FACULDADE DE TECNOLOGIA DE SÃO JOSÉ DOS CAMPOS

FATEC PROFESSOR Jessen Vidal

ANDRE LARS DA CUNHA

APLICAÇÃO DE TÉCNICAS DE APRENDIZAGEM SUPERVISIONADA E NÃO SUPERVISIONADA AO PROCESSAMENTO DE IMAGENS

Orientador: Dr. Fabricio Galende Marques de Carvalho

São José dos Campos

2022

SUMÁRIO

[1 Introdução 3](#__RefHeading___Toc851_3197422305)

[1.1 Definição do problema 3](#__RefHeading___Toc853_3197422305)

[1.2 Objetivo 3](#__RefHeading___Toc855_3197422305)

[2 Desenvolvimento 4](#__RefHeading___Toc857_3197422305)

[2.1 Arquitetura 4](#__RefHeading___Toc859_3197422305)

[2.2 Modelo de Dados 4](#__RefHeading___Toc861_3197422305)

[2.3 Detalhes 5](#__RefHeading___Toc863_3197422305)

[3 Resultados e Discussão 10](#__RefHeading___Toc865_3197422305)

# Introdução

Este Capítulo apresenta o problema a ser resolvido e o objetivo deste trabalho.

## Definição do problema

O uso de sistemas computacionais que substituem de modo satisfatório um agente humano na execução de certas tarefas está cada vez mais comum em vários setores de atividades. Dentre os sistemas computacionais incluem, o reconhecimento de imagens, a contagem de elementos em uma imagem, a classificação do uso de áreas urbanas ou rurais, a identificação de patologias em pacientes, etc.

Pode-se também usar do poder do computador em uma imagem digital para identificação de uma construção ou vegetação. Essa habilidade é útil para medição de cobertura vegetal em áreas urbanas, identificação de índices de vegetação para monitoramento do desmatamento e ação humana, ou seja, classificar, reconhecer ou identificar esses elementos em uma imagem é de interesse público e privado para resolver problemas que enfrentamos pela ação humana e a urbanização.

## Objetivo

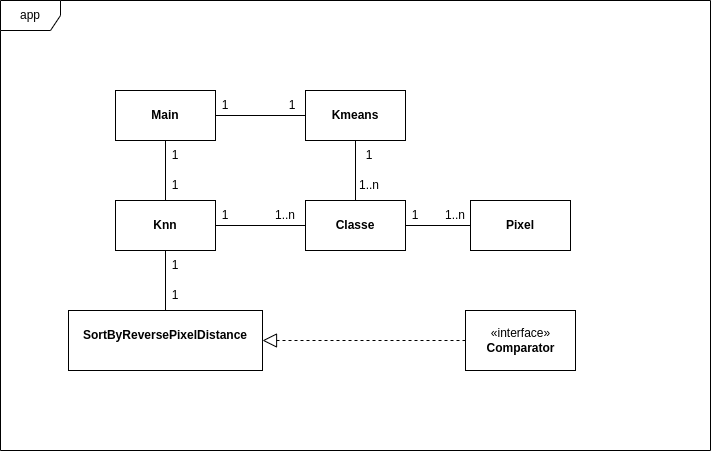
O objetivo do trabalho consiste em desenvolver componentes de software para o processamento de imagens, utilizar técnicas de inteligência artificial, aprendizagem supervisionada e não-supervisionada, para identificação de vegetação ou construção nessas imagens. O trabalho atende aos requisitos da disciplina Lab. de Desenvolvimento em BD VI.

# Desenvolvimento

## Arquitetura

A Figura 1 representa o diagrama de classes da solução para o problema proposto. As classes foram desenvolvidas usando a linguagem de programação Java.

Figura 1



## Modelo de Dados

Os componentes de software desenvolvidos operam sobre imagens raster, aceitando como entrada arquivos com extensão *jpeg*, *png* e *jpg*. Uma vez feita a leitura dos dados de entrada, os dados serão processados a nível de píxel, em ArrayLists, HashSets, HashMaps e PriorityQueues. Já a saída são arquivos com extensão *png*.

Em uma aplicação real poderiamos persistir os dados categorizados das imagens de referência para a aprendizagem supervisionada, as imagens de saída e seus respectivos centros também podem ser persistidos em um banco de dados.

Cabe ressaltar que o problema tratado nesse trabalho de graduação é caracterizado como um filtro não paramétrico ao nível de píxeis. Em outros sistemas de classificação, onde um determinado vetor de características é extraído a partir da imagem, não é necessário o armazenamento das imagens de referência, mas sim somente os vetores de características e suas respectivas classes.

