EXERCICE DE PROGRAMMATION 4 :

K-means Clustering and Principal Component Analysis

GAO BIN

06/01/2017

Introduction

Dans ce TP, dans la première on implémente l'algorithme de K-means clustering et l'appliquer cet algorithme pour compresser une image. Dans la deuxième partie, on découvrira une façon de simplifier les images.

1. K-means Clustering

Dans cette partie on utilise l'algorithme des K-means dans le but de compresser des images. on commence d'abord par un exemple de jeu de données 2D qui nous aidons à acquérir une intuition de la façon dont fonctionne l'algorithme K-means. Après on utilise l'algorithme K-means pour la compression d'image.

1.1 Implementing K-means

L'algorithme des K-means prend un ensemble{ x1, x2, … , xn} et regroupe en des «clusters».

1.1.1 Finding closest centroids

Dans cette partie, on doit finir le code dans findClosestCentroids.m, cette fonction prend la matrice de données X et les emplacements de tous les centroïdes à l'intérieur centroïdes et devrait produire un tableau unidimensionnel idx qui Indice (une valeur dans {1, ..., K}, où K est le nombre total de centroïdes) du centre de gravité le plus proche de chaque exemple de formation.

On a le code suivant :

```
24  m = size(X  , 1);
25
26 for i = 1 : m,
27    T = [];
28    for j = 1 : K,
29      T = [T ; X(i,:)];
30    end
31    [Max , idx(i)] = min(sum((T - centroids).^2 , 2));
32 end
33
```

On fait appel au programme, donc on obtient:

```
Closest centroids for the first 3 examples:
 1 3 2
(the closest centroids should be 1, 3, 2 respectively)
Program paused. Press enter to continue.
```

1.1.2 Computing centroid means

La formule du cours:

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

On complète donc le code de la fonction dans computeCentroids.m:

```
29 for i = 1:K;
30     centroids(i,:) = (X'*(idx == i))/sum(idx ==i);
31 end
```

On fait appel au programme, donc on obtient:

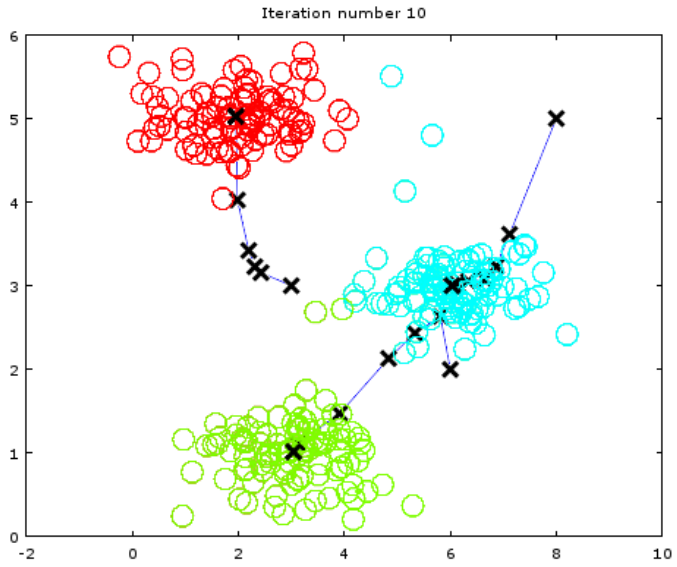
```
Centroids computed after initial finding of closest centroids:
 2.428301 3.157924
 5.813503 2.633656
 7.119387 3.616684

(the centroids should be
   [ 2.428301 3.157924 ]
   [ 5.813503 2.633656 ]
   [ 7.119387 3.616684 ]

Program paused. Press enter to continue.
```

1.2 K-means on example dataset

Dans cette partie, le programme met en place un exemple en 2D pour aider à la compréhension du processus.



La position initiale des centoïdes dans cet exemple sont situés de façon à ce que on puisse voir l'algorithme en action. Mais il vaut mieux sélectionner des points au hasard parmi les exemples d'apprentissage. Pour cela on complète le programme comme indiqué.

