

Minicurso de MATLAB - Nível Intermediário

Paulo Oliveira Lenzi Valente

2016.1

Conteúdo

Introdução	3
1 K-Means	3
1.1 Implementação do K-Means	3
2 Aplicando o K-Means para processamento de imagens	8
2.1 Compressão por redução de cores	8
2.2 Limpeza de imagem <i>grayscale</i> e separação de caracteres	11
3 Conclusão	15
Referências	15

Introdução

Este curso tem como objetivos:

- Apresentar o algoritmo K-Means e implementá-lo no MATLAB
- Aplicações do algoritmo para compressão e processamento de imagens

1 K-Means

O K-Means é um método de clusterização de dados. Ele busca agrupar os dados observados em K clusters (grupamentos) nos quais cada um dos N dados observados pertence ao cluster com a centróide mais próxima. Está dentro do grupo dos algoritmos de aprendizado não supervisionado e funciona de acordo com os seguintes passos:

- Inicialização, normalmente aleatória, das K centróides
- Para cada um dos N pontos, é escolhida a centróide mais próxima.
- A centróide se move em direção ao centro de massa (ponto médio) de seu cluster, determinado no passo anterior
- Repetição dos dois itens anteriores até a convergência. Centróides iniciais distintas podem convergir para soluções distintas.

1.1 Implementação do K-Means

A seguir, vamos implementar o algoritmo no MATLAB. Feito isso, vamos ver duas aplicações do algoritmo para processamento de imagem. Nossa implementação será baseada naquela introduzida no curso [1].

Algoritmo geral:

```
% dados a serem processados estão no vetor X, e o ...  
    algoritmo tem ordem K  
centroids = initializeCentroids(X, K);  
  
%maxiter e o numero maximo de iteracoes  
maxiter = 20;  
for iter = 1:maxiter  
    % Passo de designacao do cluster. cluster(i) indica ...  
        qual centroide foi selecionada para um dado ponto  
    clusters = buscarCentroideProxima(X, centroids);  
  
    %mover as centroids em direcao ao centro de seus ...  
        respectivos clusters  
    centroids = computarMedias(X, clusters, K);  
end
```

Inicialização aleatória de centróides:

```
function [centroids] = initializeCentroids(train_set, K)
% [centroids] = initializeCentroids(train_set, K)
% Essa funcao recebe um set de treino e a ordem K do ...
% algoritmo e retorna
% as centroides iniciais para a execucao do algoritmo
indices = randperm(size(train_set,1));
centroids = train_set(indices(1:K), :);
end
```

Busca da centróide mais próxima:

```
function clusters = buscarCentroideProxima(X, centroids)

K = size(centroids, 1);

%prealocando espaco
C = zeros(size(X,1), K);
for index=1:K
    % calcula para cada ponto, sua distancia quadratica em ...
    % relacao a cada uma das centroides

    % As linhas abaixo podem ser condensadas em uma so, ...
    % para evitar chamadas desnecessarias de atribuicao ...
    % de valores.

    distancia = bsxfun(@minus,X,centroids(index,:));
    distancia = distancia.^2
    C(:,index) = sum( distancia, 2)
end
% binariza a matriz C, colocando 1 apenas aonde ha a ...
% minima distancia
% quadratica, o que corresponde a melhor centroide para ...
% aquele ponto
% Depois, a multiplicacao pelo vetor 1:K substitui cada ...
% linha da matriz pelo
% indice da centroide correspondente

clusters = bsxfun(@eq,C,min(C,[],2))*(1:K)';
end
```

Cálculo das novas centróides:

```
function centroids = computarMedias(X, clusters, K)









    [~, n] = size(X);
    centroids = zeros(K, n);
    for m=1:K
        %selecionando os elementos pertencentes ao m-esimo ...
        cluster
        clusterK = X(clusters==m,:);
        %substituindo a centroide pelo "centro de massa" do ...
        cluster
        centroids(m,:) = mean(clusterK);
    end
end
```

2 Aplicando o K-Means para processamento de imagens

Agora que temos o algoritmo implementado, podemos utilizá-lo para processar dados. Primeiramente, vamos ver como ele pode comprimir imagens através da redução de cores. Além disso, vamos utilizar o K-Means para tratar ruído em fotos de quadro branco, o que facilita o processamento das fotos para separar os caracteres contidos no quadro. Esse processo pode acelerar a coleta de dados para treinamento de algoritmos de OCR (textitOptical Character Recognition).

