TP555 - AI/ML

Lista de Exercícios #6

k-NN

1. Neste exercício você irá calcular as distâncias entre as amostras no conjunto de treinamento e a amostra de validação para encontrar, dependendo do valor do hiperparâmetro k do algoritmo k-NN, a qual classe a amostra de validação pertence. Use a norma Euclidiana (p = 2 na distância de Minkowski) para calcular a distância entre os pontos do conjunto de treinamento e a amostra de validação. Em seguida, encontre a qual classe a amostra de validação pertence quando k = 3 e 5, respectivamente. Após o cálculo das distâncias, use os métodos ***predict*** e ***kneighbors***, da classe ***KNeighborsClassifier*** para conferir os resultados que você encontrou.
   (**Dica**: a documentação da classe ***KNeighborsClassifier*** pode ser encontrada neste link: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html)

Conjunto de Treinamento

x1	x2	y
4	7	0
5	6	0
3	4	0
6	9	0
6	4	1
7	6	1
8	0	1
10	10	1
12	3	1

Amostra de Validação

x1	x2	y
6	5	?

2. Crie um classificador para o conjunto de dados de dígitos escritos à mão da biblioteca SciKit-Learn que atinja mais de 95% de precisão no conjunto de validação. Em seguida:
   a. Imprima a precisão atingida pelo classificador.
   b. Plote a matriz de confusão.
   c. Imprima as principais métricas de classificação com a função ***classification_report***.

(**Dica**: a classe ***KNeighborsClassifier*** funciona muito bem para esta tarefa, você só precisa encontrar bons valores para os hiperparâmetros ***weights*** e ***n_neighbors*** da classe ***KNeighborsClassifier***).
(**Dica**: a documentação da classe ***KNeighborsClassifier*** pode ser encontrada em: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html)
(**Dica**: utilize o ***GridSearchCV*** da biblioteca SciKit-Learn para encontrar os hiperparâmetros ***weights*** e ***n_neighbors*** que otimizam a performance do classificador k-NN. Utilize os valores 'uniform' e 'distance' para o hiperparâmetro ***weights*** e os valores 1, 2, 3, 4, 5, 10, 15, e 20 para o hiperparâmetro ***n_neighbors***. O ***GridSearchCV*** pode demorar cerca de 1 hora para encontrar o conjunto ótimo de hiperparâmetros dependendo do hardware que você tem.).

Grid Search

Uma maneira de ajustar os hiperparâmetros de um algoritmo de ML seria ajustá-los manualmente, até encontrar uma combinação ótima de valores. Entretanto, isso seria um trabalho muito tedioso e talvez você não tenha tempo para explorar muitas combinações. Em vez disso, você pode utilizar o ***GridSearchCV*** da biblioteca Scikit-Learn para que ele faça a busca por você. Tudo o que você precisa fazer é dizer com quais hiperparâmetros você deseja experimentar e quais valores testar, e o ***GridSearchCV*** avaliará todas as combinações possíveis de valores de hiperparâmetros, usando validação cruzada. Abaixo segue um exemplo de como utilizar o ***GridSearchCV*** (note que o código abaixo é apenas um exemplo, você não deve utilizá-lo "as-is" no exercício pois ele usa outro tipo de algoritmo de classificação)***.*** A documentação do ***GridSearchCV*** pode ser encontrada em https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

```
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
iris = datasets.load_iris()
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
svc = svm.SVC()
clf = GridSearchCV(svc, parameters, cv=5, verbose=3, n_jobs=-1)
clf.fit(iris.data, iris.target)
clf.best_params_
clf.best_score_
```

Exemplo de código-fonte para leitura da base de dados

```
# Import all necessary libraries.
import numpy as np
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
```

```
# Load the digits dataset.
digits = load_digits()

# Plot some digits from the data-set.
plt.figure(figsize=(20, 5))
for i in range(0,10):
    ax = plt.subplot(1, 10, i+1)
    plt.imshow(digits.images[i], cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Training: %i' % digits.target[i])
plt.show()

# In order to apply a classifier on this data, we need to flatten the image, to turn the
data in a (samples, feature) matrix:
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

# Apply GridSearch to the whole dataset.
-------> ADD YOUR CODE HERE

# Split data into train and test subsets.
x_train, x_test, y_train, y_test = train_test_split(data, digits.target, test_size=0.2,
random_state=42)

# Train a new KNeighborsClassifier with the optimum hyperparameters on the training
dataset, which has been created above.
-------> ADD YOUR CODE HERE

# Perform prediction with test dataset.
-------> ADD YOUR CODE HERE

# Show performance metrics below (score, confusion matrix, classification report).
-------> ADD YOUR CODE HERE
```

