

Trabalho Final

Computação Científica e Análise de Dados

K-means para minimização de cores sem e com PCA

EDUARDO GABRIEL TEIXEIRA BARROS
122060506

Professor:
JOÃO ANTÔNIO RECIO DA PAIXÃO

Dezembro, 2025.
Rio de Janeiro, RJ.

Problema do projeto:

Iremos simplificar a quantidade de cores distintas de uma imagem qualquer com o algoritmo *K-means*, de modo a diminuir a complexidade da imagem, diminuindo o mínimo da aparência original. Faremos isso por duas abordagens:

- Sem *PCA*: aplicando o *K-means* diretamente na imagem
- Com *PCA*: fazendo uma aproximação de melhor posto e depois aplicando *K-means*

Deste modo, poderemos mensurar o quanto a eficiência do algoritmo aumenta em detrimento da qualidade da imagem gerada ao utilizar *PCA* antes do *K-means*.

Entradas:

- Imagem *.jpg*, armazenada em matriz mXn de pixels, onde cada ponto tem como coordenadas seus valores RGB (*red*, *green* e *blue*);
- Número k de clusters.

Saídas:

Em ambos os casos, sem e com *PCA*:

- imagem de mesmo tamanho que a original, mas com no máximo k cores distintas;
- a porcentagem de semelhança com a imagem original;
- o tempo de execução do algoritmo *K-means*.

O Algoritmo *K-means*

O *k-means* é um método para dividir amostras em torno de centróides, de modo que tenhamos grupos de pontos semelhantes e seus devidos representantes (média dos pontos).

Neste projeto, aplicamos ele para encontrar quais cores são mais próximas na imagem, e as dividimos em k grupos.

A implementação dele é feita pela função *kmeans()* da biblioteca *Clustering*.

O *PCA*

O *PCA* (*Principal Component Analysis*) é um procedimento para reduzir o número de dimensões de grandes conjuntos de dados aos componentes principais que retém a maior parte das informações originais.

Neste projeto, aplicamos ele nas imagens como um pré-processamento do *K-means*, diminuindo suas dimensões de modo a facilitar a execução do algoritmo.

A implementação dele foi feita de forma iterativa, buscando os componentes principais enquanto suas contribuições para a aproximação da imagem sejam relevantes.

Pré-processamento

O código carrega a imagem com a função *load* da biblioteca *Images* do Julia, e transforma em reais seus números.

```
img = load("entrada/$(nome_entrada)")  
img = float64.(img) # K-means trabalha com números reais
```

Mas a imagem ainda se trata de uma matriz mXn de pontos (pixels) no R³. Então a transformamos numa matriz onde as linhas são as cores e as colunas são os pixels:

```
img = load("entrada/$(nome_entrada)")  
img = float32.(img) # K-means trabalha com números reais  
  
alt, lar = size(img)  
channels = channelview(img)  
  
img_t_original = reshape(channels, 3, alt * lar)
```

PCA no pré-processamento

Primeiro, encontramos qual é o número de componentes principais relevantes para a aproximação. Fazemos assim: calculamos os autovalores da matriz de covariância da matriz imagem, do maior para o menor, e vemos quais deles são numericamente significativos em relação ao restante. Se tenho 10 autovalores, por exemplo, mas só 5 tem valores significativos, uso 5 componentes principais na minha aproximação.

```
function k_otimo(img, limiar)  
    λ, V = eigen(img*img')  
    λ = sort(λ, rev=true)  
  
    variancia_total = sum(λ)  
  
    k_otimo = 0  
  
    for i in 1:length(λ)  
        significancia = (λ[i] / variancia_total)  
  
        if significancia < limiar  
            k_otimo = i - 1  
            break  
        end  
        k_otimo = i  
    end  
    return k_otimo  
end
```

Isso impede o código de continuar rodando o PCA para achar mais componentes sem ter um ganho real na aproximação.

Então, tendo o número de componentes ótimo, aplicamos *PCA* na matriz coresXpixels para melhorar a execução do *K-means*. Primeiro, calculamos a média da amostra e subtraímos de cada pixel, de modo que a origem do sistema seja o centro de massa dos dados. Então aplicamos o *PCA*.

Após a execução, adicionamos a média de volta aos pixels e a matriz está pronta para o *K-means*.

Execução e Pós-processamento

Usamos agora a função *kmeans()* da biblioteca *Clustering*, cujo resultado é um objeto R. Dele acessamos a qual *cluster* cada pixel pertence e quais são os centróides, e usamos essas informações para substituir cada pixel pelo centróide do *cluster* a que ele pertence.