## Detalhes

2.3.1. **Tecnologias utilizadas**

* Linguagem de programação Java: Java foi a tecnologia escolhida por ser multi-plataforma com a JVM, a linguagem é compilada, ou seja, a performance pode ser melhor que uma linguagem interpretada, mas principalmente pela experiência prévia com a tecnologia.
* Gradle: Responsável pelo gerenciamento de dependências do projeto e execução.

2.3.2. **Descrição do Algoritmo KNN (K-Nearest Neighbors)**

KNN ou K-vizinhos mais próximos, trata-se de um algoritmo de classificação que atribui uma classe, a um dado ainda não classificado, dependendo de quão próximo esse item está de um certo número de itens cujas classes já são conhecidas.

Dado um certo conjunto de N itens já classificados, o algoritmo calcula a distância do item não classificado a cada um desses itens. Considerando os K itens mais próximos (vizinhos), a classe que tiver a maior ocorrência corresponderá à classe do item desconhecido, no caso de um empate de frequência das classes, o algoritmo não consegue realizar uma escolha, esse cenário só pode acontecer se K for um número par, portanto é esperado que o parâmetro usado seja somente um número ímpar.

Os parâmetros de configuração do algoritmo são o número K, um numero inteiro positivo e ímpar, e também o conjunto de N itens já classificados, ou seja, as imagens raster de referência para o algoritmo (supervisão).

Como entrada do sistema é esperado os itens a serem classificados, portanto, a imagem raster a ser classificada. Já os dados de saída são os píxeis da entrada classificados, que formará uma imagem com os píxeis classificados.

A Figura 2 representa o código da seleção dos vizinhos mais próximos.

Figura 2

A linha 1 corresponde a assinatura do método privado getKVizinhosMaisProximos(), que recebe como parâmetros o conjunto de dados já classificados como referência, o píxel a ser classificado e o valor inteiro de K, e o método retorna uma fila de prioridade com os K vizinhos mais próximos.

1 **private** PriorityQueue<Pixel> getKVizinhosMaisProximos(Set<Pixel> dadosRef, Pixel pixel, int k) {

2 **var** vizinhosMaisProximos = **new** PriorityQueue<>(k, SORT\_BY\_REVERSE\_PIXEL\_DISTANCE);

3 **for** (Pixel dadoRef : dadosRef) {

4 dadoRef.setDistance(pixel);

5 vizinhosMaisProximos.offer(dadoRef);

6 **if** (vizinhosMaisProximos.size() > k) {

7 vizinhosMaisProximos.poll();

8 }

9 }

10 **return** vizinhosMaisProximos;

11 }

Na linha 2 é instanciado uma nova PriorityQueue chamada vizinhosMaisProximos, ela inicia com o tamanho k e um Comparator chamado SORT\_BY\_REVERSE\_PIXEL\_DISTANCE, que ordena a fila de acordo com a distância em ordem descendente, ou seja, o primeiro da fila é o mais distante.

A linha 3 começa com um laço *for* iterando sobre os dados em dadosRef, para cada dado de referência é calculada a distância dele com o píxel não classificado e adicionado esse dado à fila de prioridade. Se a fila tem o tamanho maior que K, retiramos o primeiro da fila, portanto o dado mais distante ao píxel que estamos querendo classificar sai da fila. Esse processo é repetido para todos os dados de referência, assim no final da iteração retornamos a fila somente com os K vizinhos mais próximos do píxel não classificado.

A Figura 3 representa o código da eleição da classe mais frequente entre os vizinhos mais próximos.

O método getClasseMaisFrequente() recebe a fila com os K vizinhos mais próximos do código anterior e calcula qual das classes é a mais frequente entre os vizinhos. Primeiro na linha 2 é instanciado um HashMap que tem como chave uma Classe e como valor um Integer que representa a quantidade que aquela classe aparece.

Figura 3

1 **private** Classe getClasseMaisFrequente(PriorityQueue<Pixel> fila) {

2 **var** mapClasseQtd = **new** HashMap<Classe, Integer>();

3 int qtdMaisFrequente = 0;

4 Classe classeMaisFrequente = **null**;

5 **while** (Objects.nonNull(fila.peek())) {

6 **var** classe = fila.poll().getClasse();

7 **var** qtdAtualOuZero = mapClasseQtd.getOrDefault(classe, 0);

8 mapClasseQtd.put(classe, qtdAtualOuZero + 1);

9 **var** qtd = mapClasseQtd.get(classe);

10 **if** (qtd > qtdMaisFrequente) {

11 qtdMaisFrequente = qtd;

