Mineração de dados aplicado a previsão de internamento de pacientes com SRAG

Nayana Julia de Araujo Moreira PG39294
Júlio Miguel de Sá Lima Magalhães Alves PG47390 Joel Salgueiro Martins PG47347
Carlos Filipe Coelho Ferreira PG47087

2022-05-28

Resumo

O trabalho idealizado para esta UC enquadra-se no domínio da saúde, mais concretamente, no do ambiente hospitalar de um país, Brasil, com o principal foco nas doenças respiratórias na época do covid19. O caso de estudo idealizado pela equipa para a mineração dos dados encontrados relativos ao tema passa por conseguir construir um modelo de previsão capaz de descobrir se um determinado utente que apresente um certo conjunto de sintomas deve ou não ser internado na UTI.

Para proceder com o projeto a equipa descobriu um vasto conjunto de dados em bruto fornecidos pelo governo brasileiro, cujo respetivo link se segue: https://opendatasus.saude.gov.br/dataset/srag-2021-e-2022/resource/dd91a114-47a6-4f21-bcd5-86737d4fc734

A equipa realizou diversos processos de mineração de dados, tendo começado por uma análise detalhada do conjunto de dados, seguido do seu tratamento tendo em mente a fase seguinte, portanto, a aplicação de processos de mineração de dados para a construção de um modelo que irá ser avaliado como um sucesso ou não dependendo da adequação desse determinado modelo face ao problema em questão.

As ferramentas usadas em todo o desenvolvimento foram sobretudo o Python com as suas bibliotecas altamente capazes, sklearn e pandas, e o Rstudio, que nos proporcionou todo o ambiente de desenvolvimento intuitivo e eficaz. Também foi utilizado numa fase inicial a ferramenta da Microsoft MS Excel.

Preparação do Dataset

Devido ao facto de o conjunto de dados ser relativo a todo um país com elevado número de habitantes, este apresenta uma enormíssima quantidade de dados. Com aproximadamente 170 atributos e 1 Milhão de registos este não só contem informação relativa aos sintomas das diversas doenças, mas também as características físicas do utente que a possui, assim como a data, hora, hospital onde se dirigiu, entre muitas outras informações técnicas.

Para a construção de um modelo, não é necessária toda informação presente no conjunto de dados em bruto, uma vez que, poderá levar a muitos problemas relativos a desempenho e exatidão, que têm como consequências modelos pouco adequados, com incapacidades de realizar generalizações.

Antes de qualquer avanço, a equipa decidiu reduzir o conjunto de dados bruto, fazendo uso da ferramenta MS Excel. Esta ferramenta permitiu remover atributos dos quais, após breves investigações no dicionário disponibilizado no site correspondente à fonte do conjunto de dados, não foram reconhecidos os seus significados, além de atributos como datas, códigos de registos, localidades tanto de utentes como de hospitais e informações pessoais do utente, tais como número de telefone, família, ocupação profissional. Portanto,

foi removido neste pré-processamento qualquer atributo cuja sua informação é muito difícil de tratar, ou simplesmente porque não possui relação com o utente, fornecedores e lotes de vacinas.

Também devido à potencialidade da ferramenta, foi possível extrair do conjunto de dados em bruto um igual número de registos para cada tipo, sendo o primeiro tipo os registos em que os utentes estavam internados na UTI e o segundo tipo os que não estavam internados. Assim sendo, bastou juntá-los para assim conseguir obter um conjunto de dados ligeiramente mais equilibrado e preparado para responder ao nosso caso de estudo.

Mais concretamente, decidiu-se extrair 20 mil registos de cada, não sendo extraídos mais, devido às capacidades computacionais limitadas das máquinas da equipa usadas no tratamento e processamento dos dados, assim como no treino dos modelos. No entanto, no futuro quando o tratamento de dados for trivializado poder-se-á então aumentar esta quantidade.

Outra das funcionalidades que o MS Excel, permitiu fazer foi a transformação de valores do tipo strings ou caracteres em valores categóricos. No entanto só foi necessário realizar tal para a coluna referente ao sexo do utente, onde foi feita a transformação de 'M' e 'F' para 1 ou 2. Outra tarefa possível foi a filtração de linhas retirando aquelas cujas colunas não estavam preenchidas corretamente, como uma data ou idade invalida, ou erros de digitação.

É possível afirmar que o recurso ao MS Excel, para a remoção de atributos é uma técnica muito utilizada por analistas e cientistas de dados quando estes pretendem realizar processos de mineração de dados. Sendo uma técnica de introspeção, onde se questiona e avalia o significado dos dados realizando operações resultantes do conhecimento e da experiência da pessoa que a executa. Esta técnica é muito eficaz, no entanto, leva a certos possíveis problemas no que toca à tomada de decisões erradas devido a conhecimento desconhecido ou até mesmo incorreto.

Instalações

Para a devida configuração do ambiente "Rstudio", foi necessário proceder à instalação de um certo conjunto de bibliotecas requeridas para o desenvolvimento deste trabalho, sendo apresentados de seguida os comandos para as devidas instalações. O "package reticulate" permite a utilização de "Python" assim como a instalação das suas bibliotecas, o "tinytex" possibilita a transformação de ficheiros "rmarkdown" em formato "pdf". As restantes bibliotecas dizem respeito aos processos de análise, onde fizemos uso do "matplotlib" e do "seaborn", de tratamento de dados, onde fizemos uso dos "pandas" e do "numpy" e de aplicação de modelos, disponibilizados pelo "scikit-learn".

```
#install.packages("reticulate")
#library(reticulate)
#tinytex::install_tinytex()
#py_install("pandas")
#py_install("seaborn")
#py_install("matplotlib")
#py_install("numpy")
#py_install("scikit-learn")
```

Importação

Nesta secção foi feita a importação das bibliotecas requeridas, já mencionadas anteriormente, assim como o carregamento do conjunto de dados em formato "csv" resultante da extração efetuada com o MS Excel anteriormente. No futuro irão ser apresentadas mais importações de bibliotecas, sempre referenciadas.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
import seaborn as sns

df_raw = pd.read_csv("uti_1_e_2_40mil.csv", sep=";")
```

Analise exploratória

Verificação do carregamento:

Após efetuarmos o carregamento do conjunto de dados, executamos a seguinte função para confirmar se tudo for carregado corretamente.

```
df_raw.head()
```

```
##    CS_SEXO  NU_IDADE_N  TP_IDADE  ...  PERD_OLFT  PERD_PALA  TOMO_RES
## 0        1          70         3  ...        2.0        2.0       5.0
## 1        2          75         3  ...        2.0        2.0       NaN
## 2        1          79         3  ...        NaN        NaN       5.0
## 3        2          79         3  ...        2.0        2.0       6.0
## 4        2          82         3  ...        2.0        2.0       1.0
##
## [5 rows x 39 columns]
```

Informação das colunas:

De seguida, é possível observar a existência de 39 atributos distintos no conjunto de dados, assim como, devido ao anterior tratamento anterior efetuado via MS Execll, é possível observar que todos os atributos selecionados são do tipo numérico, portanto, inteiro ou float, o que irá facilitar o posterior o tratamento dos dados, fazendo com que não seja necessária a aplicação de técnicas e cuidados, que normalmente têm de ter tomados na fase de pré-processamento, como a técnica de hot-encoding, binarização, entre outros. De facto, é um acontecimento normal no domínio do problema em que estamos, uma vez que que a maioria dos atributos são relativos a sintomas e o valor do atributo, apenas reflete se o utente, tem ou não tem o sintoma, ou eventualmente é indeterminado. Contudo, existem certos atributos que são exceções à generalidade aqui apresentada, no entanto, estes serão discutidos posteriormente.

