Aluno: Felipe Colombelli Cartão: 00287698

Questão 3

Código:

Foi utilizado o editor VS Code pela facilidade de formatação que inclui também o número de linhas.

Primeiro são feitos os imports necessários e estipulado o número de clusters e de iterações feitas pelo K-means.

```
1   import pandas as pd
2   import numpy as np
3   from sklearn.cluster import KMeans
4   from sklearn.preprocessing import StandardScaler
5   from sklearn.decomposition import PCA
6   import matplotlib.pyplot as plt
7
8   # Number of clusters used in K-means
9   k = 2
10  numIterations = 50
```

Em seguida, carrega-se o conjunto de dados e faz-se o subsetting destes dados de modo a selecionar apenas os valores de expressão gênica, sem o rótulo ALL/AML associado. Para transformar as amostras em linhas - e não colunas, que é como os dados estão originalmente estruturados - faz-se a transposição deste conjunto de dados.

```
73  # Loads the dataframe
74  df = pd.read_csv("/home/colombelli/Documents/datasets/leukemia_big.csv", header=None)
75
76  # Gets the transposed dataframe without the column that labels the samples
77  x = df.T.iloc[0:, 1:]
```

Tendo a porção de dados que interessa, podemos utilizar o algoritmo de K-Means implementado pela biblioteca Scikit-Learn, muito utilizada para realização de experimentos científicos na academia e na indústria.

```
79  # Clusters the data
80  kmeans = KMeans(n_clusters=k)
81  kmeans.fit(x)
82
83  # Get information about the clusterization
84  predictedLabels = kmeans.predict(x)
85  centroids = kmeans.cluster_centers_
```

Após a clusterização, pega-se a coluna original do conjunto de dados contendo a informação do tipo de leucemia associado àqueles dados de expressão gênica (ALL/AML). Tendo isso, é possível contar as ocorrências de amostras com cada tipo de leucemia em cada um dos clusters.

```
realLabels = df.iloc[0:1].values[0]

occurrences = countOccurrences(k, realLabels, predictedLabels)
printOccurrences(occurrences)
print("\n\n\n")
```

A função que conta as ocorrências de cada tipo de leucemia em cada cluster é implementada da seguinte maneira: primeiro um dicionário vazio é criado; então itera-se pelos rótulos verificando o tipo de leucemia daquela amostra e em que cluster caiu; o dicionário tem uma chave para cada par Leucemia/Cluster que aparecer, por isso a primeira operação do loop que a função tenta fazer é aumentar a contagem daquela chave, se uma exceção for levantada por essa operação é porque não existe uma chave para aquele par ainda, logo, cria-se a chave assinalando o inteiro 1 como valor inicial.

```
def countOccurrences(k, realLabels, predictedLabels):

    occurrences = {}
    for idx, label in enumerate(realLabels):
        try:
            occurrences[label+ " - cluster: " + str(predictedLabels[idx])] += 1
        except:
            occurrences[label+ " - cluster: " + str(predictedLabels[idx])] = 1

    return occurrences


def printOccurrences(occurrences):
    print("\n\nOccurrences found:\n\n")
    for x, y in occurrences.items():
        print(x, "- occurrences:", y)

    return
```

Caso o número de clusters seja apenas dois, faz sentido também calcularmos a acurácia do modelo, pois temos apenas dois rótulos possíveis e idealmente cada amostra de cada tipo de leucemia deveria ser separada no respectivo cluster que representa aquele tipo de leucemia.

```
if k == 2:
    calculateAccuracy(realLabels, predictedLabels)
    print("\n\n\n")
```

A acurácia é calculada da seguinte maneira: primeiro precisamos transformar os rótulos ‘ALL’ e ‘AML’ em 0 ou 1, correspondendo ao cluster 0 ou 1 retornados pelo

K-Means; porém, como não sabemos se o primeiro cluster foi associado ao ALL ou ao AML, devemos testar as duas possibilidades, aquela que der o maior número de acertos será a escolhida.

