TP555 - AI/ML

Lista de Exercícios #9

k-Means

1. Cite alguns exemplos de aplicações reais do algoritmo k-Means.
2. Neste exercício, você irá utilizar o algoritmo k-Means com k = 3 para encontrar manualmente os centróides ótimos para o conjunto de dados de treinamento abaixo. Considere os centróides iniciais, C0, C1 e C2, dados ao lado. Utilize a **distância Euclidiana** para encontrar o cluster a que cada exemplo de entrada pertence. Apresente todos os cálculos necessários para se encontrar os centróides ótimos.

x1	x2
1	4
4	3
4	5
3	6
6	7
3	3
2	5
2	2
2	3

C0		C1		C2	
x1	x2	x1	x2	x1	x2
5	3	1	3	3	4

Em seguida, faça o seguinte

A. Crie uma figura mostrando os dados de treinamento.
B. Utilizando os centróides iniciais dados acima, instancie um objeto da classe KMeans da biblioteca SciKit-Learn.

```
km = KMeans(n_clusters=3, init=init_clusters)
```

C. Treine o modelo e imprima os centróides ótimos. Os valores encontrado pelo KMeans devem ser os mesmos que você encontrou manualmente. Os valores ótimos podem ser impressos como mostrado abaixo.

```
for i in range(0,3):
    print('Centroid %d: (%1.2f, %1.2f)' % (i,km.cluster_centers_[i][0],km.cluster_centers_[i][1]))
```

D. Quantas iterações foram necessárias para se treinar o modelo? (Dica: a documentação da classe KMeans pode ser acessada via:

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html)

E. Crie uma figura com os dados de treinamento indicando através de cores ou marcadores diferentes à que clusters cada um deles pertence além de mostrar os centróides encontrados pelo k-Means.

3. Crie um conjunto de dados de treinamento utilizando a função ***make_blobs*** como mostrado abaixo.

```
X, y = make_blobs(n_samples=150, n_features=2, centers=5, cluster_std=1.0, shuffle=True, random_state=42)
```

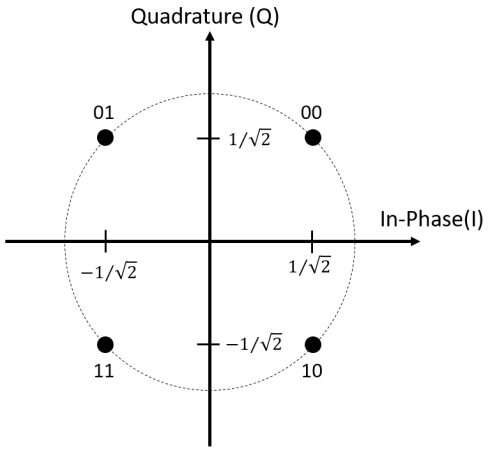
Em seguida, faça o seguinte

A. Crie uma figura mostrando os dados de treinamento.
B. Após inspecionar a figura, decida quantos clusters devem ser utilizados com o algoritmo do k-Means.
C. Instancie um objeto da classe KMeans da biblioteca SciKit-Learn.

```
km = KMeans(n_clusters=????, init=init_clusters)
```

D. Treine o modelo e imprima os centróides ótimos.
E. Quantas iterações foram necessárias para se treinar o modelo? (Dica: a documentação da classe KMeans pode ser acessada via: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html)
F. Crie uma figura com os dados de treinamento indicando através de cores ou marcadores diferentes à que clusters cada um deles pertence, além de mostrar os centróides encontrados pelo k-Means.

