Biologia Computacional 2019/2 - Lista de Exercícios 4 - Parte I

Aluno: Leonardo Oliveira Wellausen

1) Código:

```
def kmeans(samples, k):
    cs = list(choices(samples, k=k))
    prev_cs = [[np.inf for i in range(len(samples[0]))] for j in range(k)]
    gs = [[] for i in range(k)]
    while different_cs(cs, prev_cs):
        gs = [[] for i in range(k)]
        sums = [[0 for j in range(len(samples[0]) - 1)] for i in range(k)]
        counts = [0 for i in range(k)]

        for sample in samples:
            dists = [dist(sample, c) for c in cs]
            mindist = dists.index(min(dists))

            gs[mindist].append(sample)
            sums[mindist] = np.add(sums[mindist], sample[1:])
            counts[mindist] += 1

        prev_cs = deepcopy(cs)

        for i in range(k):
            cs[i][1:] = np.divide(sums[i], counts[i])

    return gs
```

A principal função do código é a função que executa k-means. A função foi escrita de forma parametrizável, então podemos separar os dados em quantos grupos quisermos apenas alterando o valor de K.

A função recebe um conjunto de amostras **samples** e um número **k** de grupos nos quais **samples** devem ser agrupadas.

Escolhemos, aleatoriamente, dois elementos de **samples** para serem os **k** centróides iniciais. Iniciamos **k** grupos vazios, e em seguida percorremos todas as amostras, calculando suas distâncias a todos centroides; escolhemos o centróide que obteve menor distância com uma dada amostra, e inserimos essa amostra no grupo desse centróide (centróides, grupos etc estão linkados através dos índices que são iguais nas diversas listas); à medida que adicionamos elementos nos grupos, também vamos adicionando os valores de todas as 7.000+ dimensões desses elementos em uma variável **sum** que existe para cada grupo, e contando os elementos de um grupo em **count,** que

também existe para cada grupo; após todos os elementos separados nos grupos, utilizamos as variáveis **sum** e **count** para calcular as médias de cada dimensão em cada grupo, e assim atualizamos os novos centróides. Iteramos este procedimento até os centróides não se alterarem, significando que nenhum elemento mudou de grupo.

```
def different_cs(cs1, cs2):
    for i in range(len(cs1)):
        for j in range(1, len(cs1[0])):
            if cs1[i][j] != cs2[i][j]:
                return True
    return False
```

Esta função é utilizada para a parada do algoritmo k-means. Retorna True se os centróides são diferentes, significando que não houve convergência ainda; e retorna False se todos os elementos em todos os centróides são iguais aos seus anteriores, o algoritmo convergiu.

```
def dist(sample, c):
    return np.linalg.norm(np.subtract(sample[1:], c[1:]))
```

Função para calcular a distância de um elemento à um centróide (genericamente, entre dois elementos). É calculada a distância euclidiana entre dois pontos em um espaço de 7.000+ dimensões na forma da norma do vetor entre os dois pontos.

```
def eval_group(group):
    labels = {'AML': 0, 'ALL': 0}
    for sample in group:
        labels[sample[0]] += 1

    return labels


def evalprint_group(group):
    labels = {'AML': 0, 'ALL': 0}
    for sample in group:
        labels[sample[0]] += 1

    print('Número de indivíduos no grupo: ', len(group), '. Sendo ', labels['ALL'], ' ALL e ', labels['AML'],  ' AML.')
```

Funções para avaliação dos grupos gerados pelo k-means. Contamos quantos indivíduos com cada label (AML e ALL) existem em um grupo e retornamos esta informação. Ou já imprimimos no terminal.

```
lines = open('leukemia_big.csv', 'r').read().splitlines()
n_samples = len(lines[0].split(','))

labels = lines.pop(0).split(',')

samples = []

for i in range(n_samples):
    this_line = [labels[i]]
    for line in lines:
        l = line.split(',')
        this_line.append(float(l[i]))
    samples.append(this_line)


leuks = {'AML': 0, 'ALL': 0}
for label in labels:
    leuks[label] += 1
```

Função main. Aqui carregamos o arquivo de entrada e o parseamos de forma que teremos uma lista contendo todas as samples, que também serão listas. Também contamos quantos indivíduos existem com cada label.

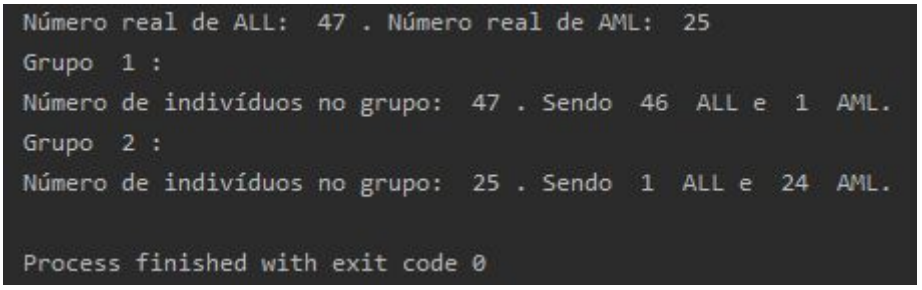
```
best_score = np.inf
best_model = None
for i in range(5):
    groups = kmeans(samples, 2)

    e0 = eval_group(groups[0])
    e1 = eval_group(groups[1])

    erro = 0

    if e0['ALL'] >= e0['AML']:
        erro += e0['AML']
        erro += e1['ALL']
    else:
        erro += e1['AML']
        erro += e0['ALL']

    if erro <= best_score:
        best_score = erro
        best_model = groups

print('Número real de ALL: ', leuks['ALL'], '. Número real de AML: ', leuks['AML'])
for i, group in enumerate(best_model):
    print('Grupo ', i+1, ':')
    evalprint_group(group)
```

