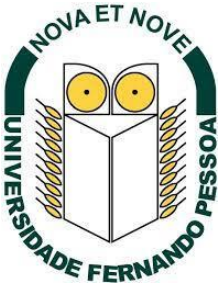

Relatório Final do Projeto

Carolina Araújo, 33348

Inteligência Artificial - 2020

Universidade Fernando Pessoa

índice

Introdução

Este projeto pretende, recorrendo aos algoritmos de aprendizagem máquina e data mining, prever as florestas com maior área queimada tendo em conta as condições meteorológicas e outros critérios de classificação na deteção de incêndios.

Pretende-se com a análise deste projeto que seja possível a adoção de sensores nas florestas para a monitorização das mesmas bem como prestar, posteriormente, através deles, medidas que ajudem a reduzir a probabilidade de incêndio nas áreas mais críticas (e menos críticas).

Motivação

A aplicação prática dos conceitos abordados neste documento permite que este sistema possa ser adotado, em condições reais, como um recurso auxiliar na prevenção e deteção de incêndios, em Portugal.

Objetivos

O objetivo deste projeto reside em prever as áreas com histórico de maior área queimada(isto é, se queimou ou não), tendo em conta as condições meteorologias e temporais associadas às suas localizações.

Estado da Arte

Os conhecimentos atuais na área de aprendizagem máquina e data mining [2], assim como as crescentes capacidades das ferramentas associadas, facilitam o desenvolvimento da solução proposta.

Desta forma, torna-se possível uma ligeira abstração do que consiste a implementação (código) e um maior foco no campo conceitual.

Alguns projetos listados em [3], [4] sugerem propostas de resolução de implementação e aplicação em muitos contextos, podendo ser aplicado parte dela no meu também.

Descrição do trabalho

Foi desenvolvido um script (ficheiro anexado como script.py) para modificar alguns dados provenientes do dataset original [1], um dataset público de estudo criado por Paulo Cortez e Aníbal Morais, da Universidade do Minho, em 2007, para uma conferência sobre Inteligência Artificial, em Portugal.

As mudanças mais importantes de salientar consiste na atribuição de valores inteiros às colunas “day” e “month” que originalmente se encontravam com valores de string.

Foi adicionado uma nova coluna de output, “burntarea” que representa se a área de floresta com as suas coordenadas associadas ardeu ou não, tendo associado dois valores booleanas 0 – não e 1 – sim.

```
# X,Y,month,day,FFMC,DMC,DC,ISI,temp,RH,wind,rain,burntarea
  #1 => 7,5,3,5,86.2,26.2,94.3,5.1,8.2,51,6.7,0.0,no
 #2 => 7,4,10,2,90.6,35.4,669.1,6.7,18.0,33,0.9,0.0,no
 #3 => 7,4,10,6,90.6,43.7,686.9,6.7,14.6,33,1.3,0.0,no
  #4 => 8,6,3,5,91.7,33.3,77.5,9.0,8.3,97,4.0,0.2,no
```

Para testar a credibilidade em Machine Learning é necessário treinar o modelo e testar o mesmo com um diferente conjunto, misturando-se ao efetuar a divisão de x de treino e x de teste, sendo um processo denominado de “Holdout”.

Levando essa lei em conta, efetuou-se a divisão do dataset com 400 entradas para os exemplos de treino (“train”) e as restantes 117 entradas para os exemplos de test (“test”).

```
train.shape
```

```
(400, 13)
```

```
test.shape
```

```
(117, 13)
```

Figura 1 *- Contagem das entradas para os exemplos de treino e teste.*

Numa fase seguinte, junta-se os exemplos de treino e teste numa lista única de forma a uniformizar os dados.

```
data_list = [train, test]
data = train.append(test, sort=False)
df = pd.DataFrame(data)
train.head(10)
```

Figura 2 *- Código relativo à junção dos exemplos numa lista única.*

Posteriormente, é efetuado um mapeamento dos valores associados à classe atributo, "burntarea", para valores booleanos.

```
burntareamapping = {'no' : 0, 'yes': 1}
train['burntarea'] = train['burntarea'].map(burntareamapping)
test['burntarea'] = test['burntarea'].map(burntareamapping)
train.head(20)
```

