Programming Exercise 7: K-means Clustering and Principal Component Analysis

Machine Learning

Introduction

In this exercise, you will implement the K-means clustering algorithm and apply it to compress an image. In the second part, you will use principal component analysis to find a low-dimensional representation of face images. Before starting on the programming exercise, we strongly recommend watching the video lectures and completing the review questions for the associated topics.

To get started with the exercise, you will need to download the starter code and unzip its contents to the directory where you wish to complete the exercise. If needed, use the `cd` command in Octave/MATLAB to change to this directory before starting this exercise.

You can also find instructions for installing Octave/MATLAB in the "Environment Setup Instructions" of the course website.

Files included in this exercise

`ex7.m` - Octave/MATLAB script for the first exercise on K-means
`ex7_pca.m` - Octave/MATLAB script for the second exercise on PCA
`ex7data1.mat` - Example Dataset for PCA
`ex7data2.mat` - Example Dataset for K-means
`ex7faces.mat` - Faces Dataset
`bird_small.png` - Example Image
`displayData.m` - Displays 2D data stored in a matrix
`drawLine.m` - Draws a line over an exsiting figure
`plotDataPoints.m` - Initialization for K-means centroids
`plotProgresskMeans.m` - Plots each step of K-means as it proceeds

`runkMeans.m` - Runs the K-means algorithm
`submit.m` - Submission script that sends your solutions to our servers
[$\star$] `pca.m` - Perform principal component analysis
[$\star$] `projectData.m` - Projects a data set into a lower dimensional space
[$\star$] `recoverData.m` - Recovers the original data from the projection
[$\star$] `findClosestCentroids.m` - Find closest centroids (used in K-means)
[$\star$] `computeCentroids.m` - Compute centroid means (used in K-means)
[$\star$] `kMeansInitCentroids.m` - Initialization for K-means centroids

$\star$ indicates files you will need to complete

Throughout the first part of the exercise, you will be using the script `ex7.m`, for the second part you will use `ex7_pca.m`. These scripts set up the dataset for the problems and make calls to functions that you will write. You are only required to modify functions in other files, by following the instructions in this assignment.

Where to get help

The exercises in this course use Octave[1] or MATLAB, a high-level programming language well-suited for numerical computations. If you do not have Octave or MATLAB installed, please refer to the installation instructions in the "Environment Setup Instructions" of the course website.

At the Octave/MATLAB command line, typing `help` followed by a function name displays documentation for a built-in function. For example, `help plot` will bring up help information for plotting. Further documentation for Octave functions can be found at the Octave documentation pages. MATLAB documentation can be found at the MATLAB documentation pages.

We also strongly encourage using the online **Discussions** to discuss exercises with other students. However, do not look at any source code written by others or share your source code with others.

---

1 K-means Clustering

In this this exercise, you will implement the K-means algorithm and use it for image compression. You will first start on an example 2D dataset that

[1] Octave is a free alternative to MATLAB. For the programming exercises, you are free to use either Octave or MATLAB.

will help you gain an intuition of how the K-means algorithm works. After that, you wil 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. You will be using `ex7.m` for this part of the exercise.

1.1 Implementing K-means

The K-means algorithm is a method to automatically cluster similar data examples together. Concretely, you are given a training set $\{x^{(1)}, ..., x^{(m)}\}$ (where $x^{(i)} \in \mathbb{R}^n$), and want to group the data into a few cohesive "clusters". The intuition behind 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.

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. Note that the converged solution may not always be ideal and depends on the initial setting of the centroids. Therefore, in practice the K-means algorithm is usually run a few times with different random initializations. One way to choose between these different solutions from different random initializations is to choose the one with the lowest cost function value (distortion).

You will implement the two phases of the K-means algorithm separately in the next sections.

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 we set

$$c^{(i)} := j \quad \text{that minimizes} \quad ||x^{(i)} - \mu_j||^2,$$

where $c^{(i)}$ is the index of the centroid that is closest to $x^{(i)}$, and μ_j is the position (value) of the j'th centroid. Note that $c^{(i)}$ corresponds to `idx(i)` in the starter code.