4. Neste exercício, você irá utilizar o algoritmo do k-Means para clusterizar os dados da modulação digital QPSK, ou seja, realizar a detecção de símbolos QPSK. Os símbolos QPSK são dados pela figura e tabela abaixo.



bits	Símbolo (I + jQ)
00	$\frac{1}{\sqrt{2}} + j\frac{1}{\sqrt{2}}$
01	$-\frac{1}{\sqrt{2}} + j\frac{1}{\sqrt{2}}$
10	$\frac{1}{\sqrt{2}} - j\frac{1}{\sqrt{2}}$
11	$-\frac{1}{\sqrt{2}} - j\frac{1}{\sqrt{2}}$

O resultado do seu *'clusterizador'* (neste caso, um detector) pode ser comparado com a curva da taxa de erro de símbolo (SER) teórica, a qual é dada por

$$\text{SER} = erfc\left(\sqrt{\frac{Es}{2N0}}\right) - \frac{1}{4}erfc\left(\sqrt{\frac{Es}{2N0}}\right)^2.$$

Utilizando a classe KMeans do módulo cluster da biblioteca sklearn, faça o seguinte

A. Construa um detector para realizar a detecção dos símbolos QPSK.
   a. Gere N = 1000000 símbolos QPSK aleatórios.
   b. Passe os símbolos através de um canal AWGN.
   c. Detecte a probabilidade de erro de símbolo para cada um dos valores do vetor Es/N0 = [-2, 0, 2, 4, 6, 8, 10,12, 14, 16, 18, 20].

B. Apresente um gráfico comparando a SER simulada e a SER teórica versus os valores de Es/N0 definidos acima.

C. Podemos dizer que a curva simulada se aproxima da curva teórica da SER?

(**Dica**: Como a ordem dos ***centróides*** encontrados pelo k-Means é aleatória, o valor do símbolo que o ***centróide*** representa pode ser encontrado através de estimativa por máxima verossimilhança (do inglês, maximum likelihood - ML), ou seja, testa-se o ***centróide*** de um símbolo detectado contra todos os símbolos possíveis, sendo o símbolo escolhido aquele que apresentar o menor erro.)

(**Dica**: A função ***erfc*** pode ser importada da seguinte forma: *from scipy.special import erfc*).

(**Dica**: A função ***train_test_split*** pode dividir qualquer número de vetores de entrada em vetores de treinamento e teste. Veja o exemplo abaixo onde três vetores de entrada, a, e c, são divididos em vetores de treinamento e teste.

```
# Split array into random train and test subsets.
a_train, a_test, b_train, b_test, c_train, c_test = train_test_split(a, b, c, random_state=42)
```

Para mais informações, leia a documentação da função ***train_test_split***: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)

(**Dica**: Uma rápida revisão sobre taxa de erro de símbolo pode ser encontrada no link: http://www.dsplog.com/2007/11/06/symbol-error-rate-for-4-qam/).

5. **Exercício sobre k-Means**: Detecção 16QAM com k-Means. Neste exercício você irá utilizar o clusterizador k-Means, para realizar a detecção de símbolos da modulação digital 16QAM. Os símbolos 16QAM são dados pelo trecho de código abaixo, o qual também apresenta uma função que deve ser utilizada para modular os bits em símbolos 16QAM.

```
mapping_table = [-3-3j, -3-1j, -3+3j, -3+1j, -1-3j, -1-1j, -1+3j, -1+1j, 3-3j, 3-1j, 3+3j, 3+1j, 1-3j, 1-1j, 1+3j, 1+1j]

def mod(bits):
    symbols =  np.zeros((len(bits),),dtype=complex)
    for i in range(0,len(bits)): symbols[i] = mapping_table[bits[i]]/np.sqrt(10)
    return symbols
```

Um exemplo de código para gerar símbolos 16QAM é dado à seguir

```
# Generate N 4-bit symbols.
bits = np.random.randint(0,16,N)

# Modulate the binary stream into 16QAM symbols.
symbols = mod(bits)
```

O resultado do seu clusterizador (neste caso, um detector) deve ser comparado com a curva da taxa de erro de símbolo (SER) teórica do 16QAM, a qual é dada por

$$\text{SER} = 2\left(1 - \frac{1}{\sqrt{M}}\right)erfc\left(k\sqrt{\frac{Es}{N0}}\right) - \left(1 - \frac{2}{\sqrt{M}} + \frac{1}{M}\right)erfc\left(k\sqrt{\frac{Es}{N0}}\right)^2,$$

onde M é a ordem da modulação, i.e., 16, e $k = \sqrt{\frac{3}{2(M-1)}}$ é o fator de normalização da energia dos símbolos. O trecho de código abaixo implementa a equação da SER apresentada acima.

```
# Theoretical 16QAM BER.
M = 16
k = np.sqrt(3/(2*(M-1)))
EsN0 = 10.0**(EsN0dB[idx]/10.0)
ser_theo = 2*(1 - (1/np.sqrt(M)) )*erfc(k*np.sqrt(EsN0)) - (1 - (2/np.sqrt(M)) + (1/M)
)*(erfc(k*np.sqrt(EsN0)))**2.0
```