12 classeMaisFrequente = classe;

13 }

14 }

15 **return** classeMaisFrequente;

16 }

Na linha 5, inicia um *loop while* que diz que enquanto tiver elementos na fila, prosseguimos com as instruções: retira-se o primeiro da fila e assinala-se a classe à variável classe, depois consultamos o mapa usando como chave a classe anterior, se a chave existir, retorna o valor atual, se não existir, retorna 0 (zero). Incrementa-se em um o valor da chave anterior, se o valor incrementado é maior que o valor da classe mais frequente até então, substituimos a classe mais frequente pela classe atual. No fim, teremos qual classe é a mais frequente nos K vizinhos mais próximos, portanto, classificando o píxel não classificado.

2.3.3. **Descrição do Algoritmo K-Means**

K-Means, trata-se de um algoritmo de agrupamento que, ao final de sua execução, pode ser utilizado, também, para classificação de acordo com as classes por esse definidas. Esse algoritmo agrupa um conjunto de N itens em K classes, de acordo com a proximidade de cada um dos itens do centro dessas classes.

Os centros são calculados considerando os vetores que representam as características dos itens a serem classificados. No caso estudado essas características são os valores do RGB do píxel, ou seja, um *array* tridimensional. A proximidade de um item de uma determinada classe (centro), será dada por uma medida de distância do vetor de características do item em relação ao centro, usaremos aqui a distância euclidiana.

O único parâmetro desse algoritimo é K, que determina o número de classes. Como entrada do sistema é esperado o valor inteiro K e uma imagem raster, que contém os itens N não classificados.

A saída esperada do programa é a imagem raster com cada um dos píxeis classificados de acordo com as classes identificadas pelo K-Means e o valor dos centros dessas classes.

A Figura 4 representa o código do algoritmo geral do K-Means.

Figura 4

O método calcularKmeans recebe como argumentos o caminho do arquivo a ser lido e o valor inteiro K com o número de classes a serem identificadas. Primeiro, na linha 2 é feito a leitura da imagem raster da entrada de acordo com o caminho. Segundo, na linha 3 é chamado o método que inicializa as K classes em posições aleatórias. Terceiro, na linha 4 inicializa uma lista de classes para comparação, essa comparação é feita a cada iteração do *loop while* verificando a condição de parada do laço de repetição.

1 **public** void calcularKmeans(String path, int k) **throws** IOException {

2 **var** img = readPathImg(path);

3 inicializarClassesAleatorios(k);

4 List<Classe> classesIteracaoAnterior = inicializarClassesParaComparacao(k);

5 **while** (centrosNaoConvergiram(classesIteracaoAnterior, k)) {

6 classes.forEach(Classe::limparLista);

7 **for** (Pixel pixel : dados) {

8 definirClasseDeUmPixel(pixel);

9 }

10 atualizarCentros();

11 }

12 printClasses();

13 gerarImagem(img, path, k);

14 }

Na linha 5 se inicia o laço *while* com a condição de enquanto os centros não convergirem, ou seja, enquanto os centros da iteração anterior não forem iguais aos centros dessa iteração, continue.

O primeiro passo a ser feito em toda iteração é limpar a lista de píxeis associados de todos os centros na linha 6, com a lista limpa inicializa um novo laço *for* iterando sobre todos os píxeis a serem classificados e para cada um é definido um centro de acordo com a distância dele para com todos os centros atuais.

Ao terminar o laço, na linha 10 chama-se o método atualizarCentros() que primeiro irá atualizar os valores dos centros de acordo com os píxeis associados a eles, depois irá escrever no std::out os valores atualizados da posição dos centros.

Quando atingir a condição de parada, terminamos o laço while e geramos a imagem de saída com todos os píxeis classificados em K classes. Pode haver mais de uma condição de parada, a que foi adotada é esperar os centros convergirem sem variação entre duas iterações consecutivas, outra abordagem poderia ser pouca variação entre duas iterações, ou até um número limite de iterações sem analisar a convergência.

A Figura 5 representa o código do método centrosNaoConvergiram.

Figura 5

O método recebe a lista de classes da iteração anterior e o valor de k. Na linha 2 é inicializado a variável booleana naoConvergiram que recebe o valor da verificação se as classes da iteração anterior é igual as classes atuais. Uma vez realizada essa operação, o valor dos centros da lista classesIteracaoAnterior é atualizado com o valor dos centros atuais, e finalmente o método retorna o booleano naoConvergiram.