3. Neste exercício você vai comparar a performance dos classificadores: GaussianNB, Logistic Regression e k-NN. Utilize o código abaixo para criar amostras pertencentes a 2 classes. As amostras serão divididas em 2 conjuntos, um para treinamento e outro para validação. Apenas para o caso do classificador k-NN, utilize ***grid search*** para encontrar os valores ótimos para os hiperparâmetros ***weights*** e ***n_neighbors***. Utilize os valores 'uniform' e 'distance' para o hiperparâmetro ***weights*** e os valores 1, 2, 3, 4, 5, 10, 15, e 20 para o hiperparâmetro ***n_neighbors***. Plote um único gráfico comparando a curva ROC e a área sob a curva de cada um dos classificadores. Analisando as curvas ROC e os valores das áreas sob as curvas, qual classificador apresenta a melhor performance?

```
# Import all necessary libraries.
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
```

```
# generate 2 class dataset
x, y = make_classification(n_samples=10000, n_classes=2, weights=[0.9,0.5],
random_state=42)

# Split data into train and test subsets.
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.8)
```

4. Utilize ***grid search*** para encontrar os hiperparâmetros ótimos da ***regressão*** com k-NN com o seguinte conjunto de dados:

```
# Import all necessary libraries.
import numpy as np
from sklearn.model_selection import train_test_split

N = 1000
np.random.seed(42)
x = np.sort(5 * np.random.rand(N, 1), axis=0)
y = np.sin(x).ravel()
y_orig = np.sin(x).ravel()

# Add noise to targets.
y += 0.1*np.random.randn(N)

# Split data into train and test subsets.
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

Utilize os seguintes hiperparâmetros com o ***GridSearch***:

```
# Set parameters for grid-search.
param_grid = [{'weights': ['uniform', 'distance'], 'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15,
20]}]
```

Em seguida faça o seguinte:

A. Plote um gráfico que mostre os dados originais, os ruidosos e a aproximação encontrada com os parâmetros ótimos do k-NN regressor.
B. Qual o erro quadrático médio (MSE) para o conjunto de validação/teste?

5. **Exercício sobre k-NN**: Neste exercício, você irá utilizar o algoritmo do k-NN para classificar os dados da modulação digital BPSK, ou seja, realizar a detecção de símbolos BPSK. Os símbolos BPSK são dados na tabela abaixo.

bits	**Símbolo (I + jQ)**
0	-1
1	$+1$

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

$$\text{SER} = 0.5erfc\left(\sqrt{\frac{Es}{N0}}\right).$$

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

A. Construa um detector para realizar a detecção dos símbolos BPSK.
    a. Gere N = 1000000 símbolos BPSK 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 = [-10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10,12].
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*).

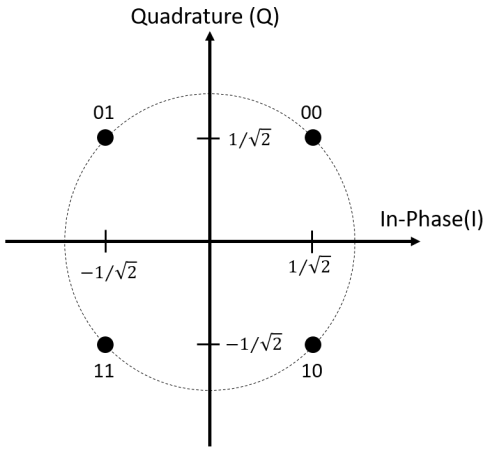
(**Dica**: A função ***train_test_split*** pode dividir qualquer número de vetores de entrada em vetores de treinamento e teste. Veja o exemplo abaixo onde três vetores de entrada, a, e c, são divididos em vetores de treinamento e teste.

```
# 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/).

6. **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. Use Grid Search para encontrar o valor ótimo do parâmetro k, i.e., o número de vizinhos.
B. 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].
C. Apresente um gráfico comparando a SER simulada e a SER teórica versus os valores de Es/N0 definidos acima.
D. 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/).