Depois disso, transformamos de volta a matriz numérica em uma imagem e a aproximação está pronta.

```
@time R = kmeans(img_t, k; maxiter=200, display=:none)

# Substitui cada pixel pela cor do centróide do cluster
img_clust = zeros(Float32, 3, alt * lar)
for i in 1:(alt*lar)
    cluster_idx = R.assignments[i]
    img_clust[:, i] = R.centers[:, cluster_idx]
end

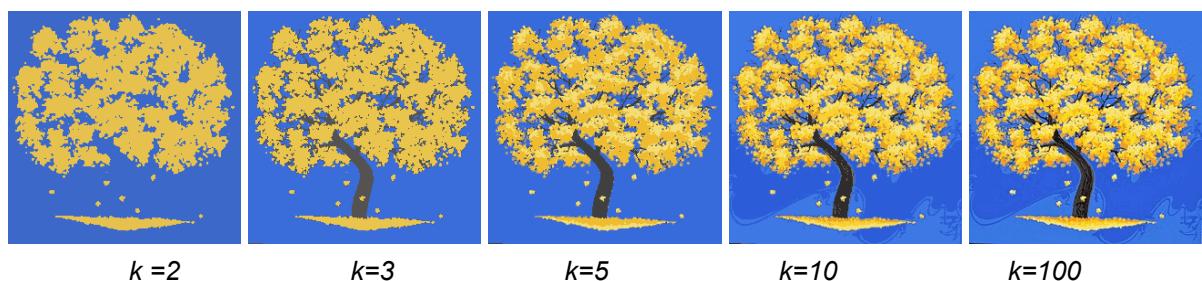
# Reconstrói a imagem
img_k = colorview(RGB, reshape(img_clust, 3, alt, lar))
return img_k, norm(img_clust)/norm(img_t_original)
```

Resultados obtidos

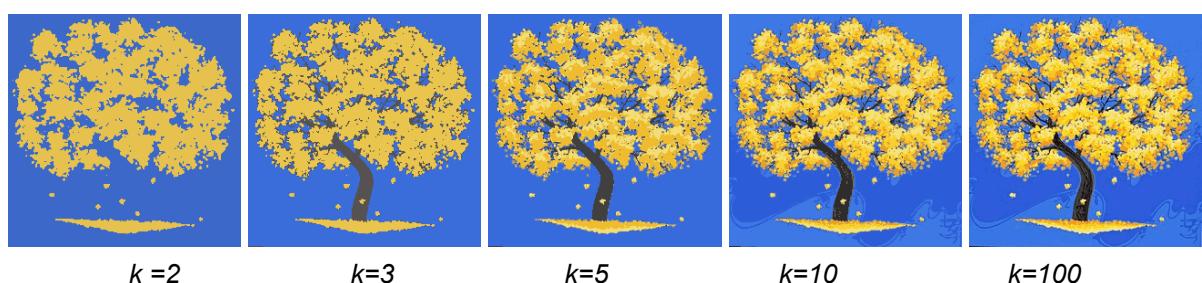
Foram feitos testes com 3 tamanhos de imagens: 256x256, 500x500 e 1200x1600. Para cada imagem, geramos aproximações com 2, 3, 5, 10 e 100 cores, com e sem PCA. Abaixo, deixo as imagens originais e suas devidas aproximações.