Nota: de fato, foi testado sem realizar esta checagem e, às vezes, a acurácia era mais de 98% e, nas execuções seguintes, menos de 2%, indicando que os clusters eram estocasticamente associados ao tipo de leucemia, e não o 0 ser sempre necessariamente associado ao ALL ou AML.

Após essa transformação dos rótulos, comparamos com o vetor de predições feitas pelo modelo. Podemos comparar os dois vetores simplesmente com a operação de ==, que retornará um vetor com a mesma quantidade de elementos cujo domínio são variáveis booleanas. Enfim, podemos contar o número de *True*s utilizando o método *np.sum* e isso nos dirá quantas amostras foram corretamente classificadas.

```
36  def convertLabelsTo01(realLabels, ALL, AML):
37
38      realLabels_01 = []
39      for label in list(realLabels):
40          if label == 'ALL':
41              realLabels_01.append(ALL)
42          elif label == 'AML':
43              realLabels_01.append(AML)
44      return realLabels_01
45
46
47  def calculateAccuracy(realLabels, predictedLabels):
48
49      realLabels_01 = convertLabelsTo01(realLabels, 0, 1)
50      realLabels_10 = convertLabelsTo01(realLabels, 1, 0)
51
52      rigthGuesses01 = (np.array(realLabels_01) == predictedLabels)
53      rigthGuesses10 = (np.array(realLabels_10) == predictedLabels)
54
55      # Because there's no way to know which cluster will be assigned to each class
56      rigthGuesses = max(np.sum(rigthGuesses01), np.sum(rigthGuesses10))
57
58      numSamples = len(realLabels)
59      numRigthGuesses = np.sum(rigthGuesses)
60
61      accuracy = numRigthGuesses / numSamples * 100
62
63      print("Number of samples:", numSamples)
64      print("Number of rigth classifications:", numRigthGuesses)
65      print("Accuracy:", accuracy)
66      return
```

Tendo estas métricas, o próximo passo para analisar os dados é visualizá-los graficamente. Para isso, utilizaremos o procedimento estatístico Principal Component Analysis, ou simplesmente PCA.

Projetamos as sete mil dimensões em duas com a utilização do PCA também implementado pelo scikit-learn. Os dois novos componentes principais definidos pelo PCA são apenas as duas principais dimensões de variação, não há um significado em particular para cada componente principal.

Fonte: https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60

```
102  # PCAing
103  pca = PCA(n_components=2)
104  principalComponents = pca.fit_transform(x)
105  pcaDf = pd.DataFrame(data = principalComponents,
106                       columns = ['principal component 1', 'principal component 2'])
107  pcaDf['cluster'] = predictedLabels
108
```