7. **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

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

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 ***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 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**: 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/).

8. Um outro tipo de tarefa de classificação que discutiremos aqui neste exercício é chamada de classificação com múltiplas saídas e múltiplas classes (ou simplesmente classificação *multioutput*). Ela é uma generalização da classificação *multilabel* onde cada rótulo pode ser de múltiplas classes (ou seja, pode ter mais de dois valores possíveis). Para entendermos este tipo de classificação, você irá criar um modelo que remove ruído de imagens. O modelo terá como entrada uma imagem de um dígito ruidoso e produzirá uma imagem limpa, representada como uma matriz de intensidades de pixel, assim como as imagens da base de dados MNIST. Observe que a saída do classificador é *multilabel* (ou seja, um rótulo por pixel) e cada rótulo pode ter vários valores (intervalos de intensidade de pixel de 0 a 255). É, portanto, um exemplo de sistema de classificação de múltiplas saídas.

   IMPORTANTE: A linha de separação entre tarefas de classificação e de regressão às vezes fica embaçada, como neste exemplo. Indiscutivelmente, a previsão da intensidade do pixel é mais semelhante à regressão do que à classificação. Além disso, os sistemas de múltiplas saídas não se limitam a tarefas de classificação; você pode até ter um sistema que produz vários rótulos por instância, incluindo rótulos de classe e rótulos de valor.

   Agora faça o seguinte:

   1. Carregue a base de dados dos dígitos escritos à mão. Use o trecho de código abaixo.

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

   2. Faça o reshape dos conjuntos de treinamento e validação para que eles tenham 784 atributos, ou seja, 784 colunas.
   3. Treine um classificador k-NN (***KNeighborsClassifier***) com as imagens limpas. Use *grid search* para encontrar os melhores valores para os parâmetros ***n_neighbors*** e ***weights***.
   4. Imprima a acurácia e plote a matriz de confusão com o conjunto de validação.
   5. Adicione ruído às imagens dos dois conjuntos (treinamento e validação). Use o trecho de código abaixo:

```
noise = np.random.randint(0, 100, (len(X_train), 784))
X_train_mod = X_train + noise

noise = np.random.randint(0, 100, (len(X_test), 784))
X_test_mod = X_test + noise

y_train_mod = X_train
y_test_mod = X_test
```

   6. Usando o modelo treinado com as imagens limpas, calcule a acurácia e plote a matriz de confusão com o conjunto de validação ruidoso.

7. Treine um **novo** classificador k-NN (***KNeighborsClassifier***), agora para remover o ruído das imagens, i.e., *denoise*. Para treinar este modelo, use as imagens ruidosas como entrada do modelo e as imagens originais como os rótulos. Use *grid search* para encontrar os melhores valores para os parâmetros ***n_neighbors*** e ***weights***.
8. Quantas **saídas** tem esse modelo de *denoising*?
9. Quantos valores diferentes um **rótulo** pode ter?
10. Use o modelo de *denoising* para limpar todas as imagens do conjunto de validação ruidoso e, em seguida, use as imagens limpas pelo modelo de *denoise* como entrada do modelo de classificação treinado com as imagens limpas.
11. Imprima a acurácia e plote a matriz de confusão com este conjunto limpo.
12. A partir dos resultados obtidos, o que você pode concluir? O *denoise* foi eficiente?

9. **Exercício sobre kNN**: Neste exercício, você irá utilizar o algoritmo dos ***k-Vizinhos mais próximos*** para classificar a base de dados dos dígitos escritos à mão. Em seguida, você utilizará Análise de Componentes Principais (do inglês, *Principal Component Analysis*, PCA) para reduzir o número de atributos e com isso diminuir o tempo de treinamento e possivelmente aumentar a performance do classificador.

Extração de atributos para redução de dimensionalidade utilizando análise de componentes principais

Muitos problemas de aprendizado de máquina envolvem centenas, milhares ou até milhões de atributos para cada exemplo de treinamento. Isso não apenas torna o treinamento extremamente lento, mas também pode tornar muito mais difícil encontrar uma boa solução. Felizmente, em problemas do mundo real, muitas vezes é possível reduzir consideravelmente o número de atributos, transformando um problema intratável em um tratável.

A redução de dimensionalidade transforma dados de um espaço de alta dimensão em um espaço de menor dimensão de forma que a representação de menor dimensão retenha algumas propriedades significativas dos dados originais, idealmente próximas de sua dimensão original.

