

# 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			Amostra de Validação		
x1	x2	y	x1	x2	y
4	7	0	6	5	?
5	6	0			
3	4	0			
6	9	0			
6	4	1			
7	6	1			
8	0	1			
10	10	1			
12	3	1			

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](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

$$SER = 0.5 \operatorname{erfc}\left(\sqrt{\frac{E_s}{N_0}}\right).$$

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  $E_s/N_0 = [-10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10, 12]$ .
  - B. Apresente um gráfico comparando a SER simulada e a SER teórica versus os valores de  $E_s/N_0$  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: 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](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.dsblog.com/2007/11/06/symbol-error-rate-for-4-qam/>).

6. 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 **nov**o 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?