1.3 Random initialization

On complète donc le code de la fonction dans kMeansInitCentroids.m:

```
21 randidx = randperm(size(X,1));
22 centroids = X(randidx(1:K),:);
23
```

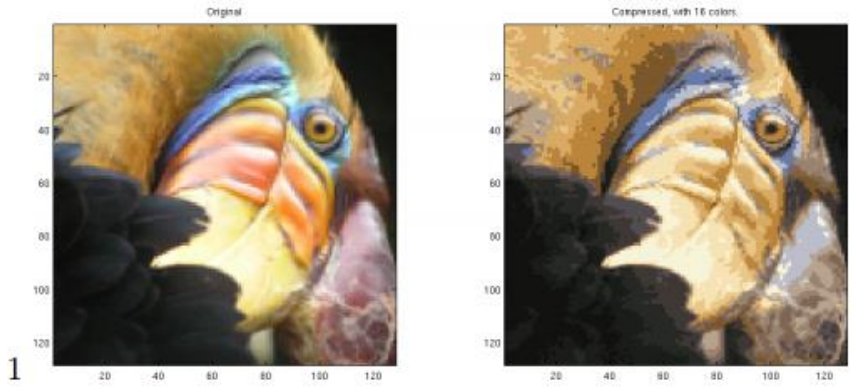
Le code ci-dessus permute d'abord aléatoirement les indices des exemples (en utilisant randperm). Ensuite, il sélectionne les premiers K exemples basés sur le hasard permutation

des indices. Cela permet de sélectionner les exemples au hasard sans risque de choisir deux fois le même exemple.

1.4 Image compression with K-means

Dans cette partie, on applique K-means à la compression d'image. on compresse l'image en diminuant le nombre de couleur. Après être passée par l'algorithme, l'image n'a plus que 16 couleurs qui sont une sorte de moyenne de leur groupe.

1.4.1 K-means on pixels



La figure montre la reconstruction que nous avons obtenue. Même si l'image résultante conserve la plupart des caractéristiques de l'original, nous voyons également une certaine compression artefacts.

2. Principal Component Analysis

Dans cette partie, on utilise l'analyse des composantes principales (PCA) pou réduire de la dimensionnalité.

2. 2 Implementing PCA

Pour lancer l'analyse, il faut avoir auparavant normalisé les données ce qui est fait dans le programme featureNormalize. Pour implémenter la PCA, il faut calculer la matrice de la covariance.

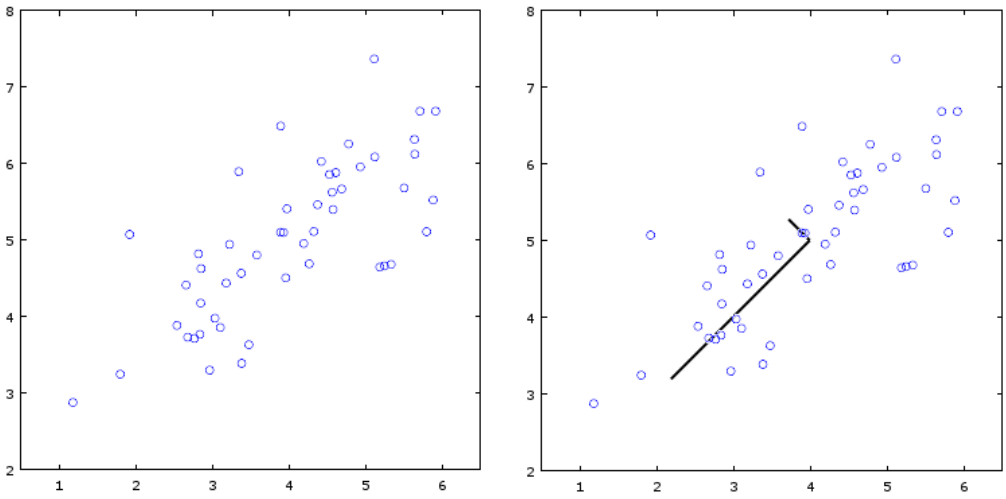
On a la formule suivant :

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

On complète donc le code de la fonction dans pca.m:

```
23  sigma = X'*X/m;
24  [u,s,v] = svd(sigma);
```

On fait appel au programme， donc on obtient alors la figure :



On peut obtenir les directions principales des données, grâce aux vecteur propre dont on a même la valeur:

```
Top eigenvector:
 U(:,1) = -0.707107 -0.707107
```

2. 3 Dimensionality Reduction with PCA

Dans cette partie de l'exercice, vous utiliserez les vecteurs propres retournés par l'ACP et projeter l'exemple de jeu de données dans un espace à une dimension. Une fois qu'on a la direction principale, on peut réduire la dimension de 2D à 1D en projetant les données orthogonalement à la direction principale.