```
df_raw.info()
```

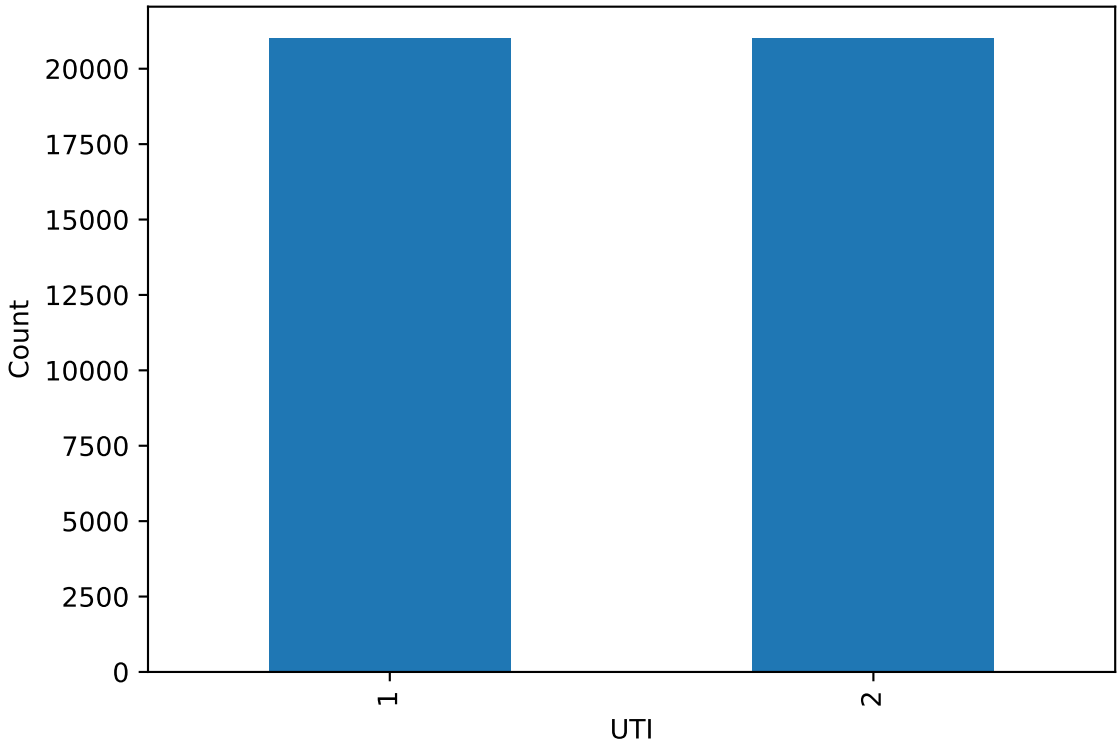
```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 42006 entries, 0 to 42005
## Data columns (total 39 columns):
##  #   Column      Non-Null Count  Dtype
## ---  ------      --------------  -----
##  0   CS_SEXO     42006 non-null  int64
##  1   NU_IDADE_N  42006 non-null  int64
##  2   TP_IDADE    42006 non-null  int64
##  3   CS_GESTANT  42006 non-null  int64
##  4   NOSOCOMIAL  36452 non-null  float64
##  5   AVE_SUINO   35994 non-null  float64
##  6   FEBRE       35474 non-null  float64
##  7   TOSSE       36868 non-null  float64
##  8   GARGANTA    30250 non-null  float64
```

```
##  9   DISPNEIA    37720 non-null  float64
##  10  DESC_RESP   34948 non-null  float64
##  11  SATURACAO   36430 non-null  float64
##  12  DIARREIA    29876 non-null  float64
##  13  VOMITO      29420 non-null  float64
##  14  OUTRO_SIN   30271 non-null  float64
##  15  PUERPERA    15824 non-null  float64
##  16  FATOR_RISC  42006 non-null  int64
##  17  CARDIOPATI  20453 non-null  float64
##  18  HEMATOLOGI  15812 non-null  float64
##  19  SIND_DOWN   15781 non-null  float64
##  20  HEPATICA    15779 non-null  float64
##  21  ASMA        16172 non-null  float64
##  22  DIABETES    18936 non-null  float64
##  23  NEUROLOGIC  16201 non-null  float64
##  24  PNEUMOPATI  16201 non-null  float64
##  25  IMUNODEPRE  15989 non-null  float64
##  26  RENAL       16108 non-null  float64
##  27  OBESIDADE   17018 non-null  float64
##  28  OUT_MORBI   19363 non-null  float64
##  29  VACINA      32424 non-null  float64
##  30  ANTIVIRAL   36835 non-null  float64
##  31  UTI         42006 non-null  int64
##  32  SUPORT_VEN  39757 non-null  float64
##  33  RAIOX_RES   26247 non-null  float64
##  34  EVOLUCAO    38538 non-null  float64
##  35  FADIGA      30646 non-null  float64
##  36  PERD_OLFT   29219 non-null  float64
##  37  PERD_PALA   29196 non-null  float64
##  38  TOMO_RES    30169 non-null  float64
## dtypes: float64(33), int64(6)
## memory usage: 12.5 MB
```

Equilíbrio do dataset:

Com o intuito de demonstrar o equilíbrio do conjunto de dados, o gráfico seguinte apresenta a correção da afirmação anteriormente feita, de que o nosso dataset se encontra equilibrado para o atributo que se pertente prever. Este equilíbrio foi obtido devido ao pré-processamento feito no MS Excel, refletindo bastante importância para a capacidade de generalização do modelo a ser construído. Caso o conjunto de dados não se encontrasse equilibrado, o modelo construído seria incapaz de generalizar, ou apresentaria uma tendência para prever valores que se encontrassem em maior número.

```
plt.figure();
df_raw['UTI'].value_counts().plot(kind='bar')
plt.xlabel('UTI')
plt.ylabel('Count')
plt.show()
```



Valores nulos:

Com a execução do comando seguinte é possível observar a existência de atributos que possuem registos com valor nulo (os atributos com com o "True" associado), estes atributos irão ser tratados posteriormente numa secção seguinte. A realização desse tratamento requer muita atenção, uma vez que como observado anteriormente, o nosso dataset contém uma grande quantidade de valores nulos.

```
df_raw.isna().any()
```

```
## CS_SEXO       False
## NU_IDADE_N    False
## TP_IDADE      False
## CS_GESTANT    False
## NOSOCOMIAL     True
## AVE_SUINO      True
## FEBRE          True
## TOSSE          True
## GARGANTA       True
## DISPNEIA       True
## DESC_RESP      True
## SATURACAO      True
## DIARREIA       True
## VOMITO         True
## OUTRO_SIN      True
## PUERPERA       True
```

```
## FATOR_RISC    False
## CARDIOPATI     True
## HEMATOLOGI     True
## SIND_DOWN      True
## HEPATICA       True
## ASMA           True
## DIABETES       True
## NEUROLOGIC     True
## PNEUMOPATI     True
## IMUNODEPRE     True
## RENAL          True
## OBESIDADE      True
## OUT_MORBI      True
## VACINA         True
## ANTIVIRAL      True
## UTI           False
## SUPORT_VEN     True
## RAIOX_RES      True
## EVOLUCAO       True
## FADIGA         True
## PERD_OLFT      True
## PERD_PALA      True
## TOMO_RES       True
## dtype: bool
```

Conteúdo de cada coluna

Tal como explicado anteriormente, de seguida poderemos observar quais os valores presentes em cada atributo categórico, ou seja, todos menos a idade. A sua descrição no geral, interpreta-se da seguinte forma, 1 significa que contém o sintoma, 2 significa que não contém o sintoma e 9 para desconhecido. No entanto, como era previsível, existem exceções, portanto atributos que têm valorações diferentes das descrita. Entre elas, temos por exemplo a idade de um utente, correspondente ao atributo NU_IDADE_N, assim como muitas outras tais como: CS_GESTANT, SUPORT_VEN, RAIOX_RES, EVOLUCAO, TOMO_RES, AVE_SUINO. Estes atributos que são exceção à regra, terão de ser analisados posteriormente e se for necessário tratados na secção de pré-processamento.

