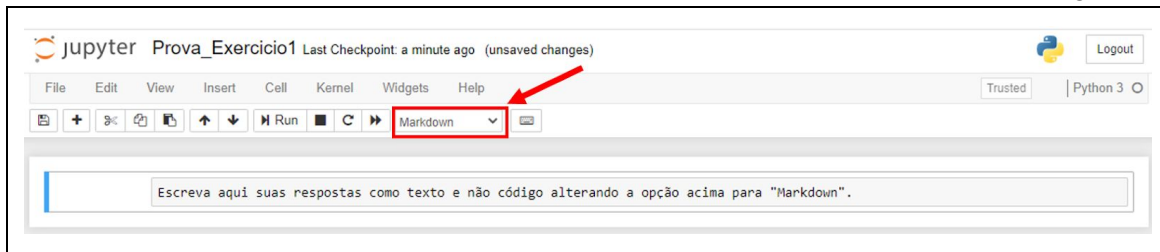


TP555 - AI/ML

Prova (1S2020)

Orientações

- Crie um notebook do Jupyter para cada um dos exercícios, mesmo para os teóricos. Para os exercícios teóricos ou mesmo exercícios que requerem respostas textuais, crie um nova célula e mude o tipo da célula para “Markdown” conforme mostrado na figura.



- Desenhos podem ser feitos à mão ou qualquer outro SW de sua preferência. Colocar na pasta **prova** uma figura ou pdf do desenho.
- Crie uma pasta chamada **prova** no seu repositório e coloque todos os notebooks lá.
- A prova deve ser resolvida **individualmente**.
- A interpretação faz parte da prova.
- Boa sorte!

Questões

- 1) **Exercício sobre regressão polinomial:** Neste exercício, você irá deve descobrir qual o polinômio que melhor aproxima um conjunto de pontos ruidosos. Utilize o arquivo [poly_reg_p.csv](#) onde a primeira coluna contém os valores de x e a segunda de y. O arquivo contém a versão ruidosa da função original, ou seja o modelo gerador ao qual ruído é adicionado. Em seguida
 - A. Apresente o gráfico de x versus y, mostrando os **pontos** amostrados do modelo gerador.
 - B. Encontre uma **aproximação polinomial** que represente bem os dados do arquivo. Para encontrar a melhor aproximação, utilize os seguintes métodos: validação cruzada holdout (com 70% do conjunto original para treinamento e 30% para validação), validação cruzada k-fold (com k=10 folds), validação cruzada leave-p-out (com p=1). (**Dica:** Analise polinômios com ordem variando de 1 até 20.)
 - C. Plote as curvas de aprendizado para as ordens de polinômio que melhor aproximam o modelo gerador, inclua também as ordens 1, 2 e 20. O que você pode concluir após analisar estes resultados?

- D. Em seguida, de posse da melhor ordem de polinômio que aproxima os dados do modelo gerador, treine o modelo com todos os dados do arquivo. Utilize padronização de atributos com a classe `StandardScaler` da biblioteca `SciKit-Learn`.
- E. Plote um gráfico que mostre os pontos ruidosos do arquivo ***poly_reg_p.csv*** e os valores encontrados com o modelo para os valores de x vindos do arquivo csv, ou seja, use o modelo para “prever” os valores de y com os valores de x vindos do arquivo.

2) Exercício sobre classificação de bayes: Usando a teoria Bayesiana de decisão e os dados de treinamento abaixo, encontre a probabilidade de uma pessoa com os seguintes atributos: **idade ≤ 30 , renda = Média, estudante = Sim e classificação de crédito = boa**, comprar ou não um personal computer (PC). Baseado nas probabilidades encontradas, esta pessoa compraria ou não o PC? Apresente todos os cálculos necessários para se calcular as probabilidades.