A redução da dimensionalidade faz com que algumas informações sejam perdidas. Desta forma, embora acelere o treinamento, também pode fazer com que o desempenho do modelo seja um pouco pior. Entretanto, em alguns casos, a redução de dimensionalidade dos dados de treinamento pode filtrar ruído e detalhes desnecessários e, assim, resultar em melhor desempenho, mas em geral, ela apenas irá acelerar o treinamento.

A Análise de Componentes Principais (em inglês, Principal Component Analysis - PCA) é o algoritmo de redução de dimensionalidade linear mais popular e o que utilizaremos neste exercício. Uma apresentação sobre redução da dimensionalidade pode ser encontrada no seguinte link: redução da dimensionalidade. Veja o seguinte notebook para entender como utilizar o algoritmo PCA implementado pela biblioteca SciKit-Learn: aplicando_pca_em_conjunto_3D.

Agora faça o seguinte:

1. Importe a base de dados com o código fornecido na célula abaixo. Este é um conjunto de dados com imagens em tons de cinza com 28x28 pixels representando 10 dígitos. A base de dados contém 60.000 imagens para treinamento e 20.000 para validação, entretanto, devido ao elevado tempo de treinamento este conjunto necessitaria, utilizaremos apenas 4000 para treinamento e 1000 para validação.

```
from tensorflow import keras

# Número de exemplos para treinamento.
Ntrain = 4000
# Número de exemplos para validação.
Ntest = 1000

# Baixa a base de dados.
(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()

# Reduz e altera as dimensões dos dados.
X_train = X_train[0:Ntrain,:,:].reshape(Ntrain, 28*28)
X_test = X_test[0:Ntest,:,:].reshape(Ntest, 28*28)
y_train = y_train[0:Ntrain].flatten()
y_test = y_test[0:Ntest].flatten()
```

2. Qual é o número de atributos desta base de dados? (**Dica**: Qual o número de colunas das matrizes de treinamento e validação?)
3. Usando o algoritmo k-NN, o qual é implementado pela classe ***KNeighborsClassifier*** da biblioteca SciKit-Learn, encontre o número ideal de vizinhos, ou seja, k, para o conjunto de dados de treinamento. O número de vizinhos, k, é configurado através do parâmetro ***n_neighbors*** da classe ***KNeighborsClassifier***. Teste ***n_neighbors*** variando de 1 até 15 vizinhos. Não altere o valor padrão do parâmetro ***weights*** da classe ***KNeighborsClassifier***, ou seja, deixe ***weights='uniform'***. Use um objeto da classe ***GridSearchCV*** para encontrar o número ideal de vizinhos, k. (**Dicas**: Faça os parâmetros ***param_grid***, ***cv***, e ***verbose*** da classe ***GridSearchCV*** iguais a [{'n_neighbors': range(1,16)}], 3 e 3, respectivamente. O parâmetro ***param_grid*** especifica quais valores devem ser testados, ***cv*** configura o número de divisões usadas com a

abordagem de validação cruzada do k-Fold e ***verbose*** controla o detalhamento: quanto mais alto, mais mensagens. O parâmetro ***verbose*** feito igual a 3 fará com que o tempo de treinamento seja impresso da seguinte forma: *Done 45 out of 45 | elapsed: 2.7min finished*, onde os 2.7 minutos indicam o tempo total de treinamento. A documentação da classe ***GridSearchCV*** pode ser encontrada em GridSearchCV).

4. Qual foi o tempo total de treinamento? (**Dica**: use o código abaixo para medir o tempo de treinamento).

```
import time

start = time.time()

# Adicione aqui, entre as duas chamada para a função time(), o código que você quer medir o tempo de execução.

end = time.time()
print(end - start)
```

5. Qual o número ideal de vizinhos, k, encontrado através do ***GridSearch***? (**Dica**: O valor ótimo para ***n_neighbors*** pode ser acessado através do atributo ***best_params_*** da classe ***GridSearchCV*** com o parâmetro ***n_neighbors*** como índice, ou seja, ***best_params_['n_neighbors']***. Veja a documentação: GridSearchCV).
6. Calcule e imprima a acurácia do modelo obtida com o conjunto de validação.
7. Plote a matriz de confusão deste classificador.
8. Imprima o reporte de classificação obtido através da função ***classification_report***.