```
for col in set(df_raw.columns) - {'NU_IDADE_N'}:
    print(col,end=": ")
    print(df_raw[col].unique())
```

```
## VOMITO: [ 2. nan  1.  9.]
## FATOR_RISC: [1 2]
## OBESIDADE: [ 2. nan  1.  9.]
## TOMO_RES: [ 5. nan  6.  1.  2.  9.  4.  3.]
## RAIOX_RES: [ 2.  9.  6. nan  4.  5.  1.  3.]
## DIARREIA: [ 2. nan  1.  9.]
## DISPNEIA: [ 1. nan  2.  9.]
## OUT_MORBI: [nan  1.  2.  9.]
## VACINA: [ 2.  1. nan  9.]
## SIND_DOWN: [nan  2.  9.  1.]
## SUPORT_VEN: [ 2.  1. nan  3.  9.]
## PUERPERA: [ 2. nan  9.  1.]
## IMUNODEPRE: [ 2. nan  1.  9.]
```

```
## AVE_SUINO: [ 2. nan  9.  1.  3.]
## EVOLUCAO: [ 2.  1.  3. nan  9.]
## FADIGA: [ 1. nan  2.  9.]
## ANTIVIRAL: [ 2.  9. nan  1.]
## RENAL: [ 2. nan  1.  9.]
## TP_IDADE: [3 2 1]
## SATURACAO: [ 2.  1. nan  9.]
## PNEUMOPATI: [ 2. nan  1.  9.]
## DIABETES: [ 2. nan  1.  9.]
## TOSSE: [ 2.  1. nan  9.]
## HEPATICA: [ 2. nan  1.  9.]
## CARDIOPATI: [ 2. nan  1.  9.]
## UTI: [1 2]
## PERD_OLFT: [ 2. nan  1.  9.]
## FEBRE: [ 2.  1. nan  9.]
## DESC_RESP: [ 1. nan  2.  9.]
## NEUROLOGIC: [ 2. nan  1.  9.]
## CS_GESTANT: [6 5 9 1 2 3 4 0]
## NOSOCOMIAL: [ 2. nan  1.  9.]
## OUTRO_SIN: [ 2.  1. nan  9.]
## PERD_PALA: [ 2. nan  1.  9.]
## HEMATOLOGI: [ 1. nan  2.  9.]
## ASMA: [ 2. nan  1.  9.]
## CS_SEXO: [1 2 9]
## GARGANTA: [ 2.  1. nan  9.]
```

Para além de observarmos quais os valores que cada atributo pode ter, também pode-se realizar a contagem de cada valor, para cada atributo, recorrendo ao comando apresentado de seguida. Como exemplo, podemos observar que no caso do atributo CS_SEXO, o conjunto de dados possui 5 registos de utentes, cujo valor foi ignorado.

```
df_raw['CS_SEXO'].value_counts(dropna=False)
```

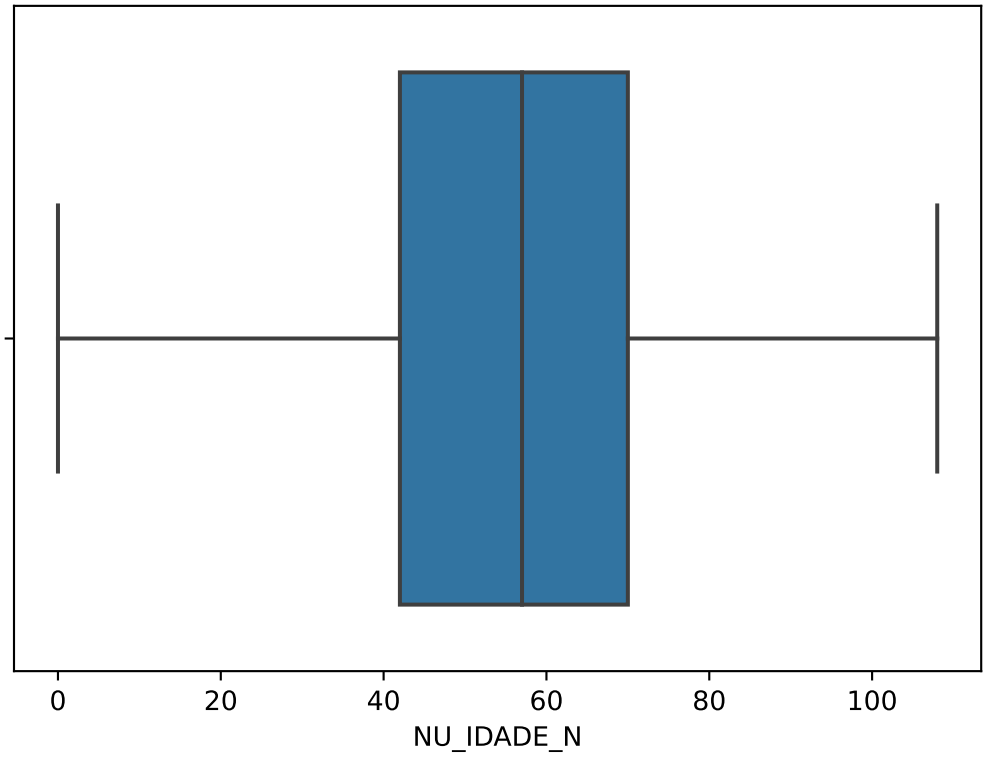
```
## 1    23007
## 2    18994
## 9        5
## Name: CS_SEXO, dtype: int64
```

```
#dropna é pra contar os nulls
df_raw['FEBRE'].value_counts(dropna=False)
```

```
## 1.0    22074
## 2.0    12970
## NaN     6532
## 9.0      430
## Name: FEBRE, dtype: int64
```

Para analisar os valores contínuos, apesar da sua existência reduzida, utilizamos o seguinte gráfico. Este gráfico, permitiu-nos analisar a distribuição de idades dos utentes abrangidos pelos nossos registos. Como sua interpretação, podemos concluir que grande parte dos utentes, cerca de 50% se encontram com idade entre os 40 e 70 anos.

```
plt.close('all')
sns.boxplot(x='NU_IDADE_N',data=df_raw)
plt.show()
```



Devido ao formato do nosso conjunto de dados, nomeadamente no que diz respeito à reduzida presença de atributos com valores numéricos do tipo contínuo, consideramos que não existe necessidade de tratar outliers. Apesar do tratamento de outliers não ser necessário, os atributos previamente mencionados, nos quais se usava o valor inteiro 9 para representar o valor desconhecido, demonstram a necessidade de realizar possíveis tratamentos. Estes serão abordados na próxima secção de pré-processamento.

Correlação entre colunas

De seguida, decidimos perceber qual a correlação entre os atributos, uma vez que este é um fator muito importante para a construção dos modelos. No entanto, podemos observar que que existe pouca correlação entre o atributo que desejamos prever e os atributos utilizados para a sua previsão, isto irá gerar um pouco dificuldade na construção de modelo eficiente. Em contrapartida, o tratamento dos atributos ainda não foi realizado, pelo que estes valores poderão estar fortemente errados.

Iremos novamente realizar esta verificação na próxima secção, após o tratamento dos atributos.

```
corr_matrix=df_raw.corr()
corr_matrix['UTI']
```

```
## CS_SEXO       0.019331
## NU_IDADE_N   -0.101530
```

```
## TP_IDADE     -0.005628
## CS_GESTANT   -0.014245
## NOSOCOMIAL   -0.034405
## AVE_SUINO    -0.103191
## FEBRE        -0.019043
## TOSSE        -0.042260
## GARGANTA     -0.049394
## DISPNEIA      0.051445
## DESC_RESP     0.047422
## SATURACAO     0.065971
## DIARREIA     -0.045222
## VOMITO       -0.055331
## OUTRO_SIN    -0.020347
## PUERPERA     -0.021916
## FATOR_RISC    0.131108
## CARDIOPATI    0.033703
## HEMATOLOGI   -0.010104
## SIND_DOWN    -0.010030
## HEPATICA     -0.008010
## ASMA         -0.018375
## DIABETES      0.025483
## NEUROLOGIC   -0.010347
## PNEUMOPATI   -0.002334
## IMUNODEPRE   -0.008450
## RENAL         0.016857
## OBESIDADE     0.031236
## OUT_MORBI     0.032466
## VACINA       -0.115801
## ANTIVIRAL    -0.078766
## UTI           1.000000
## SUPORT_VEN    0.261652
## RAIOX_RES    -0.026084
## EVOLUCAO     -0.118165
## FADIGA       -0.022392
## PERD_OLFT    -0.046053
## PERD_PALA    -0.050454
## TOMO_RES      0.084708
## Name: UTI, dtype: float64
```

Pre-processamento

Tratamento de colunas especificas - CS_GESTANT

O atributo relativo a gestante tem como objetivo identificar o desenvolvimento da gravidez em utentes do sexo feminino, sendo que:

- valor 1 = 1º Trimestre
- valor 2 = 2º Trimestre
- valor 3 = 3º Trimestre
- valor 4 = Idade Gestacional Ignorada
- valor 5 = Não
- valor 6 = Não se aplica (tem sexo masculino)
- valor 9 = ignorado (devido a idade)

Portanto, foi então decidido representar este atributo sob a forma quantitativa do período de gravidez. Todos os valores representados com "não", "não se aplica", "ignorado", serão convertidos para o valor 0.