Exemplo	Idade	Renda	Estudante	Classificação de crédito	Classificação: Compra o PC?
1	≤ 30	Alta	Não	Boa	Não
2	≤ 30	Alta	Não	Excelente	Não
3	31 a 40	Alta	Não	Boa	Sim
4	> 40	Média	Não	Boa	Sim
5	> 40	Baixa	Sim	Boa	Sim
6	> 40	Baixa	Sim	Excelente	Não
7	31 a 40	Baixa	Sim	Excelente	Sim
8	≤ 30	Média	Não	Boa	Não
9	≤ 30	Baixa	Sim	Boa	Sim
10	> 40	Média	Sim	Boa	Sim
11	≤ 30	Média	Sim	Excelente	Sim
12	31 a 40	Média	Não	Excelente	Sim
13	31 a 40	Alta	Sim	Boa	Sim
14	> 40	Média	Não	Excelente	Não

3) Exercício sobre regressão logística: Utilizando a seguinte função hipótese $h_a(x) = f(a_0 + a_1x_1 + a_2x_2)$, onde $f(\cdot)$ é a função de limiar sigmóide (ou logística), e o algoritmo do ***gradiente descendente em batelada com early-stopping*** que você implementou para a lista #5, treine um classificador de regressão logística para classificar os dados gerados através da execução do código abaixo.

```
# Import all necessary libraries.
```

```

import numpy as np
from sklearn.model_selection import train_test_split

# Create the dataset.
X = np.random.randn(1000, 2)
y = np.array(np.logical_xor(x[:, 0] > 0, x[:, 1] > 0), dtype=int)

# Split array into random train and test subsets.
x_train, x_test, y_train, y_test = train_test_split(X, y, random_state=42)

```

Em seguida, faça o seguinte

- Plote um gráfico mostrando as diferentes classes.
- Analisando o gráfico do item (a), que tipo de fronteira de decisão seria necessária para separar essas classes (linear ou não-linear)?
- Plote um gráfico com número de épocas versus os erros de treinamento e validação.
- Plote a matriz de confusão.
- Plote uma figura com as fronteiras de decisão para a função hipótese do enunciado.
- Plote o gráfico com a curva característica de operação do receptor (ROC).
- Imprima as **métricas de classificação** utilizando a função **classification_report**.
- Encontre uma **função hipótese** que melhor separe os dados do exercício.
- Repita os itens (c) até (g), agora com a **função hipótese** que você encontrou no item (h).
- Qual a diferença na performance do classificador entre as duas funções hipóteses? (**Dica:** qual das 2 hipóteses confere ao classificador maior precisão de classificação?)

(**Dica:** utilize como base o notebook `ClassificationOfTwoLinearlySeparableClasses.ipynb`, nele você vai encontrar a implementação do algoritmo do **gradiente descendente em batelada com early-stopping**).

(**Dica:** não se esqueça de encontrar o melhor valor para o **passo de aprendizagem**).

- 4) Exercício sobre k-NN:** Detecção 16QAM com k-NN. Neste exercício você irá utilizar o classificador k-NN, para realizar a detecção de símbolos 16QAM. Os símbolos 16QAM são dados pelo trecho de código abaixo. O trecho de código 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 classificador (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

$$SER = 2 \left(1 - \frac{1}{\sqrt{M}}\right) \operatorname{erfc} \left(k \sqrt{\frac{E_s}{N_0}}\right) - \left(1 - \frac{2}{\sqrt{M}} + \frac{1}{M}\right) \operatorname{erfc} \left(k \sqrt{\frac{E_s}{N_0}}\right)^2,$$

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. Utilizando a classe **KNeighborsClassifier** do módulo **neighbors** da biblioteca **sklearn**, faça o seguinte

- A. Construa um classificador, utilizando o classificador k-NN, para realizar a detecção dos símbolos 16QAM.
 - a. Gere $N = 100000$ 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 $E_s/N_0 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]$.
- B. Apresente um gráfico comparando a SER simulada e a SER teórica versus os valores de E_s/N_0 definidos acima.
- C. Podemos dizer que a curva simulada se aproxima da curva teórica da SER?

(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.dsblog.com/2012/01/01/symbol-error-rate-16qam-64qam-256qam/>).

- 5) **Exercício sobre árvores de decisão utilizando a métrica ID3:** Neste exercício você criar uma árvore de decisões para prever se o senhor Jair pagará o empréstimo que ele está solicitando junto a um banco para montar uma indústria farmacêutica especializada na produção de hidroxiclороquina. Jair possui os seguintes atributos: **Possui casa própria? Não - Estado civil: Casado - Experiência de trabalho: 3**. Portanto, dado estes três atributos sobre o senhor Jair, e a árvore montada acima, deve-se emprestar ou não o dinheiro a ele?