Agora, usaremos análise de componentes principais (PCA) para reduzir o número de atributos do conjunto e com isso diminuir o tempo de treinamento e talvez melhorar o desempenho do modelo. Faça o seguinte:

9. Inicialmente, plote o gráfico com a ***curva da proporção de variância explicada*** para todos os atributos do conjunto de treinamento. Use o trecho de código abaixo.

```
# fit the PCA with the training data
pca = PCA(random_state=seed)

# Train the PCA preprocessing.
pca.fit(X_train)

# Plot the cumulative sum of eigenvalues
fig = plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Número de atributos', fontsize=14)
plt.ylabel('Variância (%)', fontsize=14) #for each component
```

```
plt.title('Variância da base de dados')
plt.grid()
left, bottom, width, height = [0.4, 0.4, 0.45, 0.3]
ax1 = fig.add_axes([left, bottom, width, height])
plt.plot(np.cumsum(pca.explained_variance_ratio_))
ax1.set_xlim(0, 100)
ax1.set_ylim(0.5, 1)
ax1.set_yticks([0.6, 0.9, 0.95, 1.0])
ax1.grid()
plt.show()
```

10. Observe a figura e responda aproximadamente quantos atributos seriam necessários para aproximar 90% da variância do conjunto de treinamento.
11. Escolha a quantidade de atributos que explique 90% da variância do conjunto de treinamento. Use o trecho de código abaixo.

```
pca = PCA(n_components=0.90)
pca.fit(X_train)

X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

print('Número de dimensões:', pca.n_components_)
print('Proporção de variância explicada com %d dimensões: %1.4f' % (pca.n_components_,
np.sum(pca.explained_variance_ratio_)))
```

12. Qual é o número de atributos que explica 90% da variância do conjunto de treinamento?
13. Usando o conjunto de treinamento com número reduzido de atributos, ou seja, X_train_pca, encontre o número de vizinhos ideal para a classificação dos dígitos escritos à mão. O número de vizinhos, k, é configurado através do parâmetro ***n_neighbors*** da classe ***KNeighborsClassifier***. Teste ***n_neighbors*** variando de 1 até 15 vizinhos. Use um objeto da classe ***GridSearchCV*** para encontrar o número ideal de vizinhos, k. (**Dicas**: Faça os parâmetros ***param_grid***, ***cv***, e ***verbose*** da classe ***GridSearchCV*** iguais a [{'n_neighbors': range(1,16)}], 3 e 3, respectivamente. O parâmetro ***param_grid*** especifica quais valores devem ser testados, ***cv*** configura o número de divisões usadas com a abordagem de validação cruzada do k-Fold e ***verbose*** controla o detalhamento: quanto mais alto, mais mensagens. O parâmetro ***verbose*** igual a 3 fará com que o tempo de treinamento seja impresso da seguinte forma: *Done 45 out of 45 | elapsed: 2.7min finished*, onde os 2.7 minutos indicam o tempo total de treinamento. A documentação da classe ***GridSearchCV*** pode ser encontrada em GridSearchCV).
14. Qual foi o tempo total de treinamento? Esse tempo foi menor do que o tempo de treinamento com o número original de atributos?

15. Qual o número ideal de vizinhos, k, encontrado através do ***GridSearch***? (**Dica**: O valor ótimo para ***n_neighbors*** pode ser acessado através do atributo ***best_params_*** da classe ***GridSearchCV*** com o parâmetro ***n_neighbors*** como índice, ou seja, ***best_params_['n_neighbors']***. Veja a documentação: GridSearchCV).
16. Calcule e imprima a acurácia do modelo obtida com o conjunto de validação.
17. Plote a matriz de confusão deste classificador.
18. Imprima o reporte de classificação obtido através da função ***classification_report***.
19. A acurácia do modelo com número menor de atributos é maior ou menor do que a acurácia do modelo treinado com o número original de atributos? Por que isso pode estar ocorrendo? Justifique sua resposta.

10. **Exercício sobre kNN**: Neste exercício, você irá utilizar o algoritmo dos ***k-Vizinhos mais próximos*** para classificar a base de dados que iremos gerar com a função *make_classification* da biblioteca SciKit-Learn. Em seguida, você utilizará Análise de Componentes Principais (do inglês, *Principal Component Analysis*, PCA) para reduzir o número de atributos e com isso diminuir o tempo de treinamento e possivelmente aumentar a performance do classificador.