Your task is to complete the code in `findClosestCentroids.m`. This function takes the data matrix `X` and the locations of all centroids inside `centroids` and should output a one-dimensional array `idx` that holds the index (a value in $\{1, ..., K\}$, where K is total number of centroids) of the closest centroid to every training example.

You can implement this using a loop over every training example and every centroid.

Once you have completed the code in `findClosestCentroids.m`, the script `ex7.m` will run your code and you should see the output `[1 3 2]` corresponding to the centroid assignments for the first 3 examples.

You should now submit your solutions.

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 we set

$$\mu_k := \frac{1}{|C_k|} \sum_{i \in C_k} x^{(i)}$$

where C_k is the set of examples that are assigned to centroid k. Concretely, if two examples say $x^{(3)}$ and $x^{(5)}$ are assigned to centroid $k = 2$, then you should update $\mu_2 = \frac{1}{2}(x^{(3)} + x^{(5)})$.

You should now complete the code in `computeCentroids.m`. You can implement this function using a loop over the centroids. You can also use a loop over the examples; but if you can use a vectorized implementation that does not use such a loop, your code may run faster.

Once you have completed the code in `computeCentroids.m`, the script `ex7.m` will run your code and output the centroids after the first step of K-means.

You should now submit your solutions.

1.2 K-means on example dataset

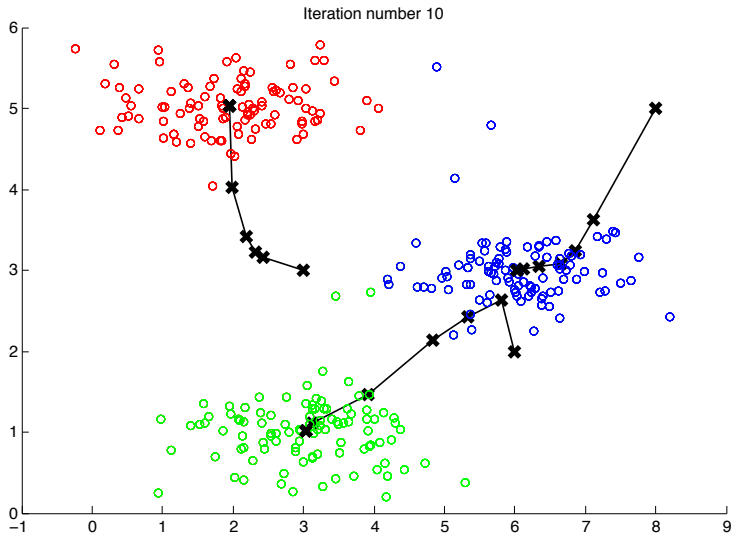

Figure 1: The expected output.

After you have completed the two functions (`findClosestCentroids` and `computeCentroids`), the next step in `ex7.m` will run the K-means algorithm on a toy 2D dataset to help you understand how K-means works. Your functions are called from inside the `runKmeans.m` script. We encourage you to take a look at the function to understand how it works. Notice that the code calls the two functions you implemented in a loop.

When you run the next step, the K-means code will produce a visualization that steps you through the progress of the algorithm at each iteration. Press *enter* multiple times to see how each step of the K-means algorithm changes the centroids and cluster assignments. At the end, your figure should look as the one displayed in Figure 1.

1.3 Random initialization

The initial assignments of centroids for the example dataset in `ex7.m` were designed so that you will see the same figure as in Figure 1. In practice, a good strategy for initializing the centroids is to select random examples from the training set.

In this part of the exercise, you should complete the function `kMeansInitCentroids.m` with the following code:

```
% 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), :);
```

The code above first randomly permutes the indices of the examples (using `randperm`). Then, it selects the first K examples based on the random permutation of the indices. This allows the examples to be selected at random without the risk of selecting the same example twice.

You do not need to make any submissions for this part of the exercise.

1.4 Image compression with K-means

Figure 2: The original 128x128 image.

