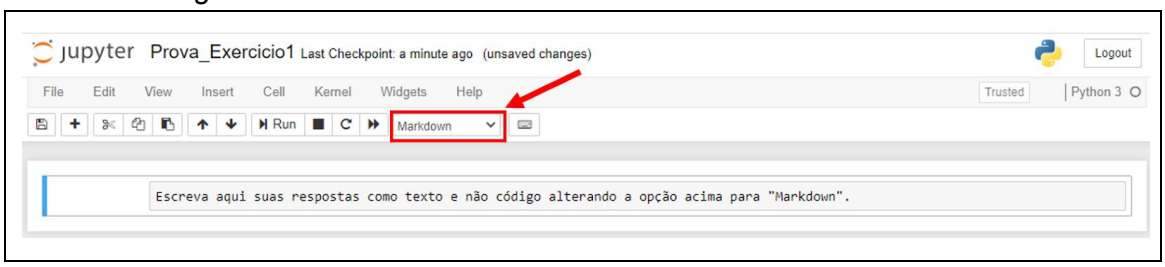
TP555 - AI/ML

Prova #2 (2S2020)

Orientações

- Crie um notebook do Jupyter diferente 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.



- Crie uma pasta chamada **prova2** em seu repositório do github e coloque todos os notebooks lá. Não se esqueça de comitar as resoluções ao final da prova.
- Desenhos ou equações podem ser feitos à mão ou qualquer outro SW de sua preferência. Coloque na pasta **prova2**, devidamente identificadas, as figuras, fotos ou pdfs dos desenhos ou equações.
- Atente-se ao prazo de entrega definido na tarefa do MS Teams.
    - O código deve ser comitado antes do prazo. Lembre-se que o github armazena o horário dos commits.
- A entrega da tarefa no MS Teams deve ser feita colando-se o link do repositório de vocês na tarefa, da mesma forma como está sendo feito coms as listas de exercícios.
- A prova deve ser resolvida **individualmente**.
- Todas as questões tem o mesmo peso.
- A interpretação faz parte da prova.
- Boa sorte!

Questões

1) **Exercício sobre classificação naive bayes**: Considere um conjunto de documentos, D, onde cada um deles pertence a classe **Esportes** (S) ou **Informática** (I). Dado um conjunto de treinamento com 11 documentos, crie 2 classificadores, um Multinomial Naive Bayes (NB) e outro Bernoulli NB, para classificar documentos não rotulados como S ou I. Para este exercício, define-se o seguinte vocabulário com 8 palavras:

$$V = \begin{bmatrix} w_1 = \text{goal} \\ w_2 = \text{tutor} \\ w_3 = \text{variance} \\ w_4 = \text{speed} \\ w_5 = \text{drink} \\ w_6 = \text{defence} \\ w_7 = \text{performance} \\ w_8 = \text{field} \end{bmatrix}$$

A. Para o classificador **MultinomialNB**, cada documento, d_i, é representado como um vetor de 8 dimensões (ou elementos), onde cada elemento define a frequência de cada palavra do alfabeto V no documento d_i. Os dados de treinamento são apresentados a seguir como uma matriz para cada uma das 2 classes, onde cada linha representa um vetor do documento.

$M^{\text{Esporte}} = \begin{bmatrix} 2 & 0 & 0 & 0 & 1 & 2 & 3 & 1 \\ 0 & 0 & 1 & 0 & 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 2 & 1 & 0 \\ 1 & 0 & 0 & 2 & 0 & 1 & 0 & 1 \\ 2 & 0 & 0 & 0 & 1 & 0 & 1 & 3 \\ 0 & 0 & 1 & 2 & 0 & 0 & 2 & 1 \end{bmatrix}$	$M^{\text{Informática}} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 2 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$

Agora, de posse dessas informações faça o seguinte:

a. Classifique **manualmente** os documentos de teste a seguir em **Esporte** ou **Informática**. Apresente todos os cálculos necessários para se decidir a classe dos 2 documentos de teste.
   i. $d_1 = [2,\ 1,\ 0,\ 0,\ 1,\ 2,\ 0,\ 1]$
   ii. $d_2 = [0,\ 1,\ 1,\ 0,\ 1,\ 0,\ 1,\ 0]$
b. Utilizando a classe **MultinomialNB** do scikit-learn, instancie e treine um classificador para decidir a classe dos documentos de teste. **OBS**.: Não é necessário utilizar a classe **CountVectorizer**, apenas junte as 2 matrizes de treinamento e as passe para o classificador juntamente com os labels, **'Esporte'** e '**Informatica**', correspondentes a cada documento.
c. Utilize o método ***predict*** da classe **MultinomialNB** e preveja a qual classe os documentos de teste pertencem.
d. Utilize o método **predict_proba** da classe **MultinomialNB** para imprimir as probabilidades encontradas e confira se elas são iguais às que você encontrou manualmente no item (A.a).