Continuação da main. Geração e avaliação de resultados para 2 grupos. Como o algoritmo k-mean é estocástico, às vezes obtemos resultados piores ou melhores dependendo da escolha inicial de centróides. Para apaziguar este problema, computamos o k-mean um número n de vezes (aqui 5 pois se mostrou apropriado). Para cada k-mean rodado é calculado um erro com base base no número de indivíduos que estão no grupo errado.

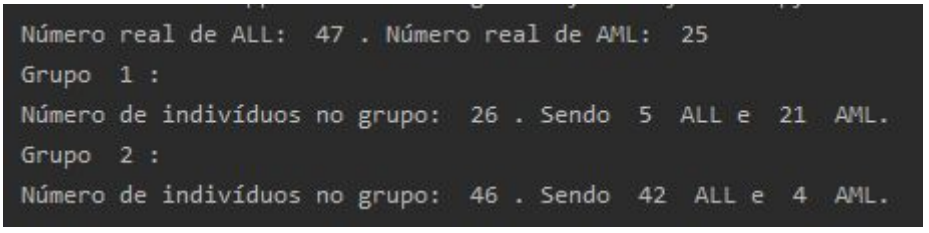
Como é calculado o erro: assumimos qual o label de um grupo com base no label predominando naquele grupo (AML ou ALL), logo o erro gerado nesse grupo é o número de indivíduos da label minoritária neste grupo. Logo o erro é a soma dos elementos que aparecem minoritariamente em cada grupo. Os **k** grupos que geram menor erros são usados como modelo final (não é calculada média ou outras métricas, apenas um modelo final é utilizado).

Resultados:

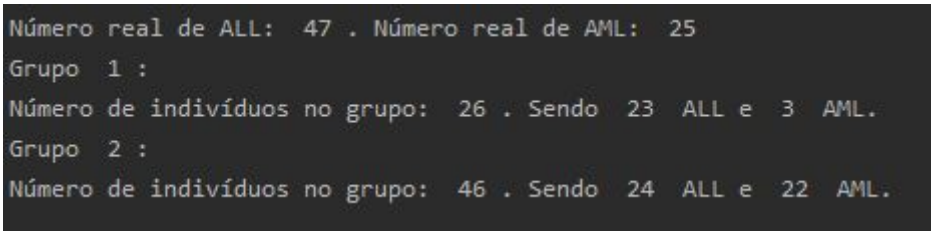
2 grupos: Para dois grupos, o algoritmo apresentou diversos resultados com bastante sentido intuitivo, separando bem os indivíduos em 2 grupos distintos, com poucos indivíduos trocados.



Resultado para 2 grupos rodando k-means 5 vezes. Nota-se uma separação excelente dos indivíduos, com apenas 1 indivíduo trocado em cada grupo.



Outro resultado bastante bom com 5 rodadas de k-means. Nota-se que aumentando o número **n** de rodadas do k-means diminuímos a possibilidade de resultados com maior troca de indivíduos. Porém, em nenhuma situação consegui uma segregação 100% correta (nenhuma troca).



Típico resultado com seleção inicial ruim de centróides (apenas uma execução de k-means aqui). Há um grupo de ALL bem conciso, porém o outro grupo tem, praticamente, metade dos elementos de cada label.

3 grupos: Os resultados para 3 grupos foram (me pareceram) bastante piores que para 2 grupos. Não nota-se, como o que seria esperado, dois grupos bem segregados (um AML e outro ALL) e um outro grupo misto. O que acontece normalmente são dois grupos ALL (pois ele é mais predominante nos dados de entrada) e um AML.

```
Número real de ALL:  47 . Número real de AML:  25
Grupo  1 :
Número de indivíduos no grupo:  31 . Sendo  18  ALL e  13  AML.
Grupo  2 :
Número de indivíduos no grupo:  27 . Sendo  26  ALL e  1  AML.
Grupo  3 :
Número de indivíduos no grupo:  14 . Sendo  3  ALL e  11  AML.
```

Um dos melhores resultados para 3 grupos. Vemos um grupo predominantemente ALL, outro AML, e um terceiro grupo bastante misturado.

```
Número real de ALL:  47 . Número real de AML:  25
Grupo  1 :
Número de indivíduos no grupo:  27 . Sendo  23  ALL e  4  AML.
Grupo  2 :
Número de indivíduos no grupo:  21 . Sendo  1  ALL e  20  AML.
Grupo  3 :
Número de indivíduos no grupo:  24 . Sendo  23  ALL e  1  AML.
```

Um resultado mais típico para 3 grupos: dois grupos predominantemente ALL e um AML.