COPPE-UFRJ

MeanShift Segmentation

Ricardo Marroquim

data entrega:
pós-graduação: 24/06/2016
graduação: 30/06/2016

1 OBJETIVO

Gerar uma filtragem e segmentação da imagem utilizando o algoritmo Mean-Shift.

A entrada do seu algoritmo deve ser uma imagem, e a saída a imagem filtrada ou a imagem com as regiões pintadas. Você deve especificar dois parâmetros, h_s e h_r, que serão descritos abaixo.

2 MEAN SHIFT

O objetivo do meanshift é encontrar máximos e mínimos de uma função a partir das amostras dadas. No nosso caso as amostras são os pixels de uma imagem, e a nossa função levará em conta a posição e cor do pixel.

Mas antes de falarmos em imagens, vamos primeiro analisar a teoria do meanshift nesta seção. Nas seções seguintes detalharemos como utilizá-lo para o caso de filtragem e segmentação de imagens nas seções seguintes.

O meanshift está baseado na seguinte premissa: se modelarmos a distribuição de densidade do nosso domínio amostral, poderemos seguir as direções de ascendência dos gradientes para encontrar os pontos máximos. Ou seja, de alguma forma precisamos transfomar nosso

espaço de amostras em um espaço contínuo, para podermos encontrar o gradiente em qualquer posição.

Podemos aproximar localmente a função de densidade coletando informações de amostras vizinhas. Para tanto vamos usar um kernel K centrado em $\mathbf{x}$, e considerando n amostras:

$$f(\mathbf{x}) = \left(\frac{1}{n}\right) \sum_{n}^{i=1} \mathrm{K}(\mathbf{x}_i - \mathbf{x}; h)$$

Você pode pensar no kernel como uma pequena janela centrada em um ponto. Esta função acima é uma estimativa da densidade no ponto $\mathbf{x}$. Note que $\mathbf{x}$ é um vetor d-dimensional, onde d depende do problema, mais a frente definiremos d para o nosso caso. K é o kernel escolhido, e h é um fator de suavização do kernel. Vamos utilizar o kernel gaussiano para este trabalho:

$$\mathrm{K}(\mathbf{u}, h) = \frac{1}{(2\pi)^{d/2} h^d} \exp\left(-\frac{1}{2}\frac{\|\mathbf{u}\|^2}{h}\right)$$

onde $\mathbf{u}$ é um vetor partindo do centro do kernel, e $\|\mathbf{u}\|$ sua norma, ou seja, a distância para o centro do kernel. O kernel gaussiano coloca mais peso para amostras perto do centro do kernel. Podemos separar um kernel em duas partes, o seu "profile":

$$\mathrm{k}(\mathbf{u}, h) = \exp\left(-\frac{1}{2}\frac{\|\mathbf{x_i} - \mathbf{x}\|^2}{h}\right)$$

e uma parte constante C:

$$\mathrm{C} = \frac{1}{(2\pi)^{d/2} n h^d}$$

Agora a nossa função kernel fica da seguinte forma:

$$f(\mathbf{x}) = \mathrm{C} \sum_{n}^{i=1} \mathrm{k}\left(\left\|\frac{\mathbf{x}_i - \mathbf{x}}{h}\right\|^2\right)$$

Para encontrar os máximos e mínimos de uma função, precisamos saber o ponto onde o seu gradiente é zero $\nabla f(\mathbf{x}) = 0$. Derivando a função e removendo alguns termos (veja o artigo para mais detalhes), chegamos ao vetor *mean shift* $\mathbf{x}$:

$$\mathbf{y} = \mathbf{m}_h(\mathbf{x}) + \mathbf{x} = \left[\frac{\sum_i \mathbf{x}_i g\left(\left\|\frac{\mathbf{x}_i - \mathbf{x}}{h}\right\|^2\right)}{\sum_i g\left(\left\|\frac{\mathbf{x}_i - \mathbf{x}}{h}\right\|^2\right)}\right] \tag{2.1}$$

onde $g = -k'$ (menos a derivado do profile do kernel), e h é o parâmetro de suavização.

OBS: estamos fazendo uma operação muito simples: é apenas a média ponderada dos vizinhos no ponto $\mathbf{x}$, onde cada amostra $\mathbf{x}_i$ tem peso dado por g.

Agora, precisamos apenas caminhar na direção dada pelo vetor *mean shift* até convergência, ou seja, até que o vetor tenha norma zero, ou suficientemente pequena.

O algoritmo fica muito simples:

1. computar o vetor *mean shift* $\mathbf{m}_h(\mathbf{x})$ no ponto $\mathbf{x}$ (centro do kernel)
2. transladar o centro do kernel do ponto $\mathbf{x}$ para o ponto $\mathbf{x} + \mathbf{m}_h(\mathbf{x})$
3. repetir os passos 1 e 2 até que $\|\mathbf{m}_h(\mathbf{x})\| < \epsilon$

Na prática, inicializamos o algoritmo com vários kernels em posições $\mathbf{x}$ diferentes, e executamos o algoritmo para cada kernel independentemente. Grupos de kernels irão convergir para máximos diferentes, e com kernels suficientes, todos os máximos da função serão encontrados.