Figura 3 *- Código para o mapeamento dos valores em valores booleanos.*

Tendo, o seguinte output:

Out[40]:

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	burntarea
0	7	5	3	5	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	no
1	7	4	10	2	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	no
2	7	4	10	6	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	no
3	8	6	3	5	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	no
4	8	6	3	7	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	no
5	8	6	8	7	92.3	85.3	488.0	14.7	22.2	29	5.4	0.0	no
6	8	6	8	1	92.3	88.9	495.6	8.5	24.1	27	3.1	0.0	no
7	8	6	8	1	91.5	145.4	608.2	10.7	8.0	86	2.2	0.0	no
8	8	6	9	2	91.0	129.5	692.6	7.0	13.1	63	5.4	0.0	no
9	7	5	9	6	92.5	88.0	698.6	7.1	22.8	40	4.0	0.0	no

Figura 4 *- Output resultante do código.*

De seguida, é feito uma agrupação da informação em valores numéricos uma vez que muitos classificadores não lidam bem com strings e de forma a reduzir os valores a serem utilizados pelos algoritmos.

A resolução passa por uma agrupação em intervalos tendo em conta o atributo, temperatura "temp" e depois é aplicado o mesmo raciocínio para o atributo, humidade, "RH".

Nível médio da temperatura reside nos intervalos de valores [2.2, 33.30], portanto, a divisão foi a seguinte.

- Se for menor ou igual a 5, é atribuído o valor 0.
- Se for maior que 5 e menor ou igual a 15, é atribuído o valor 1.
- Se for maior que 15 e menor ou igual a 25, é atribuído o valor 2.
- Se for maior que 25, é atribuído o valor 3.

```
for ds in data_list:
    ds.loc[ ds['temp'] <= 5, 'temp'] = 0
    ds.loc[(ds['temp'] > 5) & (ds['temp'] <= 15), 'temp'] = 1
    ds.loc[(ds['temp'] > 15) & (ds['temp'] <= 25), 'temp'] = 2
    ds.loc[ ds['temp'] > 25, 'temp'] = 3

train.head(10)
```

Figura 5 *- Código relativo à atribuição de valores em intervalos.*

O output do código é o seguinte:

Out[44]:

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	burntarea
0	7	5	3	5	86.2	26.2	94.3	5.1	1.0	51	6.7	0.0	0
1	7	4	10	2	90.6	35.4	669.1	6.7	2.0	33	0.9	0.0	0
2	7	4	10	6	90.6	43.7	686.9	6.7	1.0	33	1.3	0.0	0
3	8	6	3	5	91.7	33.3	77.5	9.0	1.0	97	4.0	0.2	0
4	8	6	3	7	89.3	51.3	102.2	9.6	1.0	99	1.8	0.0	0
5	8	6	8	7	92.3	85.3	488.0	14.7	2.0	29	5.4	0.0	0
6	8	6	8	1	92.3	88.9	495.6	8.5	2.0	27	3.1	0.0	0
7	8	6	8	1	91.5	145.4	608.2	10.7	1.0	86	2.2	0.0	0
8	8	6	9	2	91.0	129.5	692.6	7.0	1.0	63	5.4	0.0	0
9	7	5	9	6	92.5	88.0	698.6	7.1	2.0	40	4.0	0.0	0

Figura 6 *- Output resultante do código.*

Aplicando o mesmo raciocínio ao atributo humidade, “RH”:

Nível médio da humidade reside nos intervalos de valores [15.0, 100], portanto, a divisão foi a seguinte.

- Se for menor ou igual a 30, é atribuído o valor 0.
- Se for maior que 30 e menor ou igual a 70, é atribuído o valor 1.
- Se for maior que 70 e menor ou igual a 90, é atribuído o valor 2.
- Se for maior que 90, é atribuído o valor 3.

O código consiste no mesmo, apenas muda o atributo, o output desse código consiste no seguinte.