```
df_raw['CS_GESTANT'].value_counts(dropna=False)
```

```
## 6    26703
## 5    13283
## 9     1640
## 3      218
## 2      112
## 1       33
## 4       16
## 0        1
## Name: CS_GESTANT, dtype: int64
```

```
df_raw['CS_GESTANT'].replace(6,0,inplace=True)
df_raw['CS_GESTANT'].replace(5,0,inplace=True)
df_raw['CS_GESTANT'].replace(4,0,inplace=True)
df_raw['CS_GESTANT'].replace(9,0,inplace=True)
df_raw['CS_GESTANT'].value_counts(dropna=False)
```

```
## 0    41643
## 3      218
## 2      112
## 1       33
## Name: CS_GESTANT, dtype: int64
```

Tratamento de colunas especificas - SUPORT_VEN

Este atributo representa a necessidade de o utente utilizar suporte ventilatório, portanto, decidimos transformar o atributo de forma a que o valor 0 signifique não ou ignorado, 1 signifique que foi necessário e 2 representa necessidade, mas invasivo, para que seja novamente um atributo cujos valores possam ser ordenáveis onde ter suporte é superior a não ter.

Os significados dos valores originais eram:

- Valor 1 = sim, invasivo
- Valor 2 = sim, não invasivo
- Valor 3 = não
- Valor 9 = Ignorado

```
df_raw['SUPORT_VEN'].value_counts(dropna=False)
```

```
## 2.0    22081
## 1.0    10173
## 3.0     6684
## NaN     2249
## 9.0      819
## Name: SUPORT_VEN, dtype: int64
```

```
#Para fazer swap entre 1 e 2 transferiu-se os valores de 1 para 4 temporariamente
df_raw['SUPORT_VEN'].replace(3,0,inplace=True)
df_raw['SUPORT_VEN'].replace(9,0,inplace=True)
df_raw['SUPORT_VEN'].replace(1,4,inplace=True)
df_raw['SUPORT_VEN'].replace(2,1,inplace=True)
df_raw['SUPORT_VEN'].replace(4,2,inplace=True)
df_raw['SUPORT_VEN'].value_counts(dropna=False)
```

```
## 1.0    22081
## 2.0    10173
## 0.0     7503
## NaN     2249
## Name: SUPORT_VEN, dtype: int64
```

Tratamento de colunas especificas - RAIOX_RES

Neste atributo decidimos simbolizar com o valor 0 os casos em que não é necessário realizar o raio-x, 1 nos casos em que é necessário raio-x normal, 2 nos casos em que é necessário realizar um raio x não normal.

Os significados dos valores originais eram:

- Valor 1 = Normal
- Valor 2 = Infiltrado intersticial
- Valor 3 = Consolidação
- Valor 4 = Misto
- Valor 5 = Outro
- Valor 6 = Não realizado
- Valor 9 = Ignorado

```
#RAIOX_RES
df_raw['RAIOX_RES'].replace(6,0,inplace=True)
df_raw['RAIOX_RES'].replace(9,0,inplace=True)
df_raw['RAIOX_RES'].replace(3,2,inplace=True)
df_raw['RAIOX_RES'].replace(4,2,inplace=True)
df_raw['RAIOX_RES'].replace(5,2,inplace=True)
df_raw['RAIOX_RES'].value_counts(dropna=False)
```

```
## 0.0    16525
## NaN    15759
## 2.0     8670
## 1.0     1052
## Name: RAIOX_RES, dtype: int64
```

Tratamento de colunas especificas - EVOLUCAO

Este atributo refere-se à evolução do utente, portanto, simboliza se o utente foi curado ou falecido. Uma vez que não representa informação relativa à internação do utente, nem a nenhum sintoma ou caraterística física do utente, decidimos como consequência remover o atributo.

```
df_raw.drop(['EVOLUCAO'], axis=1,inplace=True)
```

Tratamento de colunas especificas - TOMO_RES

Este atributo informa o resultado da tomografia feita ao utente. Decidimos simbolizar com valor 0 no caso de não ter sido detetado quaisquer doenças, 1 no caso de ter sido detetado covid normal, 2 no caso de ter sido detetada qualquer doença diferente de covid normal.

Os significados dos valores originais eram:

- Valor 1 = Típico covid-19
- Valor 2 = Indeterminado covid-19
- Valor 3 = Atípico covid-19
- Valor 4 = Negativo para Pneumonia
- Valor 5 = Outro
- Valor 6 = Não realizado
- Valor 9 = Ignorado

```
#TOMO_RES
df_raw['TOMO_RES'].replace(3,2,inplace=True)
df_raw['TOMO_RES'].replace(4,0,inplace=True)
df_raw['TOMO_RES'].replace(5,2,inplace=True)
df_raw['TOMO_RES'].replace(6,0,inplace=True)
df_raw['TOMO_RES'].replace(9,0,inplace=True)
df_raw['TOMO_RES'].value_counts(dropna=False)
```

```
## 1.0    15815
## NaN    11837
## 0.0    10959
## 2.0     3395
## Name: TOMO_RES, dtype: int64
```

Tratamento de colunas especificas - AVE_SUINO

Este atributo indica se o utente esteve em contacto com aves, sendo simbolizado com valor 1 no caso de ter tido, 2 no caso de não ter tido e 3 um caso que não esta no dicionário (referência ao link da fonte do dataset), sendo este caso obviamente um erro pelo que poderíamos remover ou assumir como desconhecido, decidimos optar pela segunda hipótese.

```
#AVE_SUINO
df_raw['AVE_SUINO'].value_counts(dropna=False)
```

```
## 2.0    29009
## 9.0     6582
## NaN     6012
## 1.0      364
## 3.0       39
## Name: AVE_SUINO, dtype: int64
```

```
df_raw['AVE_SUINO'].replace(3,9,inplace=True)
df_raw['AVE_SUINO'].value_counts(dropna=False)
```

```
## 2.0    29009
## 9.0     6621
## NaN     6012
## 1.0      364
## Name: AVE_SUINO, dtype: int64
```

Tratamento da coluna target UTI

No atributo UTI, a indicação que um utente foi internado é 1, e a indicação que não foi internado é 2, depois de aplicar normalização, 0 indicará internado que muitas vezes é visto na matemática como negativo e 1 será não internado que muitas vezes é visto como positivo, para uma melhor interpretação das métricas como "sensivity" e "precision", numa secção superior trataremos já estes valores.

```
df_raw['UTI'].value_counts(dropna=False)
```

```
## 1    21007
## 2    20999
## Name: UTI, dtype: int64
```

```
df_raw['UTI'].replace(2,0,inplace=True)
df_raw['UTI'].value_counts(dropna=False)
```

```
## 1    21007
## 0    20999
## Name: UTI, dtype: int64
```

Tratamento de valores indeterminados

Uma vez que o valor indeterminado é sempre representado por 9, em semelhança ao valor null, como consequência da inserção de valoração não representativa para o atributo quando se realiza o registo, decidimos então transformar todos os registos com valor 9 em valor nulo, para assim todos serem tratados de igual forma.