1 **private** boolean centrosNaoConvergiram(List<Classe> classesIteracaoAnterior, int k) {

2 boolean naoConvergiram = !classesIteracaoAnterior.equals(classes);

3 **for** (int i = 0; i < k; i++) {

4 classesIteracaoAnterior.get(i).setCentro(classes.get(i).getCentro());

5 }

6 **return** naoConvergiram;

7 }

A Figura 6 representa o código do método definirClasseDeUmPixel.

Figura 6

O método recebe um píxel e inicializa duas variáveis, uma representando a menor distância e a segunda a classe mais próxima nas linhas 2 e 3 respectivamente. Na linha 4 inicia o laço de repetição for que para cada classe é calculada a distância euclidiana do píxel e o centro da classe.

1 **private** void definirClasseDeUmPixel(Pixel pixel) {

2 **var** menorDist = Double.MAX\_VALUE;

3 Classe maisProx = **null**;

4 **for** (Classe classe : classes) {

5 **var** dist = pixel.distanciaEuclidiana(classe.getCentro());

6 **if** (dist < menorDist) {

7 menorDist = dist;

8 maisProx = classe;

9 }

10 }

11 **if** (maisProx != **null**) {

12 maisProx.addPixel(pixel);

13 }

14 }

Uma vez que essa distância é menor que o valor na variável menorDist, substituímos o menorDist pela distância calculada e apontamos a classe maisProx para a classe em questão. No final do *loop* se maisProx não for null, associamos o pixel a classe maisProx com o método addPixel.

A Figura 7 representa o código do método atualizarCentro.

Figura 7

Na linha 2 verificamos se a lista de píxeis associados a um centro não está vazia. Se não estiver, é calculado o novo centro usando a média aritmética dos valores RGB dos píxeis associados, uma vez calculado o novo centro é definido.

1 **public** void atualizarCentro() {

2 **if** (!pixelsAssociados.isEmpty()) {

3 int somR = 0;

4 int somG = 0;

5 int somB = 0;

6 **for** (Pixel pixel : pixelsAssociados) {

7 somR += pixel.getR();

8 somG += pixel.getG();

9 somB += pixel.getB();

10 }

11 int size = pixelsAssociados.size();

12 setCentro(**new** Pixel(somR / size, somG / size, somB / size));

13 }

14 }

# Resultados e Discussão

3.1. Aplicação ao problema de computação de área asfaltada – Caso 1

3.1.1 K-Means

* Imagem de entrada: Figura 8

Figura 8

* Valor de K no K-Means: 2
* Valores dos centros na saída do K-Means:

Classe 0: Classe{centro=Rgb{r=149, g=167, b=142}}

Classe 1: Classe{centro=Rgb{r=88, g=118, b=76}}

* Imagem de saída do K-Means: Figura 9

Figura 9



3.1.2 KNN

* Imagens de referência para a supervisão do KNN:

Figura 10 Figura 11



* A figura 12 é a imagem de saída do KNN para K = 7.

Figura 12



O algoritmo KNN produz uma imagem de saída mais parecida com o resultado esperado, conseguimos ver o contorno definido da área asfaltada. Em contrapartida, o K-Means agrupa a imagem em duas classes porém algumas áreas não asfaltadas são da mesma classe que o asfalto, trazendo um resultado não satisfatório para o objetivo.

Se alterarmos o número de centros no algoritmo K-Means, teremos uma imagem de saída com K cores, podemos aumentar esse valor de K até que chegue um momento em que a imagem terá uma quantidade de cores muito próxima a imagem original. Veja exemplo na figura 13, imagem gerada com K = 30.

Figura 13



Podemos ver que com K centros correspondem a um classificador com K classes. Já para o algoritmo KNN a acurácia depende de existir ou não exemplos que sejam capaz de representar ou de se aproximar do píxel a ser classificado. Por exemplo, se alterarmos o valor de K para 3 teremos a figura 14:

Figura 14



Agora a figura 15 é uma imagem com o parâmetro K = 101:

Figura 15



Porém, como podemos ver, mesmo alterando o número K de vizinhos próximos, as diferentes execuções pro programa não traz resultados com diferenças significativa, pois os dados de referência pré-classificados são os mesmos.

3.3. Conclusões e considerações finais

Pode-se concluir que o algoritmo supervisionado gerou um resultado mais satisfatório para o objetivo. O KNN conseguiu classificar os píxeis com maior acurácia que o algoritmo sem supervisão.

Como sugestão para a continuar, sugiro alterar as características das imagens a serem processadas nos algoritmos além das cores RGB, assim poderíamos analisar outros aspectos de comparação nos resultados obtidos pelos algoritmos supervisionados e não-supervisionados.