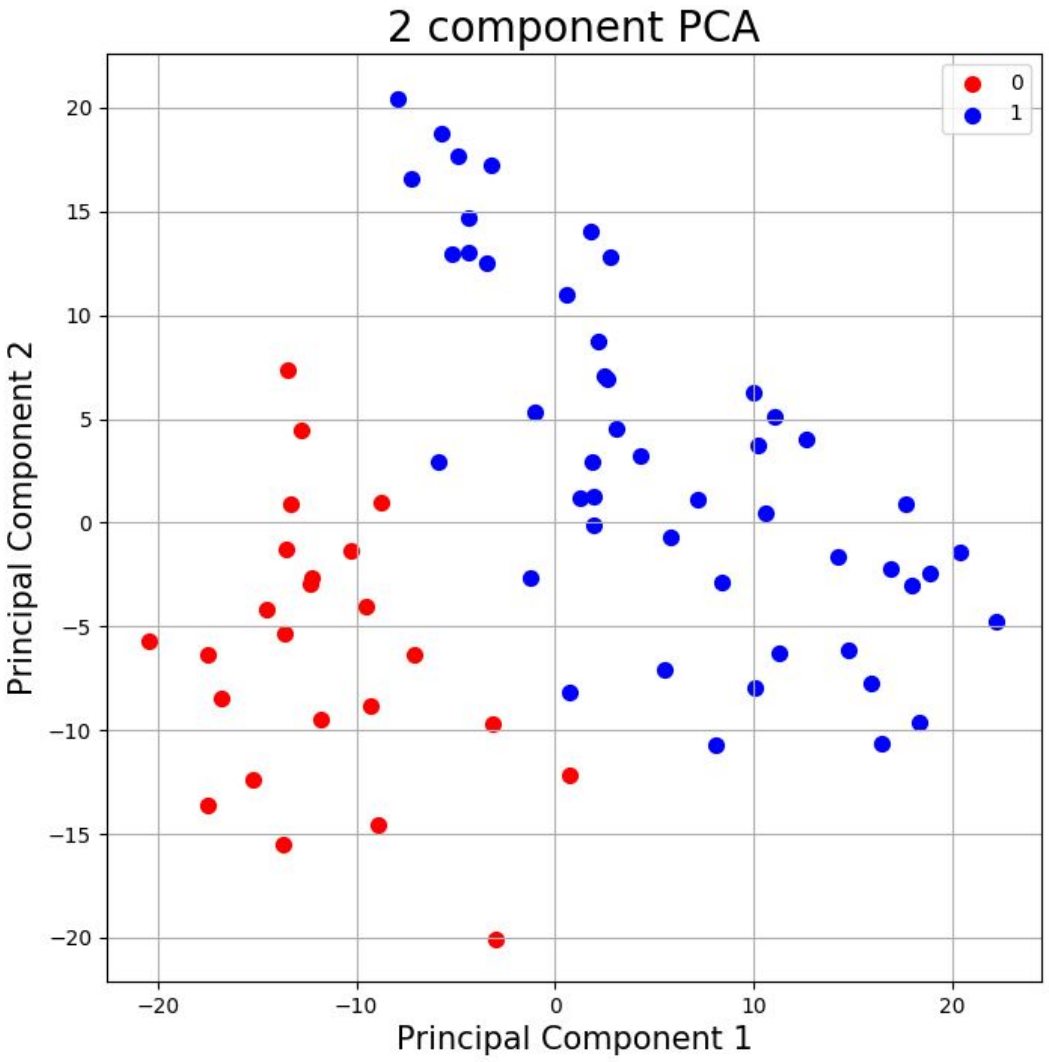
E finalmente o restante do código foca na visualização dos dados obtidos pelo PCA utilizando os objetos e métodos de uma ferramenta, matplotlib, logo, foram omitidos no presente relatório.

Resultados:

Para k = 2, a melhor acurácia obtida dentre diversas execuções, foi de 98.6%, classificando erroneamente apenas uma amostra AML como sendo ALL. Seguem os resultados de ocorrências e acurácia de saída do algoritmo.

```
colombelli@colombelli:~/Documents/test$ python3 e4-3.py


Occurrences found:


ALL - cluster: 1 - occurrences: 47
AML - cluster: 0 - occurrences: 24
AML - cluster: 1 - occurrences: 1




Number of samples: 72
Number of rigth classifications: 71
Accuracy: 98.61111111111111
```

E a seguir o plot do PCA:

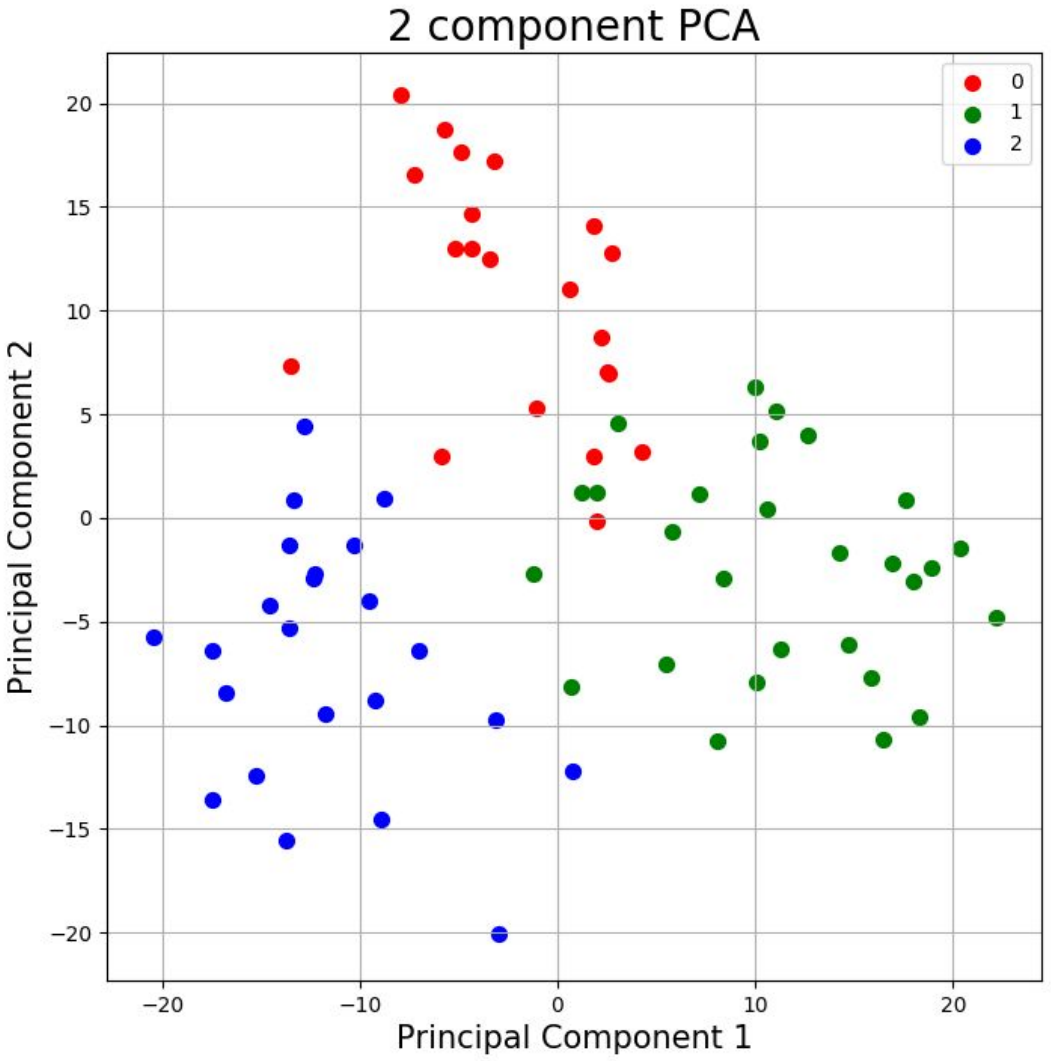


Para k = 3, não faz mais sentido medirmos a acurácia, entretanto podemos analisar o agrupamento realizado. Além disso, foram utilizadas 100 iterações, pois com 50, após múltiplas execuções, os resultados não estavam convergindo para resultados parecidos.

```
colombelli@colombelli:~/Documents/test$ python3 e4-3.py


Occurrences found:


ALL - cluster: 0 - occurrences: 20
ALL - cluster: 1 - occurrences: 27
AML - cluster: 2 - occurrences: 23
AML - cluster: 1 - occurrences: 1
AML - cluster: 0 - occurrences: 1
```

Pelo número de ocorrências, podemos inferir que o modelo utilizou o cluster 0 e 1 para classificar amostras ALL, e o cluster 2 para amostras AML. Duas amostras de AML foram, então, classificadas erroneamente.

E finalmente o plot do PCA:

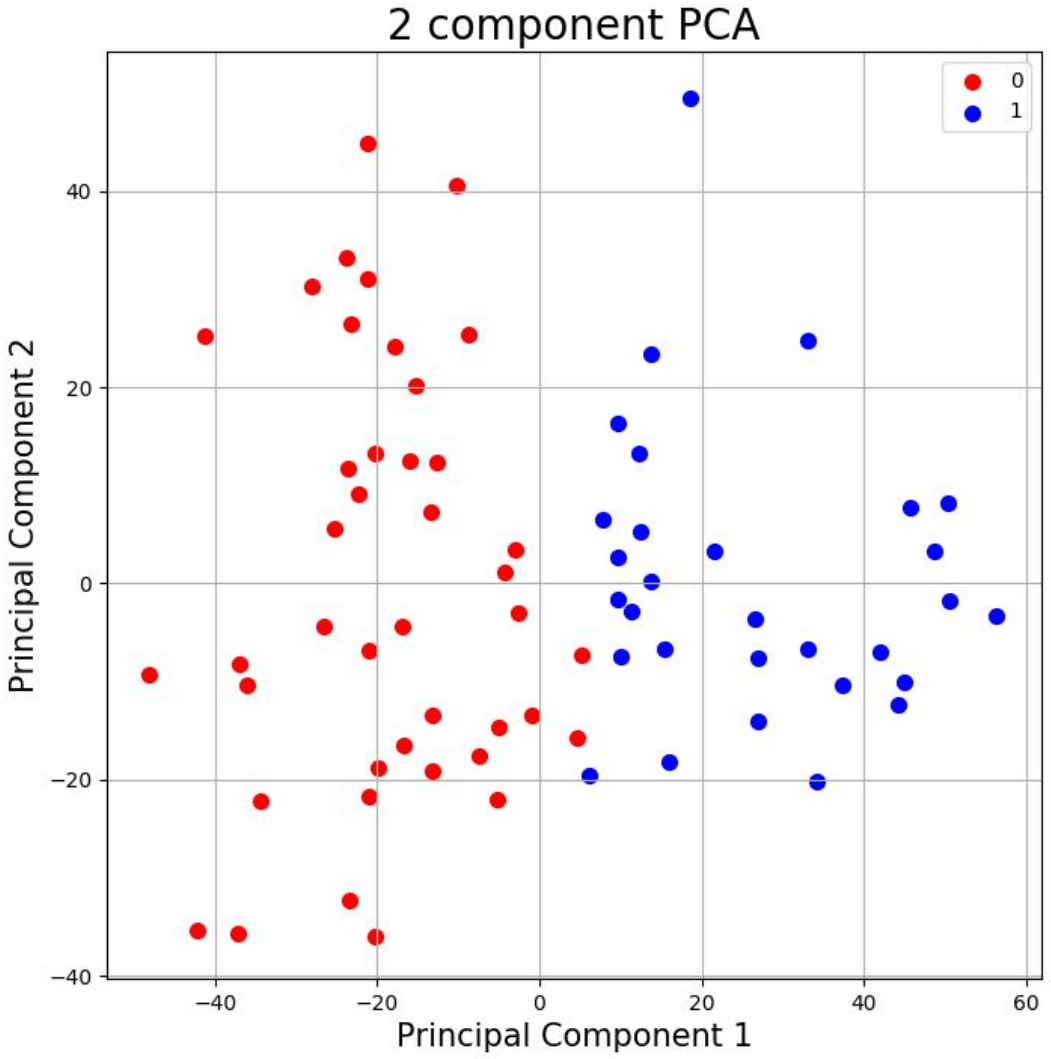


Normalização dos dados

Muito se fala de normalização dos dados em Machine Learning que de fato é um procedimento fundamental para o desempenho de vários tipos de modelos preditivos. Em se tratando de K-Means, resolvi investigar qual a influência da normalização dos dados pela técnica *Z-Score Normalization* que centra os valores numa média de 0 com desvio padrão de 1.

Para k = 2, os seguintes resultados foram obtidos:

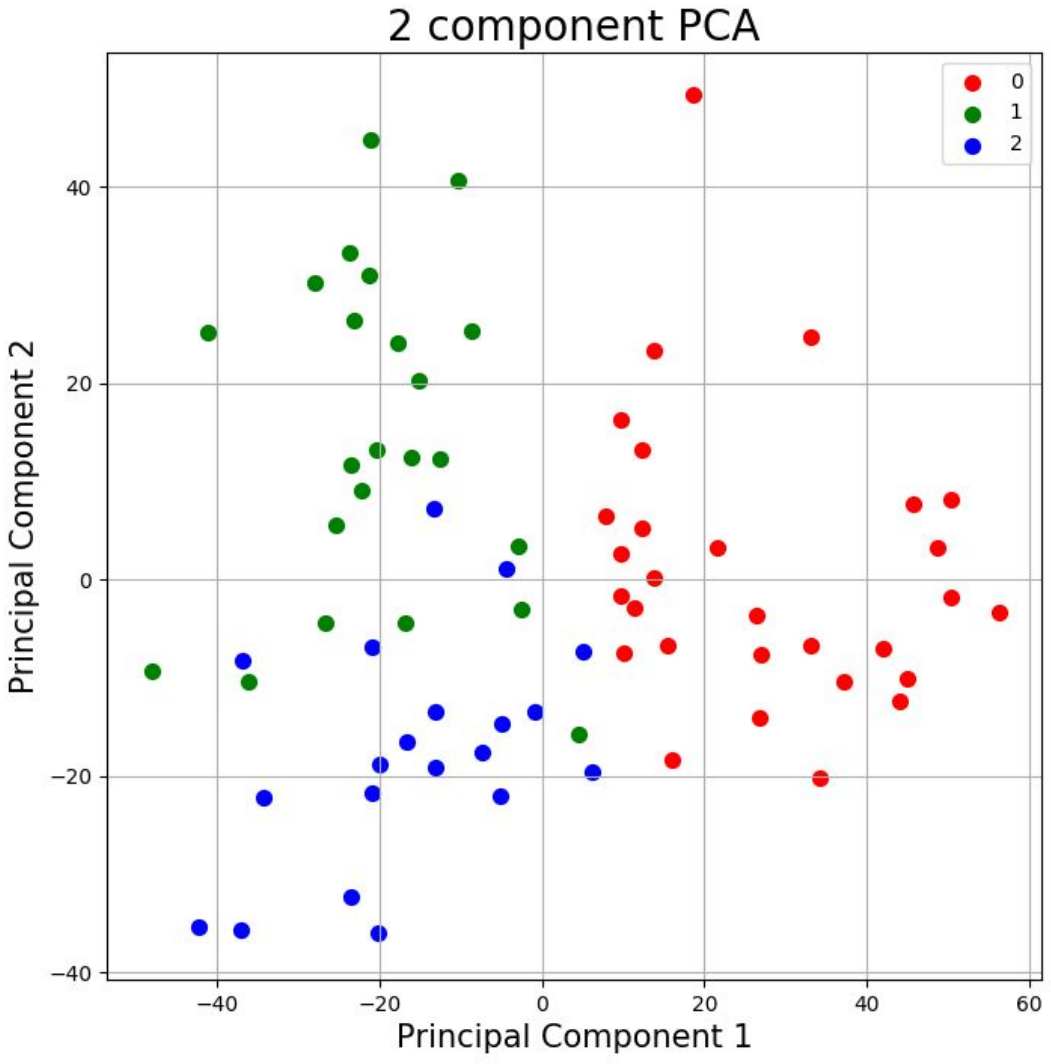
```
colombelli@colombelli:~/Documents/test$ python3 e4-3.py

Occurrences found:


ALL - cluster: 0 - occurrences: 20
ALL - cluster: 1 - occurrences: 27
AML - cluster: 0 - occurrences: 22
AML - cluster: 1 - occurrences: 3



Number of samples: 72
Number of rigth classifications: 49
Accuracy: 68.05555555555556
```



Indicando que a performance do modelo caiu drasticamente.

E **para k = 3**, os seguintes resultados foram obtidos:

```
colombelli@colombelli:~/Documents/test$ python3 e4-3.py

Occurrences found:

ALL - cluster: 2 - occurrences: 19
ALL - cluster: 0 - occurrences: 26
ALL - cluster: 1 - occurrences: 2
AML - cluster: 1 - occurrences: 21
AML - cluster: 0 - occurrences: 3
AML - cluster: 2 - occurrences: 1
```



As 6 amostras agrupadas erroneamente, se comparando com as 2 erradas do experimento sem normalização, reúnem ainda mais evidências que sugerem evitar tal procedimento para este domínio de problema quando utilizado o algoritmo de K-Means para a classificação dos dados.