B. Para o classificador **BernoulliNB**, cada documento, d_i, é representado como um vetor binário de 8 dimensões, onde cada elemento define a presença ou ausência de cada palavra do alfabeto V no documento d_i. Os dados de treinamento são apresentados a seguir como uma matriz para cada uma das 2 classes, onde cada linha representa um vetor do documento.

$M^{\text{Esporte}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$	$M^{\text{Informática}} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$

Agora, de posse dessas informações faça o seguinte:

a. Classifique **manualmente** os documentos de teste a seguir em **Esporte** ou **Informática**. Apresente todos os cálculos necessários para se decidir a classe dos 2 documentos de teste.
   i. $d_1 = [1,\ 0,\ 0,\ 1,\ 1,\ 1,\ 0,\ 1]$
   ii. $d_2 = [0,\ 1,\ 1,\ 0,\ 1,\ 0,\ 1,\ 0]$
b. Utilizando a classe **BernoulliNB** do scikit-learn, instancie e treine um classificador para decidir a classe dos documentos de teste. **OBS**.: Não é necessário utilizar a classe **CountVectorizer**, apenas junte as 2 matrizes de treinamento e as passe para o classificador juntamente com os labels, **'Esporte'** e '**Informatica**', correspondentes a cada documento.
c. Utilize o método ***predict*** da classe **BernoulliNB** e preveja a qual classe os documentos de teste pertencem.
d. Utilize o método **predict_proba** da classe **BernoulliNB** para imprimir as probabilidades encontradas e confira se elas são iguais às que você encontrou manualmente no item (B.a).

OBSERVAÇÃO: Utilizem a ***suavização de Laplace***, também conhecida como ***suavização adicione 1*** [1,2], quando vocês forem calcular as probabilidades condicionais. A suavização é diferente para os 2 classificadores estudados, i.e., MultinomialNB e BernoulliNB.

- **Suavização de Laplace para o caso do MultinomialNB**: Com a suavização as probabilidades condicionais se tornam

$$P(w_k \mid C_q) = \frac{contagem(w_k, C_q) + 1}{\sum_{l=1}^{K} contagem(w_l, C_q) + |V|},$$

onde $contagem(w_k, C_q)$ é número de vezes que a palavra w_k aparece entre todas as palavras que pertencem à classe C_q, $\sum_{l=1}^{K} contagem(w_l, C_q)$ é a soma total de palavras pertencentes à classe C_q e $|V|$ é o tamanho do vocabulário, ou seja, o número de atributos.

- **Suavização de Laplace para o caso do BernoulliNB**: Para o caso do classificador **BernoulliNB**, utilizando-se a ***suavização de Laplace***, $P(w_k \mid C_q)$ é calculada como

$$P(w_k \mid C_q) = \frac{M_{t,C_q} + 1}{M_C + 2},$$

onde M_{t,C_q} é o número de mensagens de treinamento da classe C_q que contém o termo/palavra w_k, enquanto M_C é o número total de mensagens de treinamento da categoria C_q e 2 pois existem dois casos a serem considerados para cada termo/palavra, i.e., ocorrência e não ocorrência.

Referências:

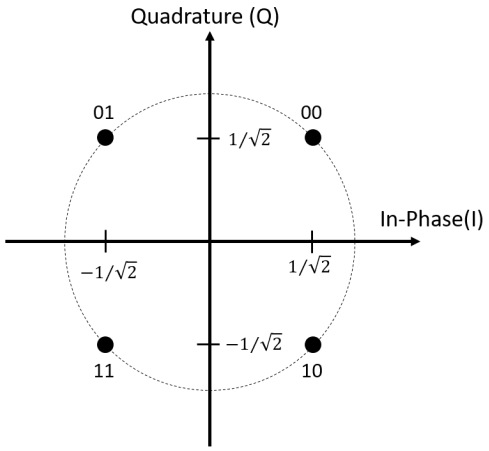
[1] 'An Introduction to Naïve Bayes Classifier', https://towardsdatascience.com/introduction-to-na%C3%AFve-bayes-classifier-fa59e3e24aaf