In this exercise, you will apply K-means to image compression. In a

straightforward 24-bit color representation of an image,[2] each pixel is represented as three 8-bit unsigned integers (ranging from 0 to 255) that specify the red, green and blue intensity values. This encoding is often refered to as the RGB encoding. Our image contains thousands of colors, and in this part of the exercise, you will reduce the number of colors to 16 colors.

By making this reduction, it is possible to represent (compress) the photo in an efficient way. Specifically, you only need to store the RGB values of the 16 selected colors, and for each pixel in the image you now need to only store the index of the color at that location (where only 4 bits are necessary to represent 16 possibilities).

In this exercise, you will use the K-means algorithm to select the 16 colors that will be used to represent the compressed image. Concretely, you will 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. Once you have computed the cluster centroids on the image, you will then use the 16 colors to replace the pixels in the original image.

1.4.1 K-means on pixels

In Octave/MATLAB, images can be read in as follows:

```
% Load 128x128 color image (bird_small.png)
A = imread('bird_small.png');

% You will need to have installed the image package to used
% imread. If you do not have the image package installed, you
% should instead change the following line to
%
%   load('bird_small.mat'); % Loads the image into the variable A
```

This creates a three-dimensional matrix A whose first two indices identify a pixel position and whose last index represents red, green, or blue. For example, `A(50, 33, 3)` gives the blue intensity of the pixel at row 50 and column 33.

The code inside `ex7.m` first loads the image, and then reshapes it to create an $m \times 3$ matrix of pixel colors (where $m = 16384 = 128 \times 128$), and calls your K-means function on it.

After finding the top $K = 16$ colors to represent the image, you can now

[2] The provided photo used in this exercise belongs to Frank Wouters and is used with his permission.

assign each pixel position to its closest centroid using the `findClosestCentroids` function. This allows you to represent the original image using the centroid assignments of each pixel. Notice that you have significantly reduced the number of bits that are required to describe the image. The original image required 24 bits for each one of the 128×128 pixel locations, resulting in total size of $128 \times 128 \times 24 = 393,216$ 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 $16 \times 24 + 128 \times 128 \times 4 = 65,920$ bits, which corresponds to compressing the original image by about a factor of 6.

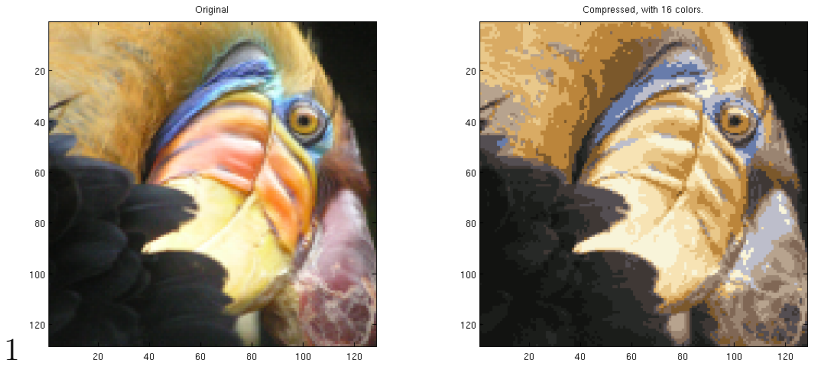

Figure 3: Original and reconstructed image (when using K-means to compress the image).

Finally, you can view the effects of the compression by reconstructing the image based only on the centroid assignments. Specifically, you can replace each pixel location with the mean of the centroid assigned to it. Figure 3 shows the reconstruction we obtained. Even though the resulting image retains most of the characteristics of the original, we also see some compression artifacts.

You do not need to make any submissions for this part of the exercise.

1.5 Optional (ungraded) exercise: Use your own image

In this exercise, modify the code we have supplied to run on one of your own images. Note that if your image is very large, then K-means can take a long time to run. Therefore, we recommend that you resize your images to managable sizes before running the code. You can also try to vary K to see the effects on the compression.

2 Principal Component Analysis

In this exercise, you will use principal component analysis (PCA) to perform dimensionality reduction. You will first experiment with an example 2D dataset to get intuition on how PCA works, and then use it on a bigger dataset of 5000 face image dataset.

The provided script, `ex7_pca.m`, will help you step through the first half of the exercise.