OBS.: Todos os atributos são discretos, ou seja, assumem valores de um conjunto finito de valores. Por exemplo, o atributo experiência de trabalho assume apenas os seguintes valores: 0, 1, 2, 3, 4 e 5.

Possui casa própria?	Estado civil	Experiência de trabalho	Pagou?
----------------------	--------------	-------------------------	--------

		(0-5)	
Sim	Solteiro	3	Sim
Não	Casado	4	Sim
Não	Solteiro	5	Sim
Sim	Casado	4	Sim
Não	Divorciado	2	Não
Não	Casado	4	Sim
Sim	Divorciado	2	Sim
Não	Casado	3	Não
Não	Casado	4	Sim
Não	Casado	2	Não
Sim	Casado	2	Sim
Não	Solteiro	2	Sim
Não	Divorciado	3	Não
Não	Solteiro	3	Sim
Sim	Divorciado	3	Sim
Sim	Solteiro	2	Não
Sim	Casado	3	Sim

6) 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 $256 \times 256 \times 256 = 16.777.216$ 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 das 3 cores do RGB é representada por 1 byte (i.e., 8 bits) qual o tamanho aproximado da imagem original e de cada uma das 4 imagens quantizadas?

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>

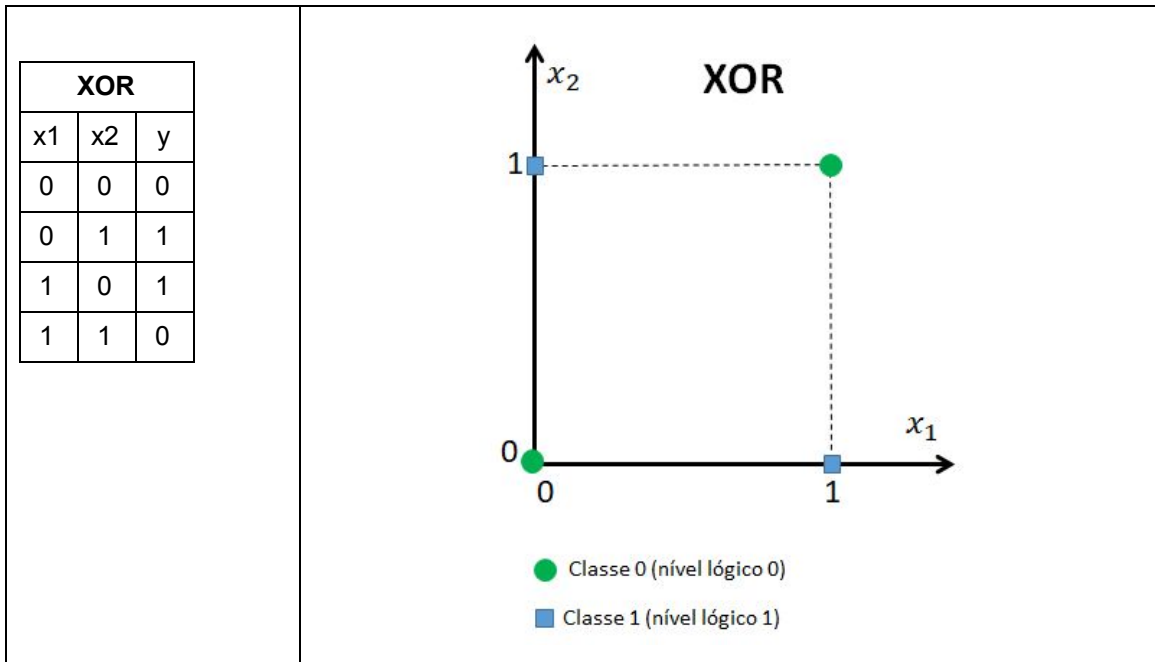
[3]

https://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html#sphx-gl-r-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/>

- 7) **Exercício sobre perceptrons:** Neste exercício você irá realizar a classificação da função lógica XOR com perceptrons. A tabela verdade e o gráfico com as classes dos elementos da tabela são mostrados abaixo. Como você deve se lembrar das listas de exercício, classificadores lineares, entre eles o perceptron, não conseguem separar classes que não são linearmente separáveis, ou seja, no caso abaixo, uma linha reta não conseguiria criar regiões que classifiquem os elementos corretamente.



