**PSI3471 - Fundamentos de Sistemas Eletrônicos Inteligentes**

Lição de Casa 4 (Aulas 7 e 8)

| **Nome Completo** | **Número USP** |
| --- | --- |
| Gustavo Henrique da Silva Amaral | 12551686 |
| Thiago da Rocha Calomino Gonçalves | 12554647 |

**Lição de casa #4 (Parte 1)**

Resolva o problema *noisynote* usando os algoritmos de vizinho mais próximo e a árvore de decisão.

Exercício: Em aprendizado de máquina, é usual normalizar os dados de entrada, isto é, subtrair a média e dividir por desvio padrão. Verifique as taxas de acerto de vizinho mais próximo e a árvore de decisão normalizando os dados de entrada.

**Explicação completa:**

-

**Código da lição:**

| // Lição de casa 4 (Parte 1)  //  // Gustavo Henrique da Silva Amaral - 12551686  // Thiago da Rocha Calomino Gonçalves - 12554647  // Em C++ usando algoritmo de vizinho mais próximo (OpenCV)  #include <cekeikon.h>  int main() {  // Leitura dos arquivos de texto  Mat\_<float> ax; le(ax, "ax.txt");  Mat\_<float> ay; le(ay, "ay.txt");  Mat\_<float> qx; le(qx, "qx.txt");  Mat\_<float> qy; le(qy, "qy.txt");  Mat\_<float> qp;  // Algoritmo KNN do OpenCV  CvKNearest ind(ax, ay, Mat(), false, 1);  ind.find\_nearest(qx, 1, &qp);  // Cálculo da acurácia do algoritmo  int erros = 0;  for (int i = 0; i < qp.rows; i++)  if (qp(i) != qy(i)) erros++;  // double taxaErros = erros/qp.rows;  printf("Erros = %d / %d ou %1.3f%%\n", erros, qp.rows, 100.0 \* erros/qp.rows);  }  // Em Python usando árvore de decisão  import numpy as np  from sklearn import tree  import matplotlib.pyplot as plt  def le(nomearq):  with open(nomearq,"r") as f:  linhas = f.readlines()  linha0 = linhas[0].split()  nl = int(linha0[0]); nc = int(linha0[1])  a = np.empty((nl,nc),dtype = np.float32)  for l in range(nl):  linha = linhas[l+1].split()  for c in range(nc):  a[l,c] = np.float32(linha[c])  return a  ### main  ax = le("/content/ax.txt")  ay = le("/content/ay.txt")  qx = le("/content/qx.txt")  qy = le("/content/qy.txt")  arvore = tree.DecisionTreeClassifier()  arvore = arvore.fit(ax, ay)  qp = arvore.predict(qx)  erros = 0  for i in range(qp.shape[0]):  if (qp[i] != qy[i]):  erros += 1  print(f"Erros= {erros} / {qp.shape[0]}\n Pct = {100.0 \* erros/qp.shape[0]}") |
| --- |

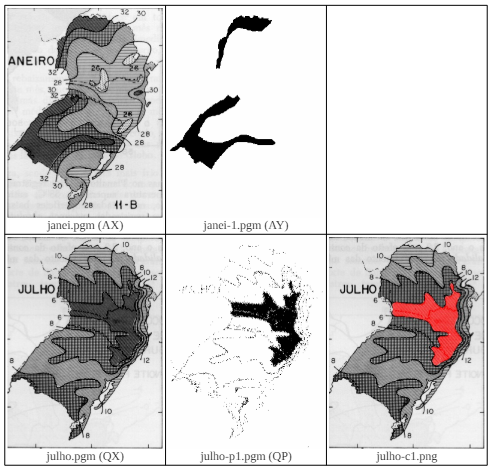
**Lição de casa #4 (Parte 2)**

Faça um programa que efetua a sequência das seguintes operações:

1. Recebe as imagens janei.pgm e janei-1.pgm como amostras de treinamento (AX, AY) e cria um filtro por aprendizado de máquina.
2. Aplica o filtro aprendido na imagem julho.pgm (QX) gerando uma imagem semelhante a julhop1.pgm (QP).
3. Filtra essa imagem com filtro mediano adequado.
4. Sobrepõe a imagem filtrada à imagem original, obtendo uma imagem semelhante à julho-c1.png.

Dica 1: Usei filtro 3×3 e usei FlaNN (você pode usar outras técnicas).

Dica 2: Para sobrepor máscara vermelha, deixei componente R da imagem de saída 255.



**Explicação:**

-

**Código da lição:**

| // Lição de casa 4 (Parte 2)  //  // Gustavo Henrique da Silva Amaral - 12551686  // Thiago da Rocha Calomino Gonçalves - 12554647  // Segmentacao de area de Mapa com KDTree e Vizinho Mais Próximo  // Usa OpenCV2  #include <cekeikon.h>  int main() {  Mat\_<GRY> ax; le(ax,"janei.pgm");  Mat\_<GRY> ay; le(ay,"janei-1.pgm");  Mat\_<GRY> qx; le(qx,"julho.pgm");  if (ax.size()!=ay.size()) erro("Erro dimensao");  Mat\_<GRY> qp(qx.rows,qx.cols);  //Cria as estruturas de dados para alimentar OpenCV  Mat\_<FLT> features(ax.rows\*ax.cols,9);  Mat\_<FLT> saidas(ax.rows\*ax.cols,1);  int i=0;  for (int l=0; l<ax.rows; l++)  for (int c=0; c<ax.cols; c++) {  // salva o pixel das entradas  features(i, 0)=ax(l,c);  features(i, 1)=ax(l+1,c);  features(i, 2)=ax(l-1,c);  features(i, 3)=ax(l,c+1);  features(i, 4)=ax(l,c-1);  features(i, 5)=ax(l-1,c-1);  features(i, 6)=ax(l-1,c+1);  features(i, 7)=ax(l+1,c+1);  features(i, 8)=ax(l+1,c-1);    // salva o respectivo pixel da saída  saidas(i)=ay(l,c);  i++;  }  // Aqui, as 4 arvores estao criadas  flann::Index ind(features,flann::KDTreeIndexParams(4));  Mat\_<FLT> query(1,9);  vector<int> indices(1);  vector<FLT> dists(1);  medianBlur(qx, qx, 1);  for (int l=0; l<qp.rows; l++)  for (int c=0; c<qp.cols; c++) {    query(0)=qx(l,c);  query(1)=qx(l+1,c);  query(2)=qx(l-1,c);  query(3)=qx(l,c+1);  query(4)=qx(l,c-1);  query(5)=qx(l-1,c-1);  query(6)=qx(l-1,c+1);  query(7)=qx(l+1,c+1);  query(8)=qx(l+1,c-1);    // Zero indica sem backtracking  ind.knnSearch(query, indices, dists, 1, flann::SearchParams(0));  // encontra a saida mais proxima baseada na entrada  // Saida é um numero entre 0 e 255  qp(l,c) = saidas(indices[0]);  }  medianBlur(qp, qp, 7);  imp(qp,"julho-p1.png");  mostra(ax); mostra(ay);  mostra(qx); mostra(qp);  // Pintando o mapa:  Mat\_<COR> mapa;  le(mapa, "julho.pgm");  for(int i=0; i< mapa.rows; i++)  for(int j=0; j< mapa.cols;j++){  if(qp[i][j] == 0.0) mapa[i][j][2] = 255; // pixel do RED  }    imp(mapa, "mapaDestacado.png");  mostra(mapa);  } |
| --- |