[2] 'Additive smoothing', https://en.wikipedia.org/wiki/Additive_smoothing

2) **Exercício sobre regressão softmax**: Neste exercício, você deve utilizar sua **implementação** (não use a biblioteca Scikit-Learn) do classificador ***Softmax*** utilizando o algoritmo do **gradiente descendente em batelada com early-stopping** para classificar os dados pertencentes a 3 classes diferentes. Utilize o conjunto de dados contido no arquivo threeConcentricCalsses.csv (baixe o arquivo e utilize seu conteúdo) para realizar o treinamento e a validação do classificador. Em seguida, pede-se
   A. Plote uma figura, utilizando todos os exemplos do arquivo threeConcentricCalsses.csv, indicando a qual classe cada exemplo pertence.
   B. Divida o conjunto de dados em 75% para treinamento e 25% para validação.
   C. Treine seu **regressor Softmax**. Utilize ao menos 50000 épocas e não se esqueça de encontrar o **passo de aprendizagem** ótimo.
   D. Plote um gráfico com número de épocas versus os erros de treinamento e validação.
   E. Plote a **matriz de confusão**.
   F. Plote uma figura mostrando as **fronteiras de decisão**.
   G. Imprima as **métricas de classificação** utilizando a função **classification_report**.

   (**Dica**: Lembre-se que os vetores de rótulos dos conjunto de treinamento e validação, y_train e y_test, devem ser convertidos para a representação **one-hot encoding)**.

3) **Exercício sobre k-NN**: Neste exercício, você irá utilizar o algoritmo do k-NN para classificar 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 *classificador* (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 **KNeighborsClassifier** do módulo **neighbours** 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**: 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/).

4) **Exercício sobre árvores de decisão utilizando a métrica ID3**: Considere o conjunto de treinamento para classificação de mamíferos dado na tabela abaixo.

Name	Body Temperature	Gives Birth?	Four-legged?	Hibernates?	Mammal?
porcupine	warm-blooded	yes	yes	yes	yes
cat	warm-blooded	yes	yes	no	yes
bat	warm-blooded	yes	no	yes	yes
whale	warm-blooded	yes	no	no	yes
salamander	cold-blooded	no	yes	yes	no
komodo dragon	cold-blooded	no	yes	no	no
python	cold-blooded	no	no	yes	no
salmon	cold-blooded	no	no	no	no
eagle	warm-blooded	no	no	no	no
guppy	cold-blooded	yes	no	no	no

Ele é composto por 4 atributos de entrada (**Body Temperature**, **Gives Birth?**, **Four-legged?** e **Hibernates?**) e uma saída binária, **Mammal?**. Usando o método ID3, encontre **manualmente** uma árvore de decisão para este conjunto de dados. Apresente todos os cálculos feitos para se determinar cada um dos nós da árvore. **OBS**.: A coluna **Name** da tabela não deve ser considerada para encontrar a árvore. Em seguida, de posse da árvore de decisão, classifique os exemplos de teste da tabela abaixo. A classificação feita pela árvore de decisão coincide com as classes da tabela?

Name	Body Temperature	Gives Birth?	Four-legged?	Hibernates?	Mammal?
human	warm-blooded	yes	no	no	yes
pigeon	warm-blooded	no	no	no	no
elephant	warm-blooded	yes	yes	no	yes
turtle	cold-blooded	no	yes	no	no
penguin	warm-blooded	no	no	no	no
dolphin	warm-blooded	yes	no	no	yes
platypus	warm-blooded	no	yes	no	yes
spiny anteater	warm-blooded	no	yes	yes	yes

Houve algum erro de classificação no conjunto de teste? Se sim, o que poderia ser feito para melhorar a acurácia do classificador.