Após ler a referência abaixo, faça o seguinte

- Implemente um **classificador** utilizando um conjunto de perceptrons que separe (classifique) os dados da tabela acima com precisão de 100%. **Use o menor número possível de perceptrons** para implementar tal classificador com precisão de 100%.
- Plote uma figura mostrando as fronteiras de decisão.
- Plote a matriz de confusão.
- Plote a curva ROC.
- Baseado na curva ROC, qual a área sob a curva?

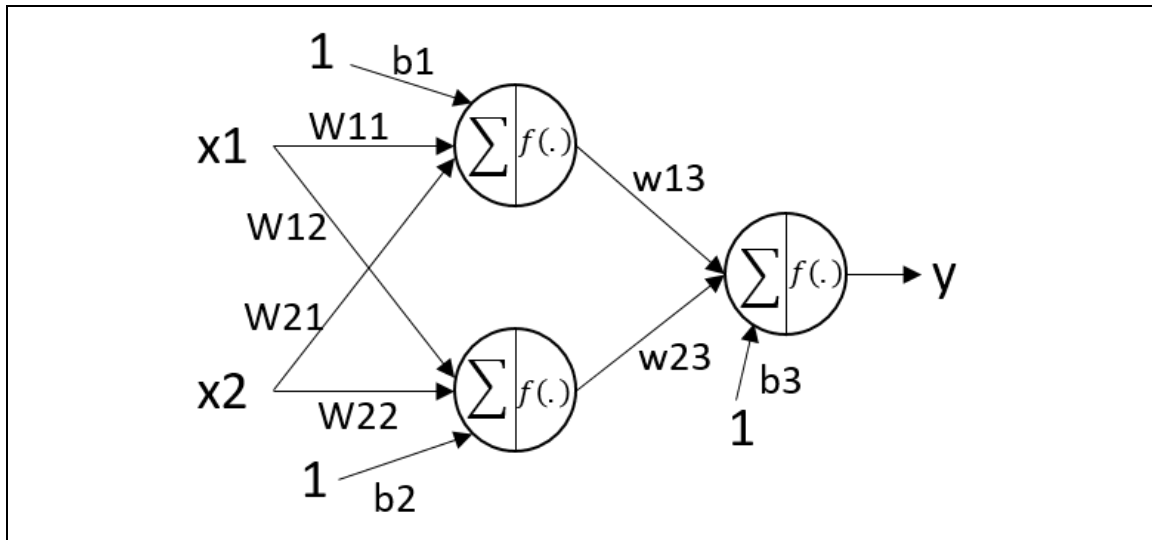
Referências

[1]

<https://medium.com/@jayeshbahire/the-xor-problem-in-neural-networks-50006411840b>

[2] http://home.agh.edu.pl/~vlsi/AI/xor_t/en/main.htm

- Exercício sobre multi layer perceptrons com TensorFlow:** Utilizando o TensorFlow, implemente um Multi Layer Perceptron (MLP) com 3 unidades (i.e., perceptrons), mostrada na figura abaixo, que classifique a função lógica XOR.



Se nós imaginarmos este classificador MLP da figura em forma matricial, então nós temos a seguinte equação:

$$y = f(f(W \cdot X + B) \cdot w + b3),$$

onde:

- X é o vetor de entrada com dimensão 2x1.
- W é uma matriz de coeficientes para a primeira camada da MLP, com dimensão 2x2.
- B é um vetor de bias para a primeira camada, com dimensão 2x1.
- w é um vetor de coeficientes para a segunda camada da MLP, com dimensão 2x1.
- b3 é um escalar de bias para a segunda camada, com dimensão, 1x1.
- y é o valor de saída da MLP.
- f(.) é a função de ativação sigmóide.

Use 40000 épocas e imprima o erro a cada 100 épocas, em seguida, faça o seguinte:

- a) Crie um grafo utilizando o **GradientDescentOptimizer**
(**Dica:** verifique a documentação deste otimizador no seguinte link: https://www.tensorflow.org/api_docs/python/tf/train/GradientDescentOptimizer)
- b) Treine o modelo.
- c) Crie um segundo grafo utilizando o **AdamOptimizer**. Não se esqueça de resetar o grafo com **tf.reset_default_graph()**.
(**Dica:** verifique a documentação deste otimizador no seguinte link: https://www.tensorflow.org/api_docs/python/tf/train/AdamOptimizer)
- d) Treine o modelo.
- e) Qual dos 2 otimizadores tem melhor performance?