2.1 Compressão por redução de cores

Em uma imagem em RGB, na representação de 24-bits, cada cor pode assumir 256 valores, variando de 0 a 255 - respectivamente Vermelho(*Red*), Verde(*Green*) e Azul(*Blue*). Essas cores podem ser representadas, por exemplo, como conjuntos de números, o que costumamos ver em programas de edição de imagem, ou como um conjunto de caracteres hexadecimais, muito utilizado na internet. O primeiro tem um formato do tipo (R,G,B) e o segundo, da forma como encontramos em desenvolvimento web assume um formato #RRGGBB, onde cada um dos segmentos pode assumir valores de 00 a FF. Assim, podemos ver cores sendo representadas de diferentes maneiras:

Cor	Hexadecimal	RGB	
Vermelho	#FF0000	(255,0,0)	
Verde	#00FF00	(0,255,0)	
Azul	#0000FF	(0,0,255)	
Preto	#000000	(0,0,0)	
Branco	#FFFFFF	(255,255,255)	
Amarelo	#FFFF00	(255,255,0)	
Ciano	#00FFFF	(0,255,255)	
Magenta	#FF00FF	(255,0,255)	

Sabendo que, no mundo vir-

tual, cores são números, podemos entender que, no Matlab, uma imagem será representada como uma matriz. As duas dimensões convencionais da matriz representam a posição dos *pixels* e uma terceira dimensão permite que se percorram as cores, que podem estar, dentre vários sistemas, em RGB, resultando em 3 posições nessa terceira dimensão ou uma posição no caso do sistema *grayscale* - preto e branco, também chamado de BW ou PB.

Para aplicarmos o K-Means na compressão de imagens, vamos carregar a imagem e tratar cada pixel como um elemento do nosso conjunto de treinamento, agrupando os pixels por suas cores.

```

%% Carregando a imagem e convertendo para numero
img = imread('imagem.png');
img = double(img)/255;

%% Transformando a matriz de pixels em uma matriz de 3 ...
    colunas, onde cada linha e um pixel
X = reshape(img, size(img,1) * size(img,2), 3);

%% Configuracoes iniciais do K-Means
K = 16;
max_iters = 10;
centroids = initializeCentroids(X, K);

%% Execucao do algoritmo
for iter = 1:maxiter
    % Passo de designacao do cluster. cluster(iter) indica ...
        qual centroide foi selecionada para um dado ponto
    clusters = buscarCentroideProxima(X, centroids);

    %mover as centroids em direcao ao centro de seus ...
        respectivos clusters
    centroids = computarMedias(X, clusters, K);
end

idx = buscarCentroideProxima(X, centroids);
X_recuperado = centroids(idx,:);
img_comprimida = ...
    reshape(X_recuperado, size(img,1), size(img,2), 3);

%% Demonstracao do resultado

figure;
subplot(1,2,1);
title('Original');
imshow(img);
subplot(1,2,2);
title('Comprimida');
imshow(img_comprimida);

```

É interessante notar que pode-se utilizar uma palheta de cores pré-definida como o conjunto inicial de centróides, em vez de um conjunto totalmente aleatório, algo que faremos no próximo algoritmo.

2.2 Limpeza de imagem *grayscale* e separação de caracteres

Agora que conseguimos aplicar o K-Means para o processamento de uma imagem, vamos utilizar o algoritmo para tentar remover o ruído de uma imagem preto e branco. Mais especificamente, essa imagem será uma foto de conteúdo escrito em quadro branco ou texto digitado. Após a remoção de ruído utilizando o K-Means, vamos separar os diversos candidatos a caracteres em imagens distintas, visando construir um conjunto de treinamento para OCR (*Optical Character Recognition*). Assim, em vez de termos o trabalho de recortar cada um dos caracteres, podemos automatizar este processo, tendo apenas o trabalho de etiquetar (*label*) cada elemento do conjunto de treinamento.