```
for i in df_raw.columns:
    if (i != 'NU_IDADE_N'):
      df_raw[i].replace(9,np.NaN,inplace=True)
```

Normalização dos valores

O passo seguinte de todo este procedimento passa por normalizar os atributos, quer isto dizer, fazer com que todos os atributos possuam valores entre 0 e 1, para que posteriormente no treino dos modelos não existam atributos tendenciosos, por outras palavras, atributos que o modelo irá reconhecer com maior importância, seja atribuindo mais peso ao seu significado, etc. Esta técnica padrão de mineração de dados revela-se muito importante para as fases seguintes.

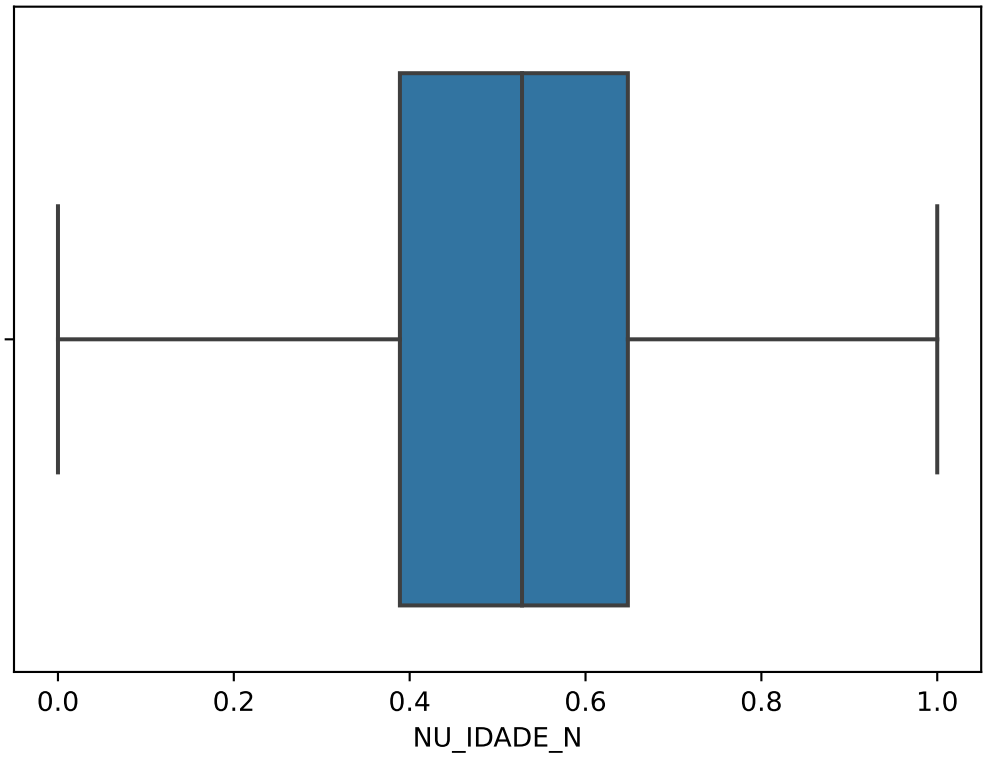
Após aplicado o processo de normalização, podemos verificar que, por exemplo, os valores do atributo idade, estão representados dentro de um intervalo limitado entre 0 e 1.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df_raw[df_raw.columns] = scaler.fit_transform(df_raw[df_raw.columns])
```

```
plt.close('all')
sns.boxplot(x='NU_IDADE_N',data=df_raw)
plt.show()
```



Tratamento de valores nulos

A nossa decisão para preencher os valores nulos, portanto, registos com algum atributo indeterminado, passou por substitui-los respetivamente pela probabilidade de um utente ter ou não ter o sintoma representado pelo atributo que estamos a tratar, em relação ao conjunto de dados na sua totalidade. Essa probabilidade pode ser calculada pela média do valor registado para o determinado atributo, sendo que todos os cálculos de probabilidades serão arredondados a duas casas decimais, uma vez que a elevada precisão de uma probabilidade para o nosso caso de estudo não se revela muito importante, assim como irá garantir a ocupação de menor espaço em memória.

```
for i in df_raw.columns[df_raw.isnull().any(axis=0)]:
    df_raw[i].fillna(round(df_raw[i].mean(),2),inplace=True)
```

Conteúdo das colunas

De seguida fez-se a observação do conteúdo de cada atributo, após todo o processamento anteriormente descrito, para assim verificarmos a os resultados de todas as operações efetuadas.

Chegado a este ponto, podemos verificar pelo comando seguinte que já não há qualquer valor nulo no conjunto de dados, indeterminadamente de qual seja o atributo.

```
for col in set(df_raw.columns) - {'NU_IDADE_N'}:
    print(col,end=": ")
    print(df_raw[col].unique())
```

```
## VOMITO: [1.   0.88 0.  ]
## FATOR_RISC: [0. 1.]
## OBESIDADE: [1.   0.76 0.  ]
## TOMO_RES: [1.   0.37 0.   0.5 ]
## RAIOX_RES: [1.   0.   0.35 0.5 ]
## DIARREIA: [1.   0.83 0.  ]
## DISPNEIA: [0.   0.19 1.  ]
## OUT_MORBI: [0.41 0.   1.  ]
## VACINA: [1.  0.  0.8]
## SIND_DOWN: [0.99 1.   0.  ]
## SUPORT_VEN: [0.5  1.   0.53 0.  ]
## PUERPERA: [1.   0.99 0.  ]
## IMUNODEPRE: [1.   0.94 0.  ]
## AVE_SUINO: [1.   0.99 0.  ]
## FADIGA: [0.   0.64 1.  ]
## ANTIVIRAL: [1.   0.96 0.  ]
## RENAL: [1.   0.91 0.  ]
## TP_IDADE: [1.  0.5 0. ]
## SATURACAO: [1.   0.   0.22]
## PNEUMOPATI: [1.   0.91 0.  ]
## DIABETES: [1.   0.54 0.  ]
## TOSSE: [1.   0.   0.24]
## HEPATICA: [1.   0.98 0.  ]
## CARDIOPATI: [1.   0.37 0.  ]
## UTI: [1. 0.]
## PERD_OLFT: [1.   0.88 0.  ]
## FEBRE: [1.   0.   0.37]
## DESC_RESP: [0.  0.3 1. ]
## NEUROLOGIC: [1.   0.91 0.  ]
## CS_GESTANT: [0.         0.33333333 0.66666667 1.        ]
## NOSOCOMIAL: [1.   0.98 0.  ]
## OUTRO_SIN: [1.   0.   0.54]
## PERD_PALA: [1.   0.88 0.  ]
## HEMATOLOGI: [0.   0.98 1.  ]
## ASMA: [1.   0.92 0.  ]
## CS_SEXO: [0.   1.   0.45]
## GARGANTA: [1.   0.   0.78]
```

```
df_raw.isnull().values.any()
```

```
## False
```

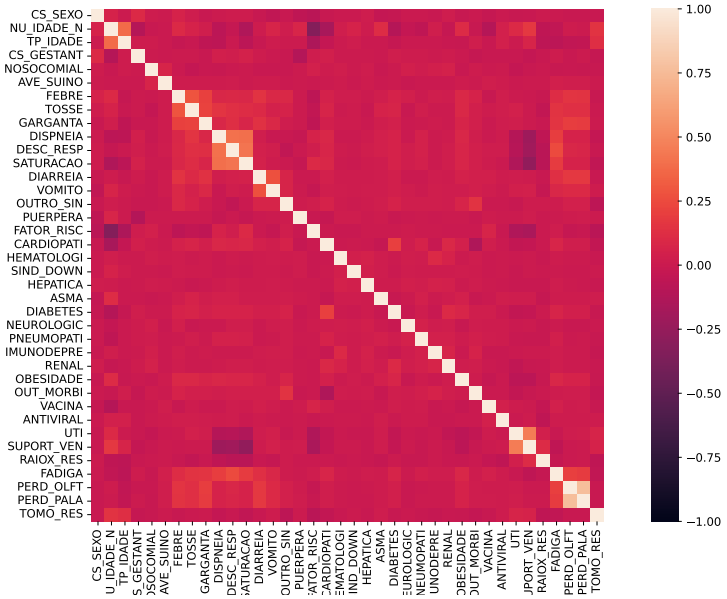
Correlação

Podemos reparar após todo o processo anterior que a correlação entre os diversos atributos com o atributo que queremos prever aumentou significativamente. Estas alterações devem-se a todo o processo de tratamento de dados realizado, demonstrando assim a sua extrema importância.

```
corr_matrix=df_raw.corr()
corr_matrix['UTI']
```

```
## CS_SEXO      -0.021399
## NU_IDADE_N    0.101530
## TP_IDADE      0.005628
## CS_GESTANT   -0.015578
## NOSOCOMIAL   -0.032621
## AVE_SUINO    -0.001116
## FEBRE         0.001776
## TOSSE         0.042073
## GARGANTA      0.027255
## DISPNEIA     -0.105254
## DESC_RESP    -0.097172
## SATURACAO    -0.127754
## DIARREIA      0.017024
## VOMITO        0.032381
## OUTRO_SIN    -0.021193
## PUERPERA      0.005146
## FATOR_RISC   -0.131108
## CARDIOPATI   -0.043627
## HEMATOLOGI   -0.004821
## SIND_DOWN    -0.012291
## HEPATICA     -0.014077
## ASMA          0.019855
## DIABETES     -0.041522
## NEUROLOGIC   -0.004303
## PNEUMOPATI   -0.015493
## IMUNODEPRE   -0.004579
## RENAL        -0.051071
## OBESIDADE    -0.070512
## OUT_MORBI    -0.043294
## VACINA        0.008004
## ANTIVIRAL    -0.001885
## UTI           1.000000
## SUPORT_VEN    0.451264
## RAIOX_RES     0.032253
## FADIGA       -0.009147
## PERD_OLFT     0.019889
## PERD_PALA     0.027642
## TOMO_RES      0.067978
## Name: UTI, dtype: float64
```

```
plt.close('all')
f,ax = plt.subplots(figsize=(15,8))
sns.heatmap(corr_matrix,vmin=-1,vmax=1,square=True,annot=False)
plt.show()
```



CheckPoint

Após realizado todo o procedimento anterior foi então guardado o dataset processado num novo documento csv, e para confirmar que tudo foi bem guardado executamos os seguintes comandos apresentados.