Sem PCA



Com PCA



Sem PCA



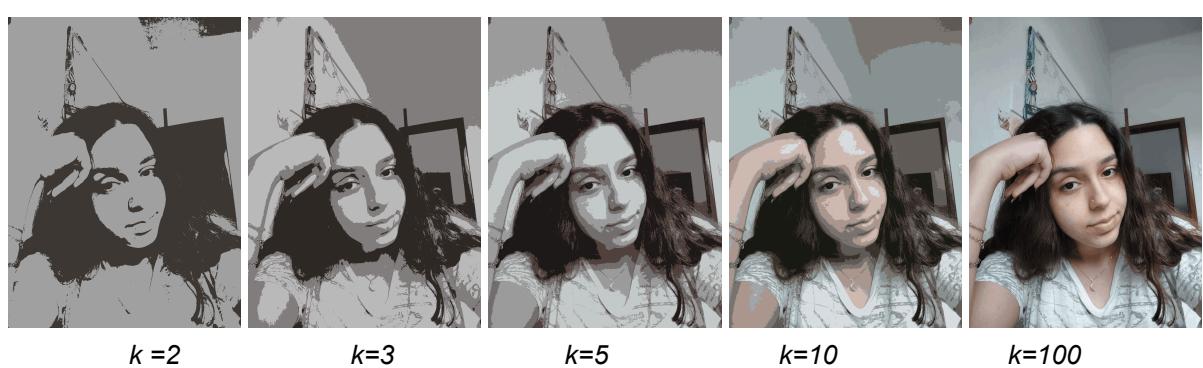
Com PCA



Sem PCA



Com PCA



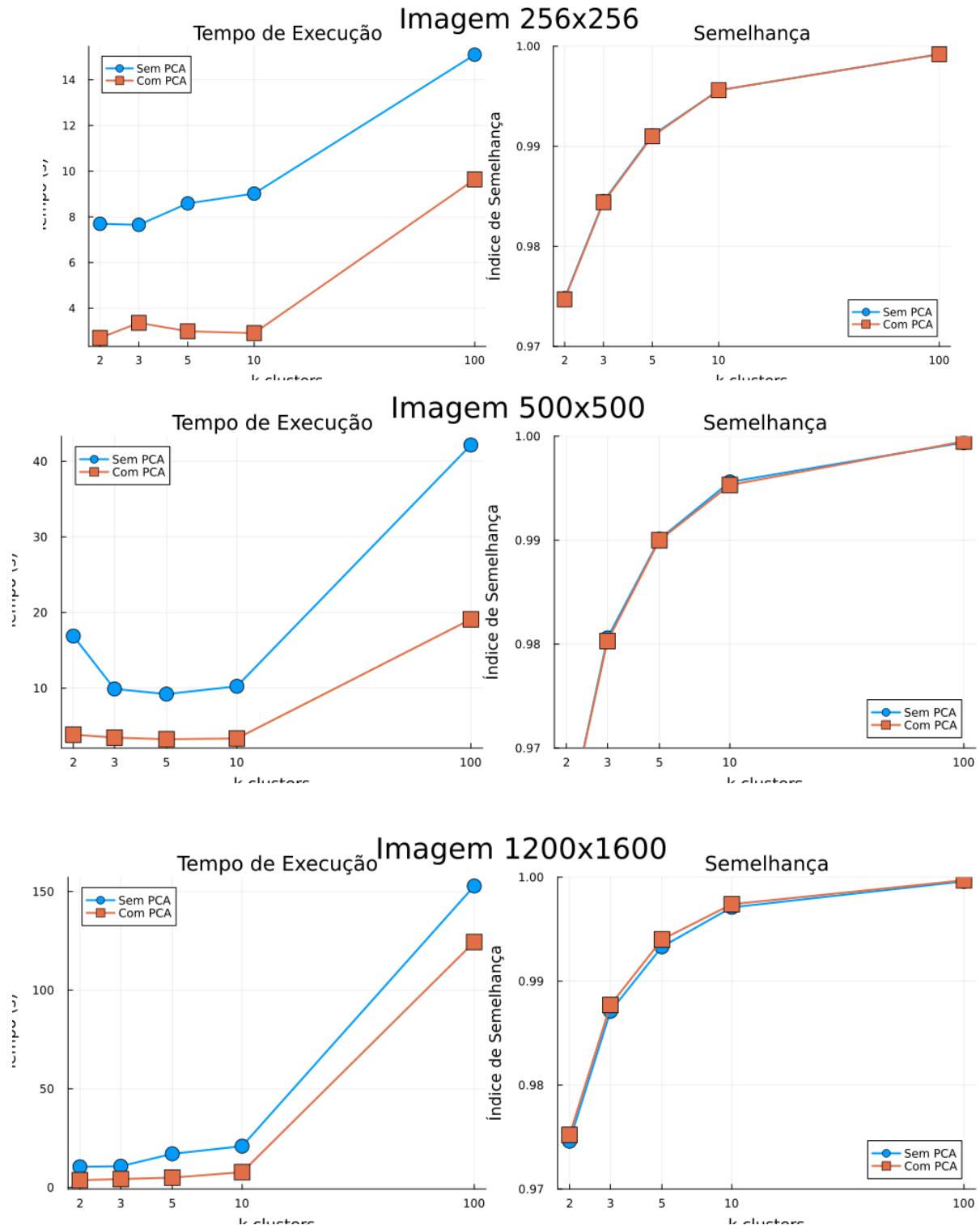
Métricas

Foram calculados então duas métricas para cada execução: o tempo de execução e a semelhança com a imagem original.

Tempo de execução: tempo em segundos calculado somente em relação à execução da função `kmeans()`. Obs.: o tempo de execução do programa como um todo geralmente aumenta quando aplicado PCA.

Semelhança: valor percentual entre 0 e 1, calculado como a razão da matriz numérica após a substituição por centróides pela matriz numérica cores X pixels.

Abaixo encontramos os gráficos comparando as duas métricas sem e com PCA no pré-processamento para cada tamanho de imagem:



Conclusões

Pela análise dos gráficos e das imagens geradas, parece-nos razoável afirmar que a *Análise de Componentes Principais* como pré-processamento oferece vantagens computacionais expressivas sem comprometer a qualidade visual de forma significativa. Enquanto a similaridade entre a imagem original e a clusterizada, com e sem PCA, são praticamente equivalentes, o tempo de processamento diminui de forma considerável para todos os tamanhos de imagem, embora de forma mais significativa nas imagens pequenas.

Isto reforça o papel fundamental que a redução de dimensionalidade tem para viabilizar soluções computacionalmente intensivas, pois contorna as barreiras instituídas pelo poder computacional.

Observações

1. O PCA utilizado foi feito de forma artesanal, sem o uso de bibliotecas que implementam esse método. O uso delas provavelmente traria resultados ainda mais eficientes e com maior estabilidade numérica.
2. Variar ainda mais a quantidade de clusters ou o tamanho das imagens nos deixaria bem mais confortáveis em considerar a eficiência da análise dos componentes no problema de cores mínimas.
3. Foi escolhido fixar o limiar de significância dos componentes principais, de modo que usássemos sempre a aproximação ótima do PCA para a imagem. Entretanto, assim como o número de clusters e o tamanho da imagem, o posto da aproximação poderia ser variável.

Links

Implementação do K-means: [Trabalho Final - CoCADA](#)

Gráficos gerados:  [Trabalho Final.jl](#)