Out[46]:

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	burntarea
0	7	5	3	5	86.2	26.2	94.3	5.1	1.0	1	6.7	0.0	0
1	7	4	10	2	90.6	35.4	669.1	6.7	2.0	1	0.9	0.0	0
2	7	4	10	6	90.6	43.7	686.9	6.7	1.0	1	1.3	0.0	0
3	8	6	3	5	91.7	33.3	77.5	9.0	1.0	3	4.0	0.2	0
4	8	6	3	7	89.3	51.3	102.2	9.6	1.0	3	1.8	0.0	0
5	8	6	8	7	92.3	85.3	488.0	14.7	2.0	0	5.4	0.0	0
6	8	6	8	1	92.3	88.9	495.6	8.5	2.0	0	3.1	0.0	0
7	8	6	8	1	91.5	145.4	608.2	10.7	1.0	2	2.2	0.0	0
8	8	6	9	2	91.0	129.5	692.6	7.0	1.0	1	5.4	0.0	0
9	7	5	9	6	92.5	88.0	698.6	7.1	2.0	1	4.0	0.0	0

Figura 7 *- Output resultante do código de atribuição em intervalos.*

Numa fase posterior, alguns testes de visualização dos dados, em gráficos foram feitos de forma a ajudar a análise dos valores dos exemplos de treino.

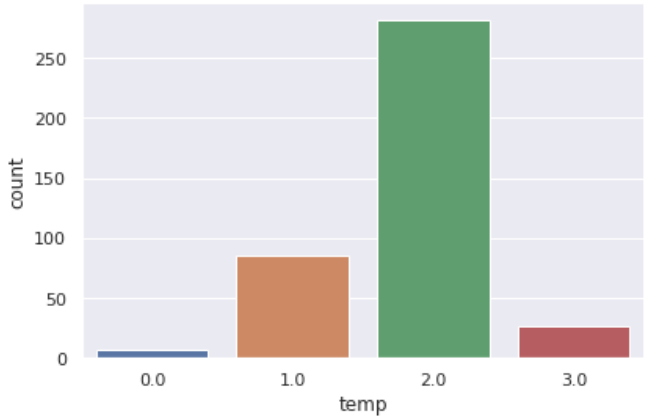

Figura 8 *- Visualização das categorias e os seus valores, associado ao atributo temperatura, "temp".*

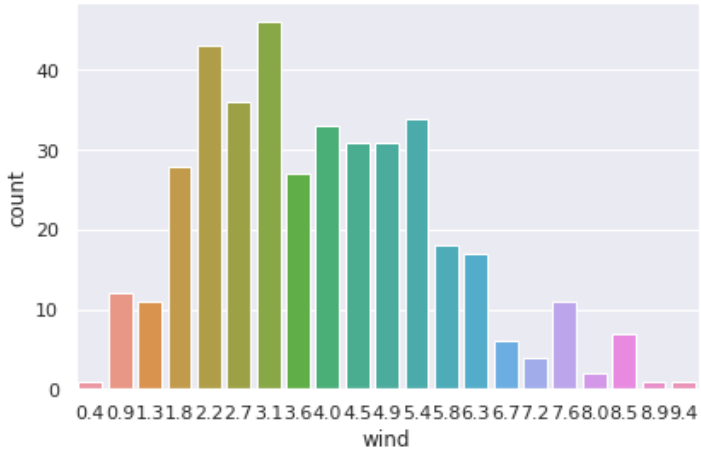

Figura 9 *- Valores associados ao atributo vento "wind".*

Efetuou-se também demonstrações das correlações tanto genericamente como mais especificamente.