Pela sua análise podemos verificar que tudo está de acordo com o previsto, portanto, temos um dataset com maior tamanho face ao original, facimente justificável com o preenchimento dos valores nulos pelas respetivas médias.

Este passo permite avançar diretamente para a fase de mineração de dados sem ser necessário executar todos os comandos anteriores.

```
#index_label=False para não formar uma nova coluna com os index de cada linha
df_raw.to_csv('transformed.csv',index_label=False,sep=";")
df_raw.head()
```

```
##    CS_SEXO  NU_IDADE_N  TP_IDADE  ...  PERD_OLFT  PERD_PALA  TOMO_RES
## 0      0.0    0.648148       1.0  ...       1.00       1.00      1.00
## 1      1.0    0.694444       1.0  ...       1.00       1.00      0.37
## 2      0.0    0.731481       1.0  ...       0.88       0.88      1.00
## 3      1.0    0.731481       1.0  ...       1.00       1.00      0.00
## 4      1.0    0.759259       1.0  ...       1.00       1.00      0.50
##
## [5 rows x 38 columns]
```

Test and Train

Com o conjunto de dados já processado e pronto para ser utilizado na fase seguinte, correspondente à criação dos modelos de previsão, precisamos de proceder à sua divisão, portanto, separar os atributos que serão fornecidos como input aos modelos do atributo que pretendemos prever, respetivamente o "UTI".

Após efetuado o processo anterior, procedemos à divisão do conjunto de dados em dois conjuntos, respetivamente, o conjunto de treino e de teste, não existindo registos pertencentes a mais do que um conjunto, uma vez que cada um deles será utilizado para uma finalidade especifica.

```
df = pd.read_csv("transformed.csv", sep=";")
df.head()
```

```
##    CS_SEXO  NU_IDADE_N  TP_IDADE  ...  PERD_OLFT  PERD_PALA  TOMO_RES
## 0      0.0    0.648148       1.0  ...       1.00       1.00      1.00
## 1      1.0    0.694444       1.0  ...       1.00       1.00      0.37
## 2      0.0    0.731481       1.0  ...       0.88       0.88      1.00
## 3      1.0    0.731481       1.0  ...       1.00       1.00      0.00
## 4      1.0    0.759259       1.0  ...       1.00       1.00      0.50
##
## [5 rows x 38 columns]
```

```
y=df['UTI'].to_frame()
X=df.drop(['UTI'],axis=1)
y.value_counts(dropna=False)
```

```
## UTI
## 1.0    21007
## 0.0    20999
## dtype: int64
```

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,np.ravel(y),test_size=0.20,random_state=2022)

y_test[:10]
```

```
## array([0., 0., 0., 1., 1., 1., 0., 0., 1., 0.])
```

```
X_train.head()
```

```
##        CS_SEXO  NU_IDADE_N  TP_IDADE  ...  PERD_OLFT  PERD_PALA  TOMO_RES
## 28994      0.0    0.277778       1.0  ...       1.00       1.00      0.50
## 1297       0.0    0.370370       1.0  ...       1.00       1.00      0.50
## 26635      1.0    0.722222       1.0  ...       1.00       1.00      1.00
## 632        1.0    0.490741       1.0  ...       0.88       0.88      0.37
## 12585      0.0    0.490741       1.0  ...       0.88       0.88      0.37
##
## [5 rows x 37 columns]
```

PCA - Principal Component Analysis

Existem ainda diversas técnicas que podem ser utilizadas para melhorar a eficiência do conjunto de dados já processado, uma delas é o PCA, uma técnica que tem como objetivo reduzir a quantidade de atributos presentes num conjunto de dados, consequentemente reduzindo o seu tamanho e aumentando a velocidade de processamento do conjunto de dados, portanto, a construção dos modelos será bastante mais rápida.

Esta técnica para ser bem aplicada obriga a certos pré-requisitos no conjunto de dados ao qual se vai aplicar, sendo um deles o facto de deverem existir diversos atributos correlacionados entre si. Portanto, como o nosso conjunto de dados revela pouca correlação entre os atributos, como é observável pelo "heat map" da matriz de correlação feito anteriormente, a aplicação desta técnica pode não resultar num aumento significativo de performance.

No entanto, com o intuito de perceber se o nosso raciocínio estava correto, para avaliar a potencialidade do PCA no conjunto de dados, decidiu-se na fase seguinte avaliar um dos modelos mais compatível usando dois diferentes cenários, comparando o seu desempenho, sendo os cenários a construção desse modelo com e sem a aplicação do PCA.

Para a aplicação da técnica PCA, a intenção original era diminuir o número de atributos para 30, descartando 7 atributos que através da matriz de correlação revelaram ser mais correlacionados que a média geral. No entanto, podemos realizar uma espécie de "gridSearch" para testar diferentes números de atributos e verificar qual destes tem a soma de taxas de variação mais desejada.

Valores abaixo de 0.6 não são recomendados pois tornaria o conjunto de dados consideravelmente diferente do original, acima de 0.9 são considerados sobre ajustamentos, pelo que indicam uma alteração não totalmente ótima.

Pela análise do gráfico decidimos adotar um número de componentes de aproximadamente 17, que demonstra uma soma das taxas arredondada aos 80%, diferindo bastante do número teoricamente deduzido de 30.

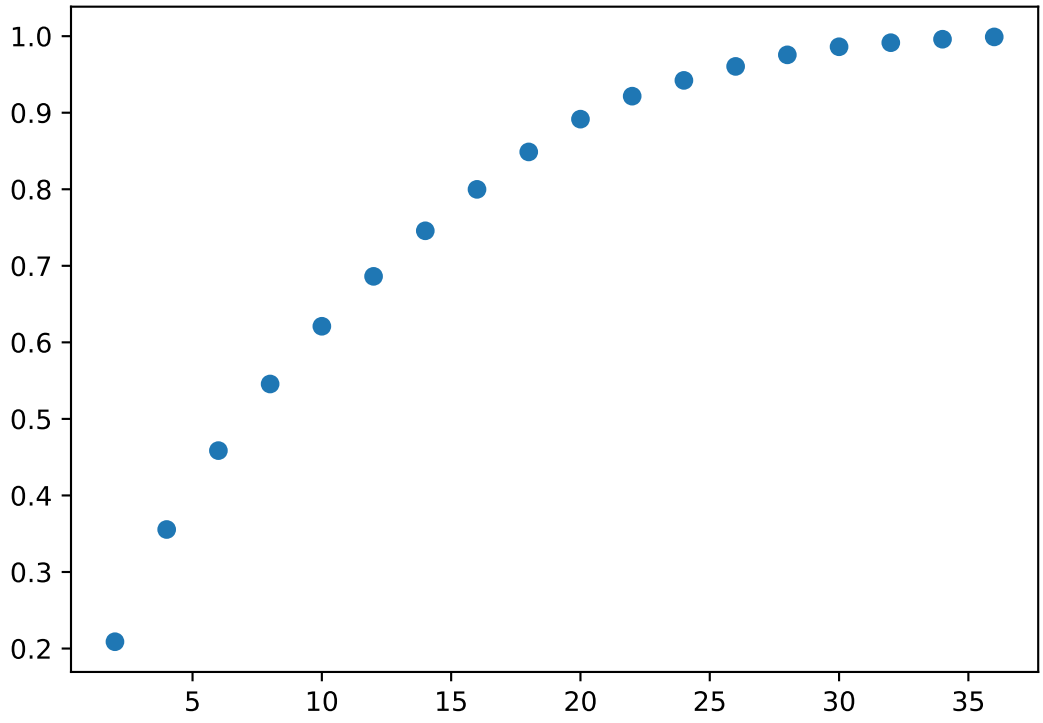
```
from sklearn.decomposition import PCA

lista=[]
for i in range(2,38,2):
  pca = PCA(n_components = i)