O trecho de código abaixo gera o vetor de ruído AWGN e em seguida o adiciona aos símbolos transmitidos, onde EsN0dB é o valor da relação energia de símbolo por densidade espectral do ruído (a sequência de valores para EsN0dB será especificada abaixo), N é o número de símbolos 16QAM, s é o vetor com os N símbolos 16QAM e y é o sinal recebido, ou seja, o sinal que passou pelo canal AWGN.

```
# Convert into linear scale.
EsN0Lin = 10.0**(-(EsN0dB/10.0))

# Generate noise vector.
noise = np.sqrt(1.0/2.0)*(np.random.randn(N, 1) + 1j*np.random.randn(N, 1))

# Pass symbols through AWGN channel.
y = s + np.sqrt(EsN0Lin)*noise
```

Agora, utilizando a classe **KMeans** do módulo **cluster** da biblioteca **SciKit-Learn**, faça o seguinte

A. Plote a constelação de símbolos da modulação 16QAM e a analise.
B. Após analisar a figura com a constelação, responda: Quantos clusters são necessários para esta clusterização?
C. Construa um clusterizador, utilizando o k-Means, para realizar a detecção dos símbolos 16QAM.
    a. Gere N = 100000 símbolos 16QAM aleatórios.
    b. Passe os símbolos através de um canal AWGN.

c. Detecte a probabilidade de erro de símbolo para cada um dos valores do vetor Es/N0 = [0, 2, 4, 6, 8, 10,12, 14, 16, 18, 20].

D. Apresente um gráfico comparando a SER simulada e a SER teórica versus os valores de Es/N0 definidos acima.

E. Podemos dizer que a curva simulada se aproxima da curva teórica da SER?

(**Dica**: Como a ordem dos ***centróides*** encontrados pelo k-Means é aleatória, o valor do símbolo que o ***centróide*** representa pode ser encontrado através de estimativa por máxima verossimilhança (do inglês, maximum likelihood - ML), ou seja, testa-se o ***centróide*** de um símbolo detectado contra todos os símbolos possíveis, sendo o símbolo escolhido aquele que apresentar o menor erro.)

(**Dica**: A função ***train_test_split*** pode dividir qualquer número de vetores de entrada em vetores de treinamento e teste. Veja o exemplo abaixo onde três vetores de entrada, a, e c, são divididos em vetores de treinamento e teste.

```
# Split array into random train and test subsets.
a_train, a_test, b_train, b_test, c_train, c_test = train_test_split(a, b, c, random_state=42)
```

Para mais informações, leia a documentação da função ***train_test_split***: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)

(**Dica**: A função **erfc** pode ser importada da seguinte forma: **from scipy.special import erfc**).

(**Dica**: Uma rápida revisão sobre a taxa de erro de símbolo para modulações M-QAM pode ser encontrada no link: http://www.dsplog.com/2012/01/01/symbol-error-rate-16qam-64qam-256qam/).

6. Neste exercício, você irá aprender e utilizar 2 métodos para se escolher o parâmetro k, ou seja, o número de clusters. Crie um conjunto de dados utilizando o trecho de código abaixo.

```
N = 1000
# Generating the sample data from make_blobs
# This particular setting has one distinct cluster and 3 clusters placed close
# together.
X, y = make_blobs(n_samples=N, n_features=2, centers=4, cluster_std=1,
center_box=(-10.0, 10.0), shuffle=True, random_state=1)
```

Leia as referências abaixo para aprender sobre os métodos do cotovelo e da silhueta. Em seguida, faça o seguinte

A. Plote os dados do conjunto de testes.

B. Visualmente, quantos clusters você acha que seriam necessários para agrupar os dados?

C. Utilizando o método do cotovelo (do inglês, *Elbow Method)*, encontre o valor mais apropriado para k.

D. Com o(s) resultado(s) do método do cotovelo, crie uma figura com os dados de treinamento indicando através de cores ou marcadores diferentes à que clusters

cada um deles pertence, além de mostrar os centróides encontrados pelo k-Means.