3 MeanShift em Imagens

Uma vez que entendemos a lógica do método mean shift, podemos aplicá-lo a imagens. Note que o nosso espaço de busca não será a imagem, mas sim um espaço de dimensão maior.

Vamos pensar no pixel como um vetor de cinco dimensões $p(x, y, r, g, b)$, e executar o algoritmo mean shift no espaço d-dimensional, onde $d = 5$.

Um passo intermediário para realizar a segmentação meanshift em imagens, é realizar uma filtragem com o mesmo algoritmo. Por isso vamos detalhar primeiro a filtragem, e depois transformá-la em segmentação.

3.1 MeanShift Filtering

Inicializa-se o algoritmo, com um kernel centrado sobre cada amostra (ou pixel) no espaço 5-dimensional. Cada kernel nada mais é do que uma réplica do ponto inicial, ou seja, para cada amostra (pixel) $\mathbf{x}_i$ incializamos um kernel $\mathbf{y}_i^0$ no passo $t = 0$:

$$\mathbf{y}_i^0 = \mathbf{x}_i$$

Se o kernel usado for o gaussiano, o vetor meanshift na Equação 2.1 terá a seguinte forma:

$$y_i^{t+1} = \left[\frac{\sum_j \mathbf{y}_i^t e^{-\left\| \frac{\mathbf{x}_j - \mathbf{y}_i^t}{h^2} \right\|^2}}{\sum_j e^{-\left\| \frac{\mathbf{x}_j - \mathbf{y}_i^t}{h^2} \right\|^2}} \right]$$

O algoritmo final "força bruta"é bem simples, inicialize os kernels e a cada iteração mova para a nova posição y^{t+1} fazendo a média ponderada com todos os pontos iniciais.

Após sucessivas iterações os pontos convergirão para os máximos que representam as regiões. Note que as amostras não se movem, só os pontos (kernels)! Quando todos os vetores forem iguais a zero, ou abaixo de um limiar, o algoritmo convergiu.

Para finalizar a filtragem, crie uma nova imagem onde cada pixel recebe a cor do seu kernel respectivo (aonde o kernel que começou no pixel foi parar: quais os valores rgb após convergência).

Este algoritmo ingênuo funciona, porém, existem alguns problemas que precisamos resolver:

3.1.1 Modificando os pesos

A parte espacial (x, y) possui uma escala diferente da parte de cor (r, g, b), isso pode causar um viés no vetor meanshift. Portanto, iremos separar o kernel em duas partes:

$$K_{h_s,h_r}(x) = \frac{C}{h_s^2 h_r^p} k\left(\left\| \frac{x^s}{h_s} \right\|^2 \right) k\left(\left\| \frac{x^r}{h_r} \right\|^2 \right)$$

O que muda é a forma de calcular os pesos da média ponderada para encontrar o vetor *mean shift*, onde o peso total será a multiplicação dos dois pesos acima. Em oturas palavras, calcule o peso gaussiano para a parte espacial (distância em 2D), depois para a parte de cores (distância em 3D), e multiplique os dois pesos (lembre-se, queremos g, e não K).

Lembrando que h_s e h_r são parâmetros de entrada para seu algoritmo. No artigo original você pode ver alguns resultados para valores diferentes de h_s e h_r.

3.1.2 Reduzindo a complexidade

O algoritmo produz resultados, mas você vai esperar um bom tempo para ele convergir, já que tem complexidade $O(Tn^2)$, onde T é o número de iterações, e n o número de pixels. O problema é que estamos comparando cada kernel com todas amostras (número de pixels). Para reduzir esse tempo, podemos calcular a média apenas com os vizinhos mais próximos. Afinal, como está escrito no início do documento, queremos uma aproximação local da densidade.

PERGUNTA: como encontrar vizinhos mais próximos em cinco dimensões?

RESPOSTA: use uma biblioteca que faça isso, ou se preferir implemente o seu algoritmo de estrutura espacial em n-dimensões (ex. kd-tree), mas cuidado com o prazo!

Para C++ existe uma biblioteca muito boa: https://www.cs.umd.edu/~mount/ANN/
Existem similares para Python.

Agora você pode procurar apenas os k vizinhos mais próximos, ou, se preferir, pode procurar todos os vizinhos dentro de um raio r. Aconselho a primeira opção.

3.1.3 ESPAÇO DE COR

O espaço RGB não é o mais apropriado para calcular distâncias, não possui uma relação óbvia, nem linear, entre cores e suas distâncias euclideanas. Solução: converta para um espaço mais apropriado, como Luv ou Lab.

to be completed ...

3.2 MEANSHIFT SEGMENTATION

to be completed ...