  X_trainPCA = pca.fit(X_train)

  sumR=sum(pca.explained_variance_ratio_)
  lista.append((i,sumR))

#print(list)
plt.close('all')
plt.scatter(*zip(*lista))
plt.show()
```




```
pca = PCA(n_components = 17)

X_trainPCA = pca.fit_transform(X_train)
X_testPCA = pca.transform(X_test)

print(pca.explained_variance_ratio_)
```

```
## [0.11762986 0.09128361 0.07830015 0.06833888 0.05400014 0.04907118
##  0.04427501 0.04281391 0.03915445 0.03624147 0.03394754 0.03116409
##  0.03037394 0.02915993 0.0273538  0.02676404 0.02594156]
```

```
print(sum(pca.explained_variance_ratio_))
```

```
## 0.8258135540943098
```

Mineracão de dados

Ao longo deste capítulo iremos realizar diferentes técnicas para a construção de modelos, avaliando resumidamente, se atingiu ou não com sucesso o objetivo delineado anteriormente, assim como efetuar uma breve comparação entre cada um deles.

É importante realçar que foi sempre utilizada a técnica de "cross validation" e, dependendo do caso, "nested cross validation", essas técnicas usam o dataset de treino previamente obtido para retirar uma percentagem que serve como dataset de validação para ajustar os hiper parámetros de cada modelo ou avaliar a sua performance de forma imparcial ao dataset usado para treino.

No entanto, como estas técnicas exigem bastante capacidade de processamento por parte das nossas máquinas, e como consequência, demoram bastante tempo, força-nos ao uso de valores pequenos para "cv" ou quantidade e variedade de hiper parámetros. Apesar destas técnicas serem mais úteis em conjuntos de dados não balanceados, ou com poucos registos, não as descartamos para o nosso caso, mesmo tendo os dados bastante balanceados e com bastantes registos.

Todos os modelos construídos foram mencionados durante a UC, pelo que iremos explicar brevemente o funcionamento de cada um.

Knn neighboards

Começando pelo modelo mais simples dos abordados nas aulas, o "Knn neighboards" começa por calcular um certo valor de registos cuja soma dos valores de input seja próxima à soma dos valores de input do registo que se deseja prever sendo assim vizinho. Por exemplo, se forem calculados 10 vizinhos e 8 desses apresentarem como output um valor x, então o valor a prever por este modelo também terá o valor x. Isto é possível ser feito sem qualquer treino, o que é uma das grandes vantagens deste modelo.

Em contrapartida assumimos que este modelo irá apresentar bons resultados para o nosso caso se a internação de um utente for consequência direta do número de sintomas que este apresenta.

Este modelo pode ser ajustado calculando-o com diferentes números de vizinhos. Para isso iremos utilizar o nested cross validation para assim proceder ao cálculo do valor compatível ao nosso conjunto de dados.

Usando somente o dataset de treino, podemos concluir que uma "accuracy" de aproximadamente 70% num problema de previsão binário não é de todo um resultado que nos satisfaz, no entanto, deixaremos os detalhes desta análise para o capítulo seguinte.

Por outro lado, podemos já observar que o desvio da nossa medida de classificação "accuracy" é de 0 demonstrando novamente que o dataset está equilibrado e em grande quantidade, diminuindo a necessidade da técnica cross validation em geral como previamente mencionado.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV , cross_val_score

knn = KNeighborsClassifier()


search_space = dict(n_neighbors=[5,20,40,50])

gridSearchKnn = GridSearchCV(knn,search_space,cv=5,verbose=0)

res=cross_val_score(gridSearchKnn, X_train, y_train, scoring='accuracy', cv=3)

txt = "Media : {media:.2f} +/- {desvio:.2f}"
print(txt.format(media = res.mean(),desvio=res.std()))
```

```
## Media : 0.67 +/- 0.01
```

```
classifierKnn = gridSearchKnn.fit(X_train,y_train)

print(classifierKnn.best_estimator_.get_params()['n_neighbors'])
```

```
## 50
```

Naive Bayes

Partindo agora para o "Naive Bayes". Este algoritmo faz uso do conjunto de dados de treino para calcular as probabilidades de um utente ser internado para um determinado valor presente em cada atributo. Após o cálculo, quando exposto a um novo registo de input, tem em consideração todas as probabilidades calculadas.

Em contr-partida também assumimos que o algoritmo irá apresentar bons resultados se a internação de um utente depender dos seus sintomas, ponderados respetivamente das suas probabilidades.

Como não existem hiper parâmetros, o uso da técnica cross validation resultou novamente numa "accuracy" bastante insatisfatória de cerca de 70% num problema de previsão binário.

```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()

res=cross_val_score(gnb, X_train, y_train, scoring='accuracy', cv=5)

txt = "Media : {media:.2f} +/- {desvio:.2f}"
print(txt.format(media = res.mean(),desvio=res.std()))
```

```
## Media : 0.66 +/- 0.00
```

```
classifierNaiveBayes = gnb.fit(X_train, y_train)
```

Decision Tree

As "decision trees" são talvez o modelo mais conhecido do machine learning. Tal como o seu nome indica o seu resultado é uma árvore em que cada nodo é uma escolha resultante do valor de input que resulta em vários caminhos possíveis, até chegar a uma folha onde estará presente o output/previsão para aquele registo.

A construção dos nodos pode ser feita de diferentes formas. De forma geral, com recurso a formulas matemáticas calcula qual o atributo que contêm mais peso numa determinada iteração, sendo o valor desse atributo utilizado para fazer a escolha de qual ramo seguir.

Existem diversos parâmetros para a configuração deste modelo, principalmente o método de decisão do nodo, usaremos então novamente o nested cross validation.

Como resultado obtivemos uma "accuracy" ligeiramente superior à anterior, no entanto, continua muito aquém do desejado para um problema de previsão binário.Em comparação este modelo também demonstra uma maior velocidade de treinamento em relação aos restantes.

```
from sklearn.tree import DecisionTreeClassifier

params = {'max_leaf_nodes': [30,100],'criterion' : ['gini','entropy']}

gridSearchTree = GridSearchCV(DecisionTreeClassifier(random_state=2022), params, verbose=0, cv=3)

res=cross_val_score(gridSearchTree, X_train, y_train, scoring='accuracy', cv=3)

txt = "Media : {media:.2f} +/- {desvio:.2f}"
print(txt.format(media = res.mean(),desvio=res.std()))
```

```
## Media : 0.71 +/- 0.00
```

```
classifierTree = gridSearchTree.fit(X_train, y_train)

print(classifierTree.best_estimator_.get_params()['criterion'])
```

```
## entropy
```

```
print(classifierTree.best_estimator_.get_params()['max_leaf_nodes'])
```

```
## 30
```

Ensamble - Random Forest Tree

Uma das estratégias mais comumente aplicada para melhorar o resultado de diversos tipos de modelos é a aplicação do mesmo modelo diversas vezes sobre o mesmo conjunto de dados, mas de diferentes formas.

Neste caso concreto iremos utilizar o método chamado "Random Forest Tree", que consiste em formar diversas árvores diferentes e inserir os registos que desejamos prever em todas elas, calculando posteriormente a maioria dos resultados obtidos.

Utilizaremos entropy um dos parâmetros calculados na realização do modelo "Decision Tree", mas tentaremos calcular qual o numero de arvores ou seja estimadores que mais se ajusta usando nested cross validation.

Infelizmente os resultados obtidos não melhoraram face à versão feita anteriormente com uma só "Decision Tree".

Como esperado este modelo tem uma velocidade muito inferior ao uso de uma única árvore.

```
from sklearn.ensemble import RandomForestClassifier

params = {'n_estimators': [10,30,50,100]}

classifierForest = RandomForestClassifier(criterion='entropy',max_leaf_nodes=100)

gridSearchForest = GridSearchCV(classifierForest, params, verbose=0, cv=3)

res=cross_val_score(gridSearchForest, X_train, y_train, scoring='accuracy', cv=3)

txt = "Media : {media:.2f} +/- {desvio:.2f}"
print(txt.format(media = res.mean(),desvio=res.std()))
```

```
## Media : 0.71 +/- 0.00
```

```
classifierForest = gridSearchForest.fit(X_train, y_train)
print(classifierForest.best_estimator_.get_params()['n_estimators'])
```

```
## 30
```

Redes Neurais

De seguida, iremos fazer uso do modelo baseado em redes neurais, uma tecnologia bastante inovadora que, por norma, garante bons resultados para todo o tipo de problemas.