Donc on remplit le code dans la fonction projectData.m :

```
21   Z = X*U(:,1:K);
```

Pour revenir à l'espace initial, on complète la fonction recoverData.m :

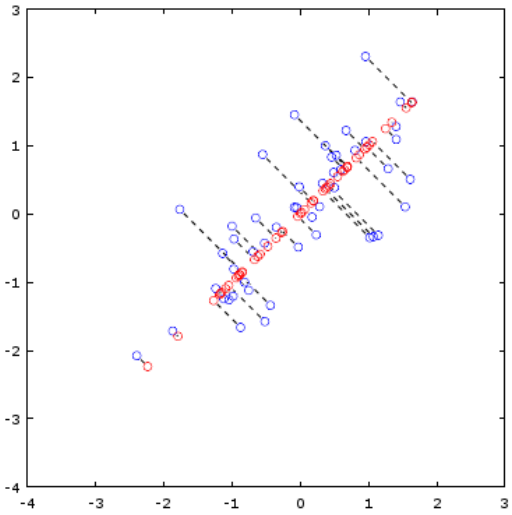
```
24   X_rec = Z*U(:,1:K)';
25   % ====================
```

On fait appel au programme， donc on obtient :

```
Projection of the first example: 1.481274

(this value should be about 1.481274)

Approximation of the first example: -1.047419 -1.047419
```



On trouve les valeur de projections demandées : 1.481274.

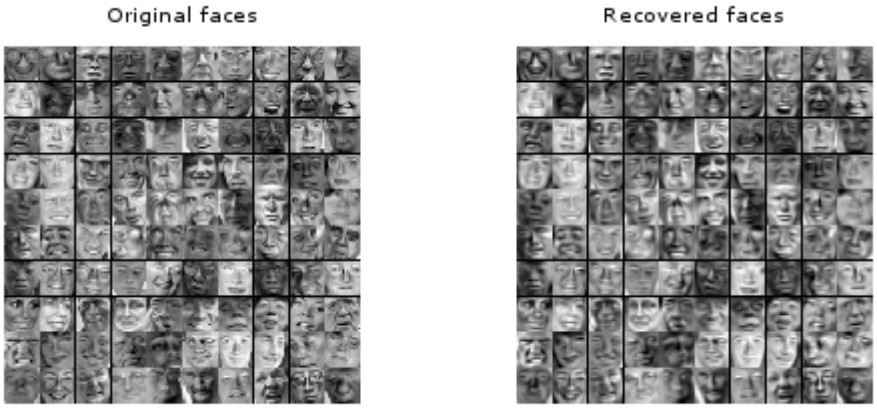
2. 4 Face Image Dataset

Dans cette partie de l'exercice, on exécute PCA sur les images de visage pour voir comment il peut être utilisé dans la pratique pour réduire les dimensions. Chaque visage comprend 1024 pixels et est donc considéré comme un vecteur de longueur 1024.

2. 4.1 PCA on Faces

Pour exécuter PCA sur l'ensemble de données face, nous normalisons d'abord la base de données en soustrayant la moyenne de chaque entité de la matrice de données X. Il s'avère que nous pouvons visualiser ces remodelant chacun d'eux en une matrice 32 × 32 qui correspond aux pixels dans le jeu de données original.

On fait appel au programme， donc on obtient suivant:

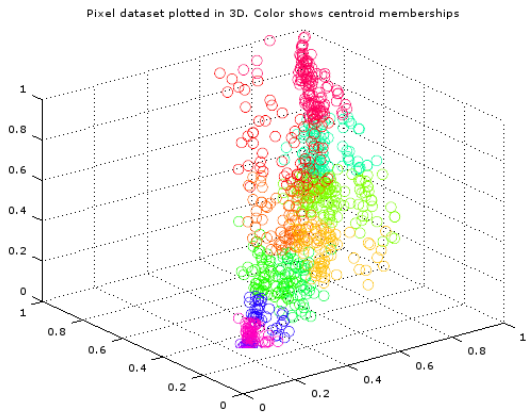


2. 5 PCA for visualization

Dans cette partie, on veut que des données 3D soient présentées en 2D pour faciliter la lecture de graph. PCA projette les données dans le plan qui maximise la dispersion des données.

On fait appel au programme， donc on obtient :

```
K-Means iteration 1/10...
K-Means iteration 2/10...
K-Means iteration 3/10...
K-Means iteration 4/10...
K-Means iteration 5/10...
K-Means iteration 6/10...
K-Means iteration 7/10...
K-Means iteration 8/10...
K-Means iteration 9/10...
K-Means iteration 10/10...
Program paused. Press enter to continue.
```



Il s'avère que la visualisation de données en 3 dimensions ou plus peut être encombrant. Par conséquent, on utilise la propriété de PCA qui diminue les dimensions et on obtient un affichage en 2D. En pratique, la PCA est souvent utilisée pour réduire la dimensionnalité des données à des fins de visualisation.