2.1 Example Dataset

To help you understand how PCA works, you will first start with a 2D dataset which has one direction of large variation and one of smaller variation. The script `ex7_pca.m` will plot the training data (Figure 4). In this part of the exercise, you will visualize what happens when you use PCA to reduce the data from 2D to 1D. In practice, you might want to reduce data from 256 to 50 dimensions, say; but using lower dimensional data in this example allows us to visualize the algorithms better.

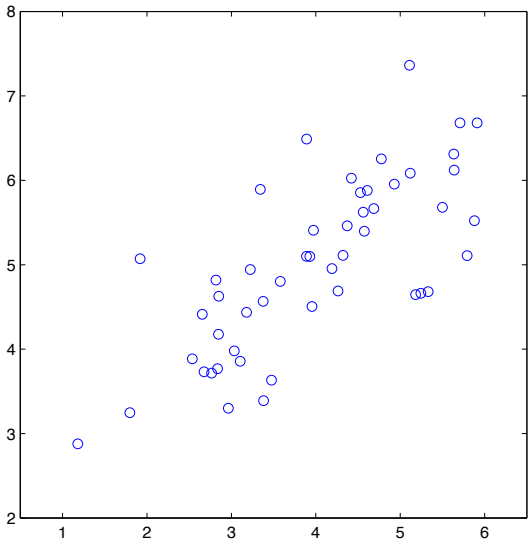

Figure 4: Example Dataset 1

2.2 Implementing PCA

In this part of the exercise, you will implement PCA. PCA consists of two computational steps: First, you compute the covariance matrix of the data.

Then, you use Octave/MATLAB's `SVD` function to compute the eigenvectors $U_1, U_2, \ldots, U_n$. These will correspond to the principal components of variation in the data.

Before using PCA, it is important to first 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. In the provided script `ex7_pca.m`, this normalization has been performed for you using the `featureNormalize` function.

After normalizing the data, you can run PCA to compute the principal components. You task is to complete the code in `pca.m` to compute the principal components of the dataset. First, you should compute the covariance matrix of the data, which is given by:

$$\Sigma = \frac{1}{m} X^T X$$

where X is the data matrix with examples in rows, and m is the number of examples. Note that Σ is a $n \times n$ matrix and not the summation operator.

After computing the covariance matrix, you can run SVD on it to compute the principal components. In Octave/MATLAB, you can run SVD with the following command: `[U, S, V] = svd(Sigma)`, where `U` will contain the principal components and `S` will contain a diagonal matrix.

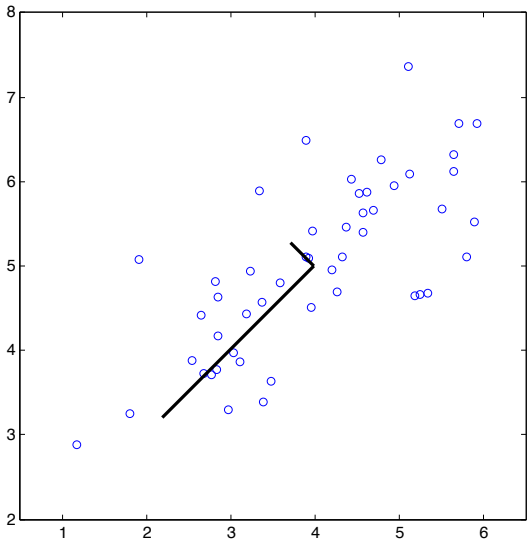

Figure 5: Computed eigenvectors of the dataset

Once you have completed `pca.m`, the `ex7_pca.m` script will run PCA on the example dataset and plot the corresponding principal components found

(Figure 5). The script will also output the top principal component (eigenvector) found, and you should expect to see an output of about `[-0.707 -0.707]`. (It is possible that Octave/MATLAB may instead output the negative of this, since U_1 and $-U_1$ are equally valid choices for the first principal component.)

You should now submit your solutions.

