computing the covariance matrix, Octave’s[2] SVD can be run on it to compute

the principal components.

2.3 Dimensionality Reduction with PCA

After computing the principal components, they can be used to reduce the feature dimension of dataset by projecting each example onto a lower dimensional space, . Use the eigenvectors returned by PCA and project the example dataset into a 1-dimensional space.

2.3.1 Projecting the data onto the principal components

For a dataset X, the principal components U, and the desired number of dimensions to reduce to K, project each example in X onto the top K components in U.

Top K components in U are given by the first K columns of U, that is U\_reduce = U ( : , 1 : K ).

2.3.2 Reconstructing an approximation of the data

After projecting the data onto the lower dimensional space, data can be approximately recovered by projecting them back onto the original high dimensional space, which is to project each example in Z back onto the original space and return the recovered approximation in X\_rec. The projection effectively only retains the information in the direction given by U1.

2.4 Face Image Dataset

|  |  |
| --- | --- |
| Run PCA on face images. The dataset[3] X is of face images, each 32 x 32 in grayscale. Each row of X corresponds to one face image (a row vector of length 1024).  2.4.1 PCA on Faces  To run PCA on the face dataset, normalize the dataset by subtracting the mean of each feature from the data matrix X, run PCA, results are the principal components of the dataset. Each principal component in U (each row) is a vector of length *n* (where for the face dataset, n = 1024). | D:\SJCE\MTechCE\II Semester\Main Seminar\Faces dataset.png  Figure 2: Faces dataset [3] |

2.4.2 Dimensionality Reduction

With computed principal components for the face dataset, it can be used to reduce the dimension of the face dataset. This allows using learning algorithm with a smaller input size (e.g., 100 dimensions) instead of the original 1024 dimensions, which can help speed up learning algorithm.

3 Applications

Reduction in the dataset size can help

(i) speed up learning algorithm significantly,

(ii) reduce memory needed to store data,

(iii) data visualization.

4 Parts to Implement

*K-*means

Find closest centroids (used in *K-*means)

Compute centroid means (used in *K-*means)

Initialization for *K-*means centroids

Principal Component Analysis

Perform principal component analysis

Projects a data set into a lower dimensional space

Recovers the original data from the projection

References:

Machine Learning by Andrew Ng: https://class.coursera.org/ml

[1] Photo : Frank Wouters

[2] Octave: http://www.gnu.org/software/octave/doc/interpreter/Matrix-Factorizations.html, http://octave.sourceforge.net/octave/function/svd.html

[3] Dataset is based on a cropped version (http://conradsanderson.id.au/lfwcrop/) of the labeled faces in the wild (http://vis-www.cs.umass.edu/lfw/) dataset.