UNIVERSIDADE FEDERAL DE MINAS GERAIS

ELT 135 – RECONHECIMENTO DE PADRÕES

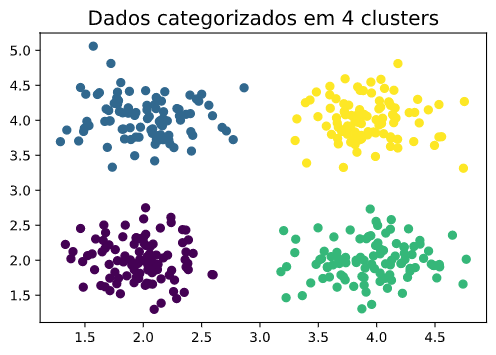
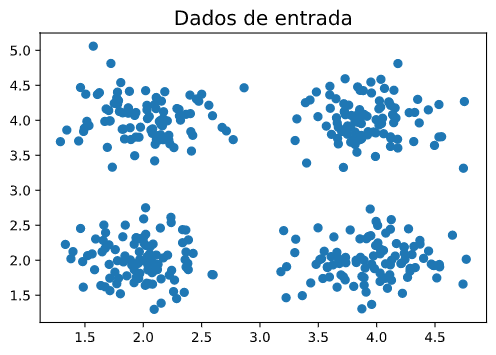
RELATÓRIO K-MEANS COM WARD

Gabriel Saraiva Espeschit – 2015065541

30 de agosto de 2020

A função *k-means* foi adaptada para incluir a inicialização e seleção *ward*. A função *kmeans\_ward* pode ser verificada no apêndice desse relatório.

Aplicando a função *kmeans\_ward* foi aplicada usada em dados de entrada gerados de forma semelhante ao primeiro exercício *K-Means*, usando uma desvio padrão de 0.3 e 4 *clusters*. Os dados de entrada e a classificação obtida pode ser vista abaixo:

A principal diferença observada utilizando o *K-Means* com *Ward* em relação a uma inicialização normal foi que a convergência dos dados ocorreu mais rapidamente. No entanto, em alguns casos, a classificação de dados não era feita da forma mais adequada.

# Apêndice

*def kmeans\_ward(input\_x, num\_clusters, tol, max\_it, num\_pontos\_rand):*

*'''*

*Função que aplica a metodologia KMeans com inicialização Ward para clusterização de dados.*

*Dados de entrada:*

*input\_x: (np.array) dados de entradas a serem clusterizados*

*num\_clusters: (int) número de grupos de clusterização*

*tol: (float) tolerância maxima do algoritimo de clusterização*

*max\_it: número máximo de iterações que o algorítimo poderá percorrer*

*num\_pontos\_rans: o número de pontos do dados de entrada que devem ser usados para calcular novo ponto central*

*Saída:*

*output\_x: (np.array) dados de entrada classificados em grupos*

*cluster\_central: os pontos do cluster central final obtido*

*'''*

*x\_min = np.amin(input\_x)*

*x\_max = np.amax(input\_x)*

*output\_x = np.zeros((input\_x.shape[0], input\_x.shape[1]+1))*

*output\_x[:,:-1] = input\_x*

*categories = range(num\_clusters)*

*means = np.zeros((num\_clusters, 2))*

*cluster\_centers = np.zeros((num\_clusters, 2))*

*random\_indices = input\_x[np.random.choice(input\_x.shape[0], num\_clusters\*num\_pontos\_rand, replace=False), :*

*for cluster in categories:*

*means[cluster] = np.mean(random\_indices[(cluster)\*(num\_pontos\_rand):(cluster+1)\*(num\_pontos\_rand), :], axis=0)*

*cluster\_centers = np.copy(means)*

*x = True*

*num\_it = 0*

*while x:*

*old\_cluster\_centers = np.copy(cluster\_centers)*

*for i in range(input\_x.shape[0]):*

*dist = []*

*for cluster\_center in cluster\_centers:*

*dist.append(np.linalg.norm(input\_x[i]-cluster\_center))*

*output\_x[i,-1] = dist.index(min(dist))*

*for category in categories:*

*in\_category = output\_x[output\_x[:,-1] == category]*

*in\_category = in\_category[:, :-1]*

*means[category] = np.mean(in\_category, axis=0)*

*cluster\_centers[category] = means[category]*

*ind\_temp = 0*

*for j, cluster\_center in enumerate(cluster\_centers):*

*dist = []*

*for i in range(input\_x.shape[0]):*

*dist.append(np.linalg.norm(input\_x[i]-cluster\_center))*

*ind\_temp = (np.argpartition(dist, num\_pontos\_rand)[:num\_pontos\_rand])*

*means[j] = np.mean(input\_x[ind\_temp], axis=0)*

*cluster\_centers[j] = means[j]*

*num\_it += 1*

*if ((old\_cluster\_centers - cluster\_centers) <= tol).all() or num\_it >= max\_it:*

*x = False*

*return(output\_x, cluster\_centers)*