2.3 Dimensionality Reduction with PCA

After computing the principal components, you can use them to reduce the feature dimension of your dataset by projecting each example onto a lower dimensional space, $x^{(i)} \rightarrow z^{(i)}$ (e.g., projecting the data from 2D to 1D). In this part of the exercise, you will use the eigenvectors returned by PCA and project the example dataset into a 1-dimensional space.

In practice, if you were using a learning algorithm such as linear regression or perhaps neural networks, you could now use the projected data instead of the original data. By using the projected data, you can train your model faster as there are less dimensions in the input.

2.3.1 Projecting the data onto the principal components

You should now complete the code in `projectData.m`. Specifically, you are given a dataset `X`, the principal components `U`, and the desired number of dimensions to reduce to `K`. You should project each example in `X` onto the top `K` components in `U`. Note that the top `K` components in `U` are given by the first `K` columns of U, that is `U_reduce = U(:, 1:K)`.

Once you have completed the code in `projectData.m`, `ex7_pca.m` will project the first example onto the first dimension and you should see a value of about `1.481` (or possibly `-1.481`, if you got $-U_1$ instead of U_1).

You should now submit your solutions.

2.3.2 Reconstructing an approximation of the data

After projecting the data onto the lower dimensional space, you can approximately recover the data by projecting them back onto the original high dimensional space. Your task is to complete `recoverData.m` to project each example in `Z` back onto the original space and return the recovered approximation in `X_rec`.

Once you have completed the code in `recoverData.m`, `ex7_pca.m` will recover an approximation of the first example and you should see a value of about `[-1.047 -1.047]`.

You should now submit your solutions.

2.3.3 Visualizing the projections

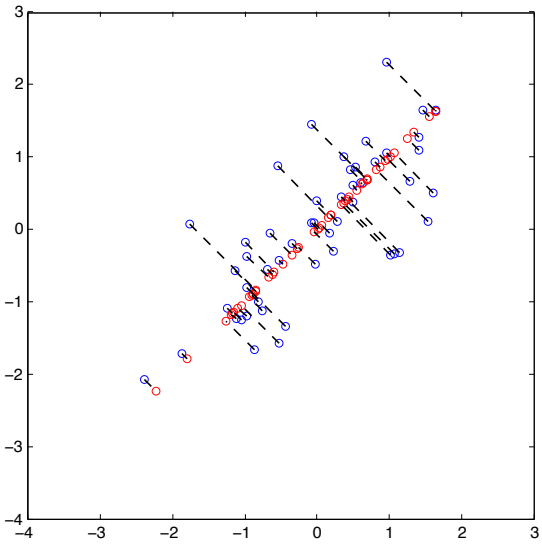

Figure 6: The normalized and projected data after PCA.

After completing both `projectData` and `recoverData`, `ex7_pca.m` will now perform both the projection and approximate reconstruction to show how the projection affects the data. In Figure 6, the original data points are indicated with the blue circles, while the projected data points are indicated with the red circles. The projection effectively only retains the information in the direction given by U_1.

2.4 Face Image Dataset

In this part of the exercise, you will run PCA on face images to see how it can be used in practice for dimension reduction. The dataset `ex7faces.mat` contains a dataset[3] `X` of face images, each 32×32 in grayscale. Each row of `X` corresponds to one face image (a row vector of length 1024). The next

[3]This dataset was based on a cropped version of the labeled faces in the wild dataset.

step in `ex7_pca.m` will load and visualize the first 100 of these face images (Figure 7).

Figure 7: Faces dataset

2.4.1 PCA on Faces

To run PCA on the face dataset, we first normalize the dataset by subtracting the mean of each feature from the data matrix `X`. The script `ex7_pca.m` will do this for you and then run your PCA code. After running PCA, you will obtain the principal components of the dataset. Notice that each principal component in `U` (each row) is a vector of length n (where for the face dataset, $n = 1024$). It turns out that we can visualize these principal components by reshaping each of them into a 32×32 matrix that corresponds to the pixels in the original dataset. The script `ex7_pca.m` displays the first 36 principal components that describe the largest variations (Figure 8). If you want, you can also change the code to display more principal components to see how they capture more and more details.