    Extração de atributos para redução de dimensionalidade utilizando análise de componentes principais

    Muitos problemas de aprendizado de máquina envolvem centenas, milhares ou até milhões de atributos para cada exemplo de treinamento. Isso não apenas torna o treinamento extremamente lento, mas também pode tornar muito mais difícil encontrar uma boa solução. Felizmente, em problemas do mundo real, muitas vezes é possível reduzir consideravelmente o número de atributos, transformando um problema intratável em um tratável.

    A redução de dimensionalidade transforma dados de um espaço de alta dimensão em um espaço de menor dimensão de forma que a representação de menor dimensão retenha algumas propriedades significativas dos dados originais, idealmente próximas de sua dimensão original.

    A redução da dimensionalidade faz com que algumas informações sejam perdidas. Desta forma, embora acelere o treinamento, também pode fazer com que o desempenho do modelo seja um pouco pior. Entretanto, em alguns casos, a redução de dimensionalidade dos dados de treinamento pode filtrar ruído e detalhes desnecessários e, assim, resultar em melhor desempenho, mas em geral, ela apenas irá acelerar o treinamento.

A Análise de Componentes Principais (em inglês, Principal Component Analysis - PCA) é o algoritmo de redução de dimensionalidade linear mais popular e o que utilizaremos neste exercício. Uma apresentação sobre redução da dimensionalidade pode ser encontrada no seguinte link: redução da dimensionalidade. Veja o seguinte notebook para entender como utilizar o algoritmo PCA implementado pela biblioteca SciKit-Learn: aplicando_pca_em_conjunto_3D.

Agora faça o seguinte:

1. Crie a base de dados com o código fornecido na célula abaixo. A função ***make_classification*** da biblioteca SciKit-Learn é usada para gerar exemplos para um problema de classificação aleatória com M classes. No caso deste exercício, o número de classes, M, é igual a 4.
   (**Dica**: Note que a célula de código abaixo já divide o conjunto total de amostras em conjuntos de treinamento e validação, portanto não é necessário dividir o conjunto).

```
from sklearn.datasets import make_classification

seed = 42
np.random.seed(seed)

N = 5000

X, y = make_classification(n_samples=N, n_features=200, n_informative=4, n_redundant=80,
n_repeated=50, n_classes=4, n_clusters_per_class=1, random_state=seed)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=seed)
```

2. Qual é o número de atributos desta base de dados? (**Dica**: Qual o número de colunas das matrizes de treinamento e validação?)
3. Usando o algoritmo k-NN, o qual é implementado pela classe ***KNeighborsClassifier*** da biblioteca SciKit-Learn, encontre o número ideal de vizinhos, ou seja, k, para o conjunto de dados de treinamento. O número de vizinhos, k, é configurado através do parâmetro ***n_neighbors*** da classe ***KNeighborsClassifier***. Teste ***n_neighbors*** variando de 1 até 20 vizinhos. Use um objeto da classe ***GridSearchCV*** para encontrar o número ideal de vizinhos, k. (**Dicas**: Faça os parâmetros ***param_grid***, ***cv***, e ***verbose*** da classe ***GridSearchCV*** iguais a [{'n_neighbors': range(1,21)}], 5 e 3, respectivamente. O parâmetro ***param_grid*** especifica quais valores devem ser testados, ***cv*** configura o número de divisões usadas com a abordagem de validação cruzada do k-Fold e ***verbose*** controla o detalhamento: quanto mais alto, mais mensagens. O parâmetro ***verbose*** feito igual a 3 fará com que o tempo de treinamento seja impresso da seguinte forma: *Done 45 out of 45 | elapsed: 2.7min finished*, onde os 2.7 minutos indicam o tempo total de treinamento. A

documentação da classe ***GridSearchCV*** pode ser encontrada em GridSearchCV).

4. Qual foi o tempo total de treinamento? (**Dica**: use o código abaixo para medir o tempo de treinamento).

```
import time

start = time.time()

# Adicione aqui, entre as duas chamada para a função time(), o código que você quer medir o tempo de execução.

end = time.time()
print(end - start)
```

5. Qual o número ideal de vizinhos, k, encontrado através do ***GridSearch***? (**Dica**: O valor ótimo para ***n_neighbors*** pode ser acessado através do atributo ***best_params_*** da classe ***GridSearchCV*** com o parâmetro ***n_neighbors*** como índice, ou seja, ***best_params_['n_neighbors']***. Veja a documentação: GridSearchCV).
6. Calcule e imprima a acurácia do modelo obtida com o conjunto de validação.
7. Plote a matriz de confusão deste classificador.
8. Imprima o reporte de classificação obtido através da função ***classification_report***.