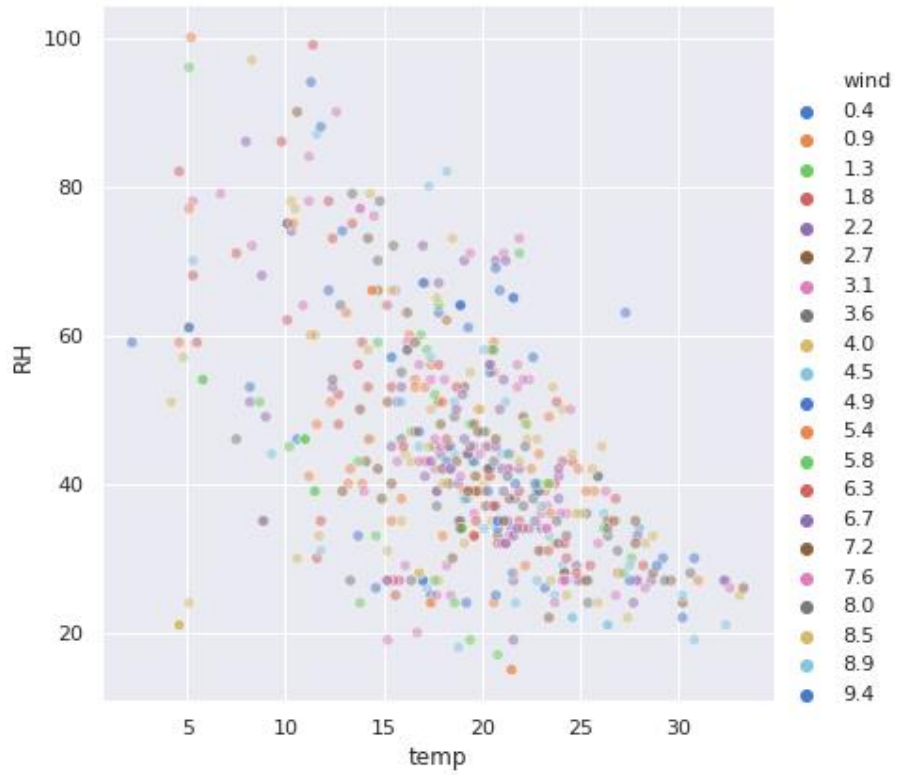

Figura 10 *- Correlação entre a temperatura, "temp" e os valores associados ao vento, "wind".*

Com este gráfico, pode-se concluir que o atributo temperatura "temp" tem uma correlação elevada com o atributo vento, quando ambos possuem altos níveis, seguindo do atributo da humidade, "RH", quando este possui valores mais baixos.

Após a recolha destas informações, existiu a necessidade de remover ("drop") as coluna "burntarea" dos exemplos de treino e de test, de forma a eliminar valores vazios.

Recorreu-se à validação cruzada, dado o tamanho razoável dos valores do dataset e a necessidade de obtenção de valores mais realísticos e precisos.

A validação cruzada consiste na repetição sistemática do processo "holdout", onde acontecem divisões de instâncias em k-folds, cada subconjunto é alternadamente removido e os restantes k-1 são usados para treino.

Recorreu-se o algoritmo de validação cruzada "Ten-Fold Cross Validation", onde k = 10, e o dataset é dividido em dez conjuntos distintos e um conjunto é selecionado para o test e os restantes para o treino.

A validação cruzada ajuda na verificação do desempenho do classificador "DecisionTreeClassifier", que proporciona uma melhor noção do desempenho do mesmo quando exemplos de testes são utilizados.

Classificador Decision Tree Classifier

O classificador "DecisionTreeClassifier" consiste numa implementação caixa-branca onde é possível a visualização do modelo (representado por uma árvore) gerado, tendo uma aprendizagem supervisionada.

```
clf = DecisionTreeClassifier()
scoring = 'accuracy'
score = cross_val_score(clf, train_data, target, cv=k_fold, n_jobs=1, scor
print(score)
```

```
[0.5   0.675 0.55  0.55  0.7   0.575 0.725 0.525 0.725 0.6  ]
```

Figura 11 *- Código e resultado gerado para a construção da árvore.*

A precisão ("*accuracy*") é calculada comparando os valores reais do conjunto dos testes com os valores previstos.

```
print("Accuracy: %0.2f (+/- %0.2f)" % (score.mean(), score.std() * 2))
```

```
Accuracy: 0.62 (+/- 0.17)
```

Figura 12 *– Resultado da precisão do classificador.*

Após a obtenção dos resultados do array é possível o mapeamento da informação obtida com a do teste, sendo possível a comparação dos resultados entre k_fold e usar diretamente o dataset de 400 entradas, sendo resultados parecidos ao da validação cruzada.

```
Score for test dataset with 117 entries:
0.5299145299145299
```

Figura 13 *- Precisão das 117 entradas de exemplos de teste.*

Os dados disponibilizados pela árvore permitem perceber as amostras, ou seja, as amostras ("samples") que chegam aos nós e aos respetivos valores divididos pelos nós.