2.4.2 Dimensionality Reduction

Now that you have computed the principal components for the face dataset, you can use it to reduce the dimension of the face dataset. This allows you to use your learning algorithm with a smaller input size (e.g., 100 dimensions) instead of the original 1024 dimensions. This can help speed up your learning algorithm.

Figure 8: Principal components on the face dataset

Figure 9: Original images of faces and ones reconstructed from only the top 100 principal components.

The next part in `ex7_pca.m` will project the face dataset onto only the first 100 principal components. Concretely, each face image is now described by a vector $z^{(i)} \in \mathbb{R}^{100}$.

To understand what is lost in the dimension reduction, you can recover the data using only the projected dataset. In `ex7_pca.m`, an approximate recovery of the data is performed and the original and projected face images are displayed side by side (Figure 9). From the reconstruction, you can observe that the general structure and appearance of the face are kept while the fine details are lost. This is a remarkable reduction (more than 10×) in

the dataset size that can help speed up your learning algorithm significantly. For example, if you were training a neural network to perform person recognition (gven a face image, predict the identitfy of the person), you can use the dimension reduced input of only a 100 dimensions instead of the original pixels.

2.5 Optional (ungraded) exercise: PCA for visualization

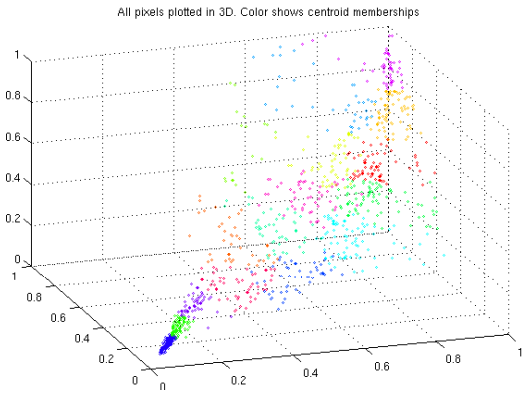

Figure 10: Original data in 3D

In the earlier K-means image compression exercise, you used the K-means algorithm in the 3-dimensional RGB space. In the last part of the `ex7_pca.m` script, we have provided code to visualize the final pixel assignments in this 3D space using the `scatter3` function. Each data point is colored according to the cluster it has been assigned to. You can drag your mouse on the figure to rotate and inspect this data in 3 dimensions.

It turns out that visualizing datasets in 3 dimensions or greater can be cumbersome. Therefore, it is often desirable to only display the data in 2D even at the cost of losing some information. In practice, PCA is often used to reduce the dimensionality of data for visualization purposes. In the next part of `ex7_pca.m`, the script will apply your implementation of PCA to the 3-dimensional data to reduce it to 2 dimensions and visualize the result in a 2D scatter plot. The PCA projection can be thought of as a rotation that selects the view that maximizes the spread of the data, which often corresponds to the "best" view.

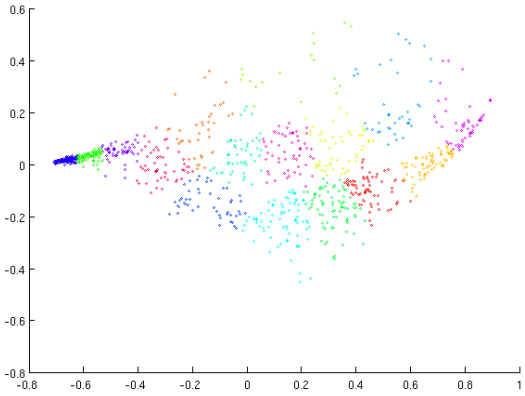

Figure 11: 2D visualization produced using PCA

Submission and Grading

After completing various parts of the assignment, be sure to use the `submit` function system to submit your solutions to our servers. The following is a breakdown of how each part of this exercise is scored.

Part	**Submitted File**	**Points**
Find Closest Centroids	`findClosestCentroids.m`	30 points
Compute Centroid Means	`computeCentroids.m`	30 points
PCA	`pca.m`	20 points
Project Data	`projectData.m`	10 points
Recover Data	`recoverData.m`	10 points
Total Points		100 points

You are allowed to submit your solutions multiple times, and we will take only the highest score into consideration.