Agora, usaremos análise de componentes principais (PCA) para reduzir o número de atributos do conjunto e com isso diminuir o tempo de treinamento e talvez melhorar o desempenho do modelo. Faça o seguinte:

9. Inicialmente, plote o gráfico com a ***curva da proporção de variância explicada*** para todos os atributos do conjunto de treinamento. Use o trecho de código abaixo.

```
# fit the PCA with the training data
pca = PCA(random_state=seed)

# Train the PCA preprocessing.
pca.fit(X_train)

# Plot the cumulative sum of eigenvalues
fig = plt.figure()
plt.plot(range(1,X_train.shape[1]+1), np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Número de atributos', fontsize=14)
plt.ylabel('Variância (%)', fontsize=14) #for each component
plt.title('Variância da base de dados')
plt.grid()
plt.xlim([1, X_train.shape[1]])
left, bottom, width, height = [0.4, 0.4, 0.45, 0.3]
ax1 = fig.add_axes([left, bottom, width, height])
```

```
plt.plot(np.cumsum(pca.explained_variance_ratio_))
ax1.set_xlim(0, 80)
ax1.grid()
plt.show()
```

10. Observe a figura e responda aproximadamente quantos atributos seriam necessários para aproximar 90% da variância do conjunto de treinamento.
11. Escolha a quantidade de atributos que explique 90% da variância do conjunto de treinamento. Use o trecho de código abaixo.

```
pca = PCA(n_components=0.90, random_state=seed)
pca.fit(X_train)

X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

print('Número de dimensões:', pca.n_components_)
print('Proporção de variância explicada com %d dimensões: %1.4f' % (pca.n_components_,
np.sum(pca.explained_variance_ratio_)))
```

12. Qual é o número de atributos que explica 90% da variância do conjunto de treinamento?
13. Usando o conjunto de treinamento com número reduzido de atributos, ou seja, X_train_pca, encontre o número de vizinhos ideal para a classificação dos dígitos escritos à mão. O número de vizinhos, k, é configurado através do parâmetro ***n_neighbors*** da classe ***KNeighborsClassifier***. Teste ***n_neighbors*** variando de 1 até 20 vizinhos. Use um objeto da classe ***GridSearchCV*** para encontrar o número ideal de vizinhos, k. (**Dicas**: Faça os parâmetros ***param_grid***, ***cv***, e ***verbose*** da classe ***GridSearchCV*** iguais a [{'n_neighbors': range(1,21)}], 5 e 3, respectivamente. O parâmetro ***param_grid*** especifica quais valores devem ser testados, ***cv*** configura o número de divisões usadas com a abordagem de validação cruzada do k-Fold e ***verbose*** controla o detalhamento: quanto mais alto, mais mensagens. O parâmetro ***verbose*** feito igual a 3 fará com que o tempo de treinamento seja impresso da seguinte forma: *Done 45 out of 45 | elapsed: 2.7min finished*, onde os 2.7 minutos indicam o tempo total de treinamento. A documentação da classe ***GridSearchCV*** pode ser encontrada em GridSearchCV).
14. Qual foi o tempo total de treinamento? Esse tempo foi menor do que o tempo de treinamento com o número original de atributos?
15. Qual o número ideal de vizinhos, k, encontrado através do ***GridSearch***? (**Dica**: O valor ótimo para ***n_neighbors*** pode ser acessado através do atributo ***best_params_*** da classe ***GridSearchCV*** com o parâmetro ***n_neighbors*** como índice, ou seja, ***best_params_['n_neighbors']***. Veja a documentação: GridSearchCV).
16. Calcule e imprima a acurácia do modelo obtida com o conjunto de validação.

17. Plote a matriz de confusão deste classificador.
18. Imprima o reporte de classificação obtido através da função ***classification_report***.
19. A acurácia do modelo com número menor de atributos é maior ou menor do que a acurácia do modelo treinado com o número original de atributos? Por que isso pode estar ocorrendo? Justifique sua resposta.