UNIVERSIDADE FEDERAL DE MINAS GERAIS

ELT 135 – RECONHECIMENTO DE PADRÕES

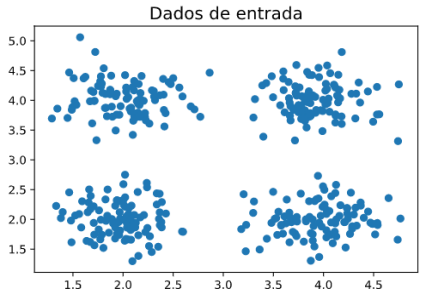
RELATÓRIO K-MEANS COM WARD

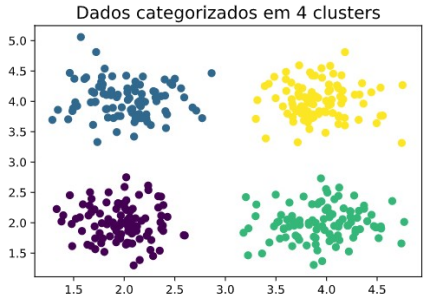
Gabriel Saraiva Espeschit – 2015065541

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 algoritimo 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)
```