2.2.1 Algoritmo de limpeza da imagem

```
function [ COLOR, BW ] = limparImagem( filename, res, ...
    invert )
%LIMPARIMAGEM Recebe uma imagem e remove o ruído baseado ...
em K-Means
    if nargin == 1
        res = 0.5;
        invert = 1;
    elseif nargin == 2
        invert = 1;
    elseif nargin == 3
    else
        fprintf('Numero invalido de argumentos\n');
        return
    end

    COLOR = imresize(imread(filename),res);
    A = double(rgb2gray(COLOR));
    COLOR = im2double(COLOR);

    A = A/max(A(:));
    img_size = size(A);
    X = reshape(A, img_size(1) * img_size(2),1);
    %initial_centroids = kMeansInitCentroids(X, K);
    X = (X)/max(X(:));
    minX = min(X(:));
    maxX = max(X(:));
    meanX = mean(X(:));
    centroids = [minX; meanX; maxX];
    %initial_centroids =[minX; maxX];
    %[centroids] = runkMeans(X, initial_centroids, ...
        max_iters, false);
    K = 3;
    maxiter = 12;
    for iter = 1:maxiter
        % Passo de designacao do cluster. cluster(i) ...
        indica qual centroide foi selecionada para um ...
        dado ponto
        clusters = buscarCentroideProxima(X, centroids);
        %mover as centroids em direcao ao centro de seus ...
        respectivos clusters
        centroids = computarMedias(X, clusters, K);
    end
```

```

idx = buscarCentroideProxima(X, centroids);
X_recovered = centroids(idx,:);
X_recovered = double(reshape(X_recovered, ...
    img_size(1), img_size(2),1) >= centroids(2));
X_recovered = X_recovered - min(X_recovered(:));
BW = ~(X_recovered/max(X_recovered(:)));
B = zeros(size(COLOR,1),size(COLOR,2),size(COLOR,3));
for index=1:size(COLOR,3)
    B(:,:,index) = BW;
end
if invert == 1
    C = ~B;
else
    C = 0;
end
COLOR = COLOR .* double(B) + double(C);
end

```

2.2.2 Algoritmo de separação dos caracteres

```
nome = './image1.jpg';
[color, clean] = limparImagem(nome,0.5);
imshow(~clean);

%utilizando a detecção preto e branco para mascarar os ...
%caracteres na imagem
%colorida
T1 = 200;
clean = bwareaopen(clean,T1);
chars = bwconncomp(clean);
Z = zeros(chars.ImageSize);
bw_list = cell(chars.NumObjects);
for index=1:chars.NumObjects
    mask = Z;
    mask(chars.PixelIdxList{index}) = 1;
    bw_list{index} = mask;
end

bounded_img_list = cell(chars.NumObjects);
for index=1:chars.NumObjects
    [~, min_x, max_x, min_y, max_y] = ...
        findBoundingBox(bw_list{index});
    bounded_img_list{index} = ...
        color(min_y:max_y,min_x:max_x,:);
    figure;
    imshow(bounded_img_list{index});
end
```

Este algoritmo aplica a função de limpeza de ruído mostrada anteriormente, para depois utilizar duas funções do Matlab que, juntas, removem um pouco mais do ruído e retornam todos os grupamentos de pixels diferentes de 0. Estas nós aplicamos sobre a imagem em PB que nos é retornada pela função de limpeza. Com isso, nós conseguimos construir máscaras de pixels para separar cada um dos caracteres sem perdermos suas cores. A segunda parte do algoritmo reconstrói cada uma das imagens com um único caracter e encontra a sua *bounding box*, ou seja, o menor retângulo que envolve todo o conteúdo útil da imagem. Para isso, usamos as máscaras de caracteres que construímos anteriormente, aplicando-as sobre a imagem que estamos analisando.

3 Conclusão

Neste curso, estudamos o algoritmo K-Means e como podemos aplicá-lo para processamento de imagens. A segunda aplicação que vimos é, na verdade, a solução para um problema auxiliar de OCR. Assim, temos um programa para nos auxiliar na construção de uma base de dados para treinamento de um algoritmo supervisionado para reconhecimento de caracteres. Também vimos a diferença que as centróides iniciais podem fazer no resultado final - um problema relacionado ao mínimo local de uma função.

Referências

- [1] Curso sobre Aprendizado de Máquina (*Coursera*):
<https://www.coursera.org/learn/machine-learning> Deste curso foi retirada a teoria do K-Means e o algoritmo de compressão de cores
- [2] Página com material adicional sobre Matlab: <http://polvalente.github.io/downloads>