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

A. 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].
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 (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)
```

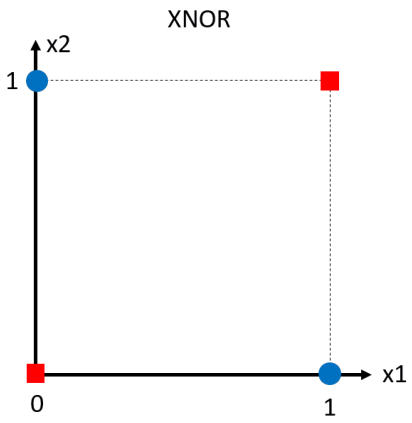
Para mais informações, leia a documentação da função ***train_test_split***: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)

(**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) **Exercício sobre multi-layer perceptrons**: Neste exercício você irá realizar a classificação da função lógica XNOR com uma rede multi-layer perceptron (MLP). A tabela verdade e o gráfico com as classes dos elementos da tabela são apresentados 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 classificariam os exemplos corretamente.

XNOR		
x1	x2	y
0	0	1
0	1	0
1	0	0
1	1	1



Após ler as referências abaixo, faça o seguinte

A. Treine um ***classificador*** utilizando um ***conjunto de perceptrons*** que separe (classifique) os dados da tabela acima com precisão de 100%. **Seu classificador DEVE usar o menor número possível de perceptrons** (ou seja, de nós/neurônios) para implementar tal classificador com precisão de 100%.
(**Dica**: Utilize a classe MLPClassifier disponibilizada pela biblioteca SciKit-Learn.)
(**Dica**: Lembre-se que devido a superfície de erro da rede não ser convexa, a convergência irá depender bastante da inicialização dos vetores de peso e bias, portanto, treine com vários valores de semente até que o classificador atinja precisão de 100%.)

B. Qual é o **menor número possível de nós** necessários para se separar essas classes?

C. Explique de forma sucinta como você chegou ao valor do menor número de nós (ou neurônios).

D. Use GridSearch (i.e., a classe GridSearchCV da biblioteca SciKit-Learn) variando os parâmetros **hidden_layer_sizes** e **random_state** para verificar qual é o valor ótimo para o número de nós escondidos.
(**Dica**: Faça com que o GridSearch teste com o parâmetro **hidden_layer_sizes** variando entre 0 e 15 e o parâmetro **random_state** variando entre 0 e 20.)

E. Plote uma figura mostrando as fronteiras de decisão.

F. Plote a matriz de confusão.

G. Plote a curva ROC.

H. Baseado na curva ROC, qual a área sob a curva?

Referências

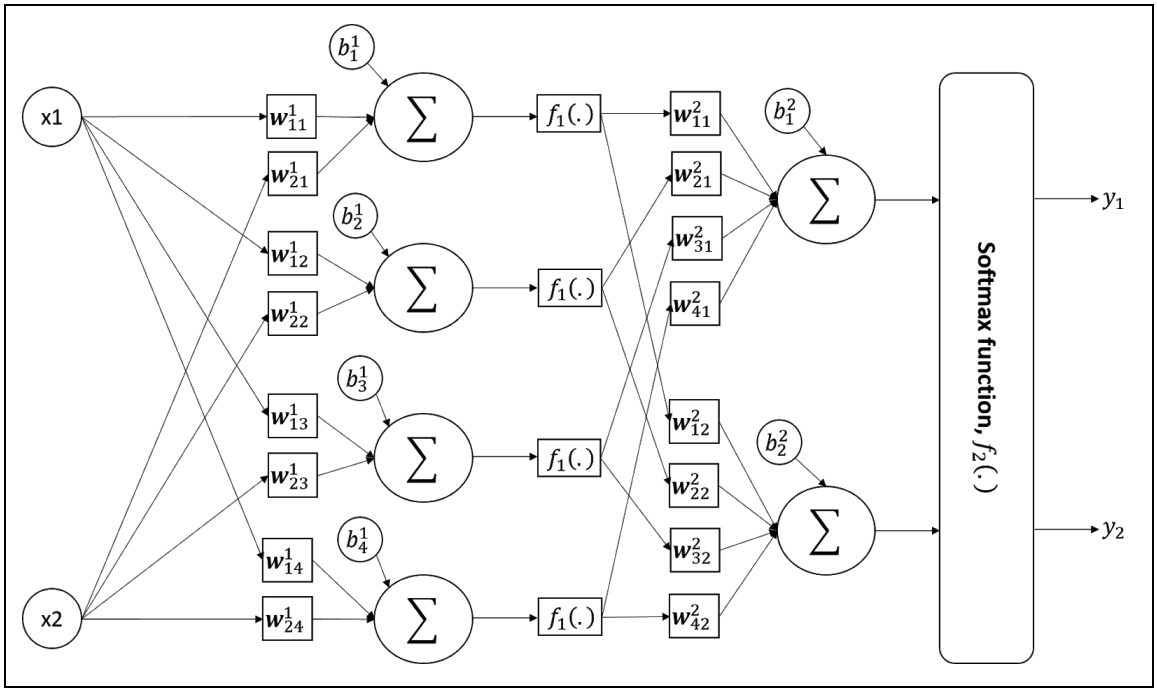
[1] https://medium.com/@jayeshbahire/the-xor-problem-in-neural-networks-50006411840b

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

[3] Neural network models (supervised), https://scikit-learn.org/stable/modules/neural_networks_supervised.html?highlight=mlp

[4] Perceptron, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html

7) **Exercício sobre TensorFlow**: Utilizando o TensorFlow, implemente o modelo mostrado na figura abaixo, para classificar 2 classes diferentes.



Se nós modelarmos o classificador da figura em formato matricial, então, nós temos a seguinte equação:

$$y = f_2\left(f_1\left(XW^1 + b^1\right)W^2 + b^2\right)$$

onde:

- X é a matriz de entrada com dimensão Nx2.
- W^1 é a matriz de coeficientes da primeira camada escondida com dimensão 2x4.

- W^2 é a matriz de coeficientes da camada de saída com dimensão 4x2.
- b^1 é o vetor de bias para a primeira camada escondida com dimensão 4.
- b^2 é o vetor de bias para a camada de saída com dimensão 2.
- y é o vetor de saída do classificador com dimensão 2.
- $f_1(.)$ é a função logística ou sigmoid.
- $f_2(.)$ é a função softmax.
- N é o número de exemplos.

Crie as 2 classes com o código abaixo.