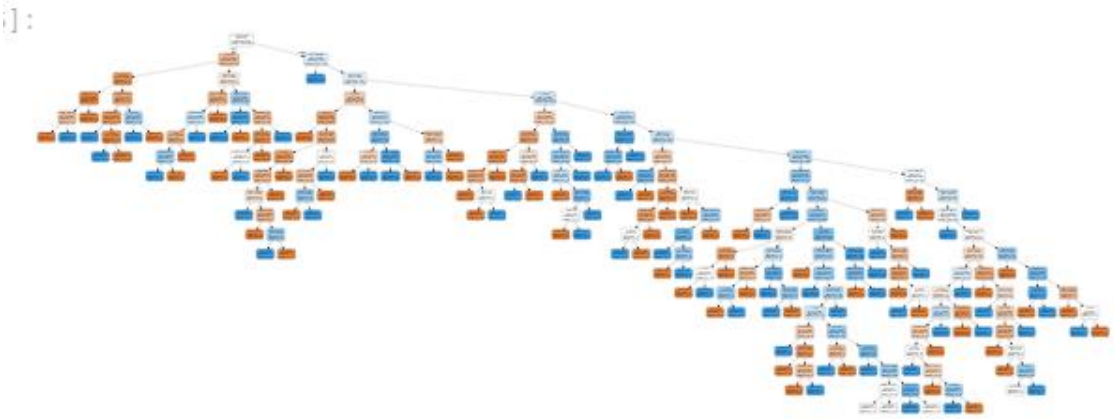

Figura 14 *– Árvore.*

Classificador Naive Bayes

Este classificador assume que a presença de um recurso específico de uma determinada classe não está interligada à presença de outro recurso, possuindo uma aprendizagem supervisionada.

Consiste num classificador que possui independência entre os atributos, isto é, todas as propriedades contribuem independentemente para a probabilidade.

Modelo Gaussian

Associado ao classificador, encontra-se o modelo Gaussian onde a informação é formatada como solicita a informação.

Na chamada do modelo, é possível efetuar o "fit" para prever os resultados, no entanto, é necessário um ajuste do output do exemplos de treino para respeitar as regras do modelo Gaussian.

```
model = GaussianNB()

scoring = 'accuracy'
score = cross_val_score(model, x_train, y_train.values.ravel(), cv=k_fold, n_jobs=1, scoring=scoring)
print("Score:")
print(score)
```

```
Score:
[0.475 0.5   0.625 0.5   0.45  0.6   0.575 0.575 0.55  0.6  ]
```

Figura 15 *– Código gerado para o classificador Naive Bayes e o seu resultado.*

```
Accuracy: 0.55 (+/- 0.11)
```

Figura 16 *- Resultado da precisão do classificador.*

```
Score for test dataset with 117 entries:
0.5384615384615384
```

Figura 17 *- Precisão das 117 entradas de exemplos de teste.*

Classificador Instance Based

Classificador Instance Based denomina k pelo número de vizinhos mais próximos, vizinhos que vão ser considerados, é também o parâmetro do algoritmo, denominado por hiperparâmetros.

Compara novos problemas com aqueles que recebeu no treino, consiste num classificador muito simples que não estabelece generalizações e os dados de validação validam os hiperparâmetros que são usados.

```
from sklearn.neighbors import KNeighborsClassifier
nbrs = KNeighborsClassifier(n_neighbors=3, algorithm='ball_tree')
score = cross_val_score(nbrs, train_data, target, cv=k_fold, n_jobs=1, scoring=scoring)
print('\nScore with nbrs:')
print(score)
```

```
Score with nbrs:
[0.525 0.625 0.575 0.625 0.6   0.6   0.65  0.525 0.575 0.525]
```

Figura 18 *- Código gerado para o classificador Instance Based e o seu resultado.*

```
Accuracy: 0.58 (+/- 0.09)
```