E. Utilizando o método da silhueta (do inglês, *Silhouette Method*), encontre o(s) valor(es) mais apropriado(s) para k.
F. Com o(s) resultado(s) do método da silhueta, crie uma figura com os dados de treinamento indicando através de cores ou marcadores diferentes à que clusters cada um deles pertence, além de mostrar os centróides encontrados pelo k-Means.

Referências

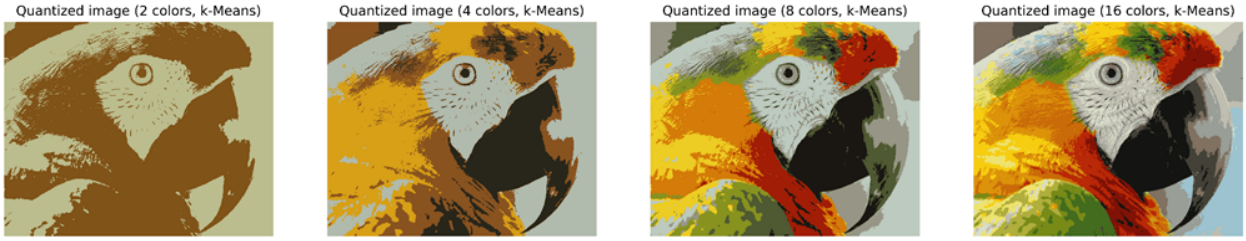
[1] 'Elbow Method', https://jtemporal.com/kmeans-and-elbow-method/
[2] 'Elbow and Silhouette Methods', https://medium.com/analytics-vidhya/how-to-determine-the-optimal-k-for-k-means-708505d204eb
[3] 'Elbow and Silhouette Methods', https://medium.com/@masarudheena/4-best-ways-to-find-optimal-number-of-clusters-for-clustering-with-python-code-706199fa957c

7. **Exercício sobre k-Means**: Quantização de cores de uma imagem. Neste exercício, os pixels da imagem img2.jpg são representados em um espaço 3D e o k-Means será usado para reduzir o número de cores necessárias para mostrar a imagem. Na literatura de processamento de imagens, o ***codebook*** obtido pelo k-Means (i.e., os centros dos clusters) é chamado de paleta de cores. Usando um único byte, é possível endereçar até 256 cores, enquanto uma codificação RGB requer 3 bytes por pixel, o que dá um total de 256*256*256 = 16777216 cores, ou seja, mais de 16 milhões de cores diferentes. Uma cor em codificação RGB geralmente é codificada como uma tupla de 3 elementos com 8 bits cada, portanto, cada dimensão assume um valor dentro do intervalo [0, 255], em que 0 representa a ausência de cor, enquanto 255 representa a presença total da cor.

   A quantização de cores encontra um pequeno número de cores representativas em uma determinada imagem. Cada pixel produz um padrão tridimensional no espaço de cores RGB. Usando ***k-Means***, podemos agrupar todos os pixels de uma imagem em k clusters e atribuir a cada pixel da imagem original a cor representada pelo centro do cluster mais próximo. Assim, uma imagem contendo milhões de cores pode ser compactada em uma imagem contendo k cores diferentes. Após ler as referências abaixo, faça o seguinte

   A. Carregue a imagem img2.jpg e verifique as dimensões da array carregada.
   B. Exiba a imagem original em seu notebook.
   C. Torne os dados da imagem, i.e., a array carregada no item (A), em uma array 2D.
   D. Treine 4 clusterizadores k-Means para 2, 4, 8 e 16 cores, respectivamente, ou seja, 2, 4, 8 e 16 clusters. Treine cada clusterizador com apenas 1000 elementos tomados aleatoriamente da imagem original, ou seja, uma array com 1000 x 2 elementos.