```
from sklearn.datasets import make_circles

seed = 21

random.seed(seed)

N = 1000

X_, y_ = make_circles(n_samples=N, random_state=42, noise=0.1, factor=0.2)
```

Em seguida, faça o seguinte:

a) Plote um gráfico mostrando as diferentes classes. (**Dica**: use cores ou marcadores diferentes para cada classe.)
b) Crie um grafo utilizando o **GradientDescentOptimizer** para classificar os dados. (**Dica**: verifique a documentação deste otimizador no seguinte link: https://www.tensorflow.org/api_docs/python/tf/compat/v1/train/GradientDescentOptimizer)
c) Imprima e salve o erro (i.e., loss) e a precisão a cada 100 épocas.
d) Treine o modelo. (**Dica**: Use 20000 épocas ou treine até que a precisão seja de 100%.)
e) Plote um gráfico mostrando o erro e a precisão versus o número de épocas.
f) Plote um gráfico com as fronteiras de decisão.
g) Crie um segundo grafo utilizando o **AdamOptimizer** para classificar os dados. (**Dica**: 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/compat/v1/train/AdamOptimizer)
h) Imprima e salve o erro (i.e., loss) e a precisão a cada 100 épocas.
i) Treine o modelo. (**Dica**: Use 20000 épocas ou treine até que a precisão seja de 100%.)
j) Plote um gráfico mostrando o erro e a precisão versus o número de épocas.
k) Plote um gráfico com as fronteiras de decisão.
l) Baseado nos valores de erro e precisão impressos e plotados nas figuras anteriores, qual dos 2 otimizadores tem melhor performance? (**Dica**: Qual dos 2 converge mais rapidamente?)

OBSERVAÇÃO: Neste exercício, você vai precisar utilizar a função **tf.nn.sparse_softmax_cross_entropy_with_logits()** para calcular o erro do modelo e assim, conseguir treiná-lo, ou seja, encontrar os pesos que minimizem o erro. A função **tf.nn.sparse_softmax_cross_entropy_with_logits()** é equivalente a aplicar a função de ativação softmax e, em seguida, calcular a entropia cruzada, mas é mais eficiente e cuida adequadamente de casos incomuns, por exemplo, onde os logits são iguais a 0. A documentação dessa função pode ser encontrada em:
https://www.tensorflow.org/api_docs/python/tf/nn/sparse_softmax_cross_entropy_with_logits
Seguem alguns exemplos de como utilizar a função:
[1] https://towardsdatascience.com/active-learning-on-mnist-saving-on-labeling-f3971994c7ba
[2] https://riptutorial.com/tensorflow/example/17628/computing-costs-on-a-softmax-output-layer