Figura 19 - *Resultado da precisão do classificador.*

```
Score for test dataset with 117 entries:
0.5555555555555556
```

Figura 20 *- Precisão das 117 entradas de exemplos de teste.*

Classificador Support Vector Classification

Classificador ideal para grandes espaços dimensionais, consistindo num classificador com aprendizagem supervisionada.

```
from sklearn import svm

result = svm.SVC(gamma='auto')
score = cross_val_score(result, train_data, target, cv=k_fold, n_jobs=1, scoring=scoring)
print("Score with svm:\n")
print(score)
```

```
Score with svm:

[0.5   0.575 0.5   0.525 0.6   0.55  0.65  0.6   0.625 0.475]
```

Figura 21 *- Código gerado para o classificador SVC e o seu resultado.*

```
Accuracy: 0.56 (+/- 0.11)
```

Figura 22 *- Resultado da precisão do classificador.*

```
Score for test dataset with 117 entries:
0.5128205128205128
```

Figura 23 *- Precisão das 117 entradas de exemplos de teste.*

Classificador Multi Layer Perceptron

Este classificador consiste numa implementação black-box, tendo uma aprendizagem supervisionada.

Em teoria, não é o classificador mais adequado para o problema apresentado tendo em conta que apresenta maiores complexidades no ajuste de hiperparâmetros.

```
from sklearn.neural_network import MLPClassifier
clfneural = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 2), random_state=1)
scoring = 'accuracy'
score = cross_val_score(clfneural, x_train, y_train.values.ravel(), cv=k_fold, n_jobs=1, scoring=scoring)
print("Score:")
print(score)
```

```
Score:
[0.5   0.625 0.5   0.55  0.35  0.525 0.575 0.575 0.525 0.475]
```

Figura 24 *- Código gerado para o classificador SVC e o seu resultado.*

```
Accuracy: 0.52 (+/- 0.14)
```

Figura 25 *- Resultado da precisão do classificador.*

```
Score for test dataset with 117 entries:
0.5128205128205128
```

Figura 26 *- Precisão das 117 entradas de exemplos de teste.*

Análise e Resultados

Classificadores	Validação - Precisão	Resultados obtidos
DecisionTreeClassifier	0,58	0,55
Naive Bayes	0,55	0,5384(…)
Instance-Based	0,58	0,5(5)
SVC	0,56	0,513
MLP	0,52	0,513

Conclusões e Perspetivas de Desenvolvimento

Com base na tabela apresentada, é possível deduzir que o classificador DecisionTree obteve os melhores resultados com 55% uma vez que a validação cruzada atribuiu um valor de precisão de 0,62.

Deduz-se que existe um empate entre dois classificadores SVC e MLP tendo resultados iguais a 51% , distinguindo-se apenas pela precisão. Se detalharmos, o MLP será o que possui pior desempenho.

Tendo em conta os classificadores utilizados, poderão ser considerados outras versões, podendo por exemplo, o classificador random forest ser um substituto à decision tree.

O ajuste de hiperparâmetros apesar de já considerado, poderá ser alvo de futuras melhorias após alterações mencionadas, anteriormente.

Referências

[1] P. Cortez and A. Morais. A Data Mining Approach to Predict Forest Fires using Meteorological Data.
In Proceedings of the 13th EPIA 2007 - Portuguese Conference on Artificial Intelligence, December 2007. (http://www.dsi.uminho.pt/~pcortez/fires.pdf)

<https://archive.ics.uci.edu/ml/datasets/forest+fires> [consultado a 20.05.2020]

[2] Witten, I., Frank, E. e Hall, M. (2011). Data Minning Pratical Machine Learning Tools and Techniques. New York, Elsevier

[2] <https://www.kaggle.com/learn/intro-to-machine-learning> [consultado no dia 24.05.2020]

[3] <https://www.kaggle.com/learn/pandas> [consultado no dia 24.05.2020]

[4]< https://www.datacamp.com/community/tutorials/naive-bayes-scikit-learn> [consultado no dia 24.05.2020

[5] <https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn> [consultado no dia 24.05.2020]

[6]<https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python> [consultado no dia 09.06.2020]

[7] <https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html > [consultado no dia 09.06.2020]