E. Faça a predição com a array 2D contendo a imagem original para cada um dos clusterizadores k-Means.
F. Para cada um dos 4 clusterizadores treinados, atribua a cada pixel da imagem original o valor do centroide do cluster mais próximo.
G. Visualize a imagem quantizada para cada um dos 4 clusterizadores, conforme mostrado na figura abaixo.



H. Se cada uma dos 3 valores da codificação RGB é representado por 1 byte (i.e., 8 bits), ou seja, um total de 24 bits para representar a cor de cada pixel da imagem, qual o tamanho aproximado da imagem original e de cada uma das 4 imagens quantizadas?
I. O que você conclui após ter resolvido este exercício?

Referências

[1] https://lmcaraig.com/color-quantization-using-k-means
[2] https://github.com/adityaguptai/Color-Quantization-using-K-Means/blob/master/Color%20Quantization%20Using%20K-Means.ipynb
[3] https://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html#sphx-glr-auto-examples-cluster-plot-color-quantization-py
[4] https://en.wikipedia.org/wiki/Color_quantization
[5] https://www.idtools.com.au/segmentation-k-means-python/

8. **Exercício sobre algoritmos semi-supervisionados utilizando k-Means e Regressão Logística:** Neste exercício, iremos verificar como o aprendizado semi-supervisionado funciona, usando clustering e classificação. Faça o seguinte:
   a. Carregue a base de dados de dígitos do SciKit-Learn. Use o código abaixo para isso.

```
from sklearn.datasets import load_digits

X_digits, y_digits = load_digits(return_X_y=True)
```

   b. A separe em dois conjuntos, treinamento e validação, com 80% e 20%, respectivamente.
   c. Treine um Regressor Logístico com apenas 50 amostras do conjunto de treinamento. Você pode usar as primeiras 50 amostras.
   d. Qual o score obtido?

e. Agora, usando kMeans com k=50, encontre as 50 imagens mais representativas, ou seja, as imagens mais próximas de cada um dos k centróides.
f. De posse destas 50 imagens mais representativas, as rotule e treine novamente um Regressor Logístico.
g. Qual o score obtido?
h. Usando as imagens mais representativas, propague seus rótulos para todas as outras instâncias que pertencem ao mesmo cluster.
i. Treine um novo Regressor Logístico com esta base de treinamento rotulada com as imagens mais representativas.
j. Qual o score obtido?
k. Você deve ter percebido um pequeno aumento no score, mas provavelmente deveríamos ter propagado os rótulos apenas para os exemplos mais próximos do centróide, porque, ao propagar para o cluster inteiro, certamente incluímos alguns **outliers**. Vamos apenas propagar os rótulos até as amostras que estejam dentro do 75º percentil mais próximo do centróide. (**Dica**: Use a função ***percentile*** da biblioteca NumPy.)
l. Treine um novo Regressor Logístico com esta base de treinamento.
m. Qual o score obtido?
n. Realize algumas iterações de aprendizado ativo e verifique a melhoria do score.

Aprendizado ativo

Para continuar melhorando seu modelo e seu conjunto de treinamento, a próxima etapa pode ser fazer algumas rodadas de aprendizado ativo: isto é, quando um especialista humano interage com o algoritmo de aprendizado, fornecendo rótulos quando o algoritmo precisa deles. Existem muitas estratégias diferentes para a aprendizagem ativa, mas uma das mais comuns é chamada de amostragem de incerteza:

- O modelo é treinado com os exemplos rotulados reunidos até agora, e esse modelo é usado para fazer previsões em todos exemplos não rotulados.
- Os exemplos para as quais o modelo é mais incerto (ou seja, quando sua probabilidade estimada é mais baixa) devem ser rotulados pelo especialista.
- Em seguida, você apenas repete esse processo repetidamente, até que a melhoria do desempenho deixe de valer o esforço de rotulagem.

Outras estratégias incluem rotular os exemplos que resultam na maior mudança do modelo, ou a maior queda no erro de validação do modelo, ou os exemplos em que diferentes modelos discordam (por exemplo, um SVM, uma floresta aleatória e assim por diante).