A forma como as redes neurais funcionam é complexa, pelo que, a sua explicação não será aprofundada neste relatório. De uma forma básica, as redes neurais simulam o comportamento do cérebro humano, sendo constituídas por diversos neurónios interligados entre si por sinapses. Estas sinapses têm um peso associado e sempre que são ativadas, o respetivo neurónio que recebe o valor dessa sinapse automaticamente o pondera com o peso atribuído à sinapse. Os pesos atribuídos às sinapses são ajustados na fase de treino do modelo.

Para este modelo, como existem hiper parâmetros, principalmente a topologia da rede, em geral usaríamos a tecnica nested cross validation, no entanto a construção desde é uma das mais demoradas por razões da sua complexidade. Como já foi comprovado que o dataset não justifica bastante o uso da técnica, será usado somente um "gridSearch".

Em algumas iterações deste modelo, as redes neurais podem demorar bastante convergir, sendo introduzido e testado um parâmetro "max_iter" que impede o treino excessivo de uma rede, neste caso pode surgir avisos sobre a incapacidade da rede convergir, logo iremos temporariamente desliga-los.

Infelizmente, com total desânimo não foi observado um aumento significativo na "accuracy".

```
import warnings
warnings.filterwarnings("ignore")

from sklearn.neural_network import MLPClassifier

params = {"hidden_layer_sizes": [(100,),(50,50)], "activation": ["logistic"]}

#Outros hyper-parametros a introduzir: solver,batch_size,learning_rate,max_iter

gridSearchNet = GridSearchCV(MLPClassifier(random_state=2022,max_iter=100), params,
verbose=0, cv=3,scoring='accuracy')

classifierNet = gridSearchNet.fit(X_train, y_train)

print(classifierNet.best_score_)
```

```
## 0.7045887495389933
```

```
print(classifierNet.best_estimator_.get_params()['activation'])
```

```
## logistic
```

```
print(classifierNet.best_estimator_.get_params()['hidden_layer_sizes'])
```

```
## (50, 50)
```

Ensamble - AdaBoost

De modo a tentar melhorar o desempenho, decidimos aplicar um dos "ensamble" disponíveis, nomeadamente o "AdaBoost", uma vez que estamos perante um problema de classificação binário. Este algoritmo efetua a construção de diversos modelos simples que executam de forma consecutiva, alterando o peso atribuído aos registos em que o modelo anterior falhou a sua classificação, portanto, ajustando o peso com base nos resíduos do modelo anterior.

Foi usado novamente nested cross validation para testar diferentes números de modelos a construir, ou seja, número de estimadores.

Infelizmente, os resultados demonstram novamente um ganho de "accuracy" insignificante.

```
from sklearn.ensemble import AdaBoostClassifier

params = {"n_estimators": [10,50,100]}

classifierAda = AdaBoostClassifier(random_state=2022, algorithm='SAMME')


gridSearchAda = GridSearchCV(classifierAda,params, verbose=0, cv=3,scoring='accuracy')


res=cross_val_score(gridSearchAda, X_train, y_train, scoring='accuracy', cv=3)

txt = "Media : {media:.2f} +/- {desvio:.2f}"
print(txt.format(media = res.mean(),desvio=res.std()))
```

```
## Media : 0.71 +/- 0.00
```

```
classifierAda = gridSearchAda.fit(X_train, y_train)

print(gridSearchAda.best_estimator_.get_params()['n_estimators'])
```

```
## 100
```

Ensamble Gradient Boosting

Para continuar a testar as funcionalidades da técnica ensamble, aplica-se também o "Gradient Boosting", que tem um funcionamento muito semelhante ao anterior, portanto, cria um modelo (normalmente uma árvore) nos resíduos (registos que errou) do modelo anterior, e dá como resultado a previsão do modelo que melhor se ajusta ao registo que se deseja prever, sendo o calculo desse ajuste decidido na fase de treino.

```
from sklearn.ensemble import GradientBoostingClassifier

params = {'n_estimators':[10,50,100]}
```

```
classifierGrad = GradientBoostingClassifier()

gridSearchGrad = GridSearchCV(classifierGrad, params, verbose=0, cv=3)

res=cross_val_score(gridSearchGrad, X_train, y_train, scoring='accuracy', cv=2)

txt = "Media : {media:.2f} +/- {desvio:.2f}"
print(txt.format(media = res.mean(),desvio=res.std()))
```

```
## Media : 0.71 +/- 0.00
```

```
classifierGrad = gridSearchGrad.fit(X_train, y_train)

print(gridSearchAda.best_estimator_.get_params()['n_estimators'])
```

```
## 100
```

Aplicação do PCA em Decision Tree

Chegando a esta fase, resolvemos aplicar o cenário dois, a construção de um modelo usando PCA. Para isso, selecionamos um dos modelo com maior "accuracy" e eficiência, nomeadamente o decisionTree, usando novamente a técnica nested cross validation.

Assim será possível comparar a eficiência de ambas as abordagens.

Como já era um pouco previsível, a "accuracy" diminuiu, apesar de ser uma uma diminuição não muito acentuada, decidiu-se que a aplicação do PCA não é vantajosa, não sendo necessário testar mais modelos usando esta técnica.

```
params = {'max_leaf_nodes': [30,100],'criterion' : ['gini','entropy']}

gridSearchTree = GridSearchCV(DecisionTreeClassifier(random_state=2022), params, verbose=0, cv=3)

res=cross_val_score(gridSearchTree, X_trainPCA, y_train, scoring='accuracy', cv=3)

txt = "Media : {media:.2f} +/- {desvio:.2f}"
print(txt.format(media = res.mean(),desvio=res.std()))
```

```
## Media : 0.67 +/- 0.00
```

```
classifierTreePCA = gridSearchTree.fit(X_trainPCA, y_train)

print(classifierTreePCA.best_estimator_.get_params()['criterion'])
```

```
## gini
```

```
print(classifierTreePCA.best_estimator_.get_params()['max_leaf_nodes'])
```

```
## 100
```

Avaliação

Reflexão

Os resultados, impressionantemente, permaneceram com muito pouca variabilidade, pelo que parece que independentemente do modelo e da técnica utilizada "ensamble", a "accuracy" permanece muito semelhante. Numa seguinte secção irá ser mencionado o acontecimento mais detalhadamente, mas este leva-nos a concluir que existe uma barreira que impede melhores resultados, este problema pode estar no conjunto de dados fornecido ou, de forma mais grave, no pré-processamento de dados, ou simplesmente é um problema difícil e com demasiada instabilidade para automatizar com elevada "accuracy".

Modelos

Tal como dito anteriormente, neste capítulo iremos efetuar uma análise detalhada da performance dos modelos, visto que todos têm uma "accuracy" semelhante, nomeadamente escolheremos três modelos com eficiência satisfatória

É importante realçar que chegado a esta fase, já com todos os modelos treinados e ajustados, iremos fazer uso do conjunto de dados de teste, para assim poder confirmar e validar todos os resultados obtidos anteriormente.

O estudo dos resultados obtidos e comparação dos modelos será realizado principalmente através de matrizes de confusão simples, visto que se trata de um problema binário e "classification reports".

Para uma melhor vizualização dos resultados obtidos, estudaremos 3 modelos diferentes obtidos e usaremos uma confusion matrix, visto de se tratar de uma previsão binária e um classification report estas métricas serão uteis para detetar as diferenças entre cada modelo visto que em accuracy como visto são ambos semelhantes

```
from sklearn.metrics import confusion_matrix

y_predTree  = classifierTree.predict(X_test)
y_predForest = classifierForest.predict(X_test)
y_predAda    = classifierAda.predict(X_test)


cm1 = confusion_matrix(y_test, y_predTree)
cm2 = confusion_matrix(y_test, y_predForest)
cm3 = confusion_matrix(y_test, y_predAda)

TN1, FP1, FN1, TP1 = cm1.ravel()
TN2, FP2, FN2, TP2 = cm2.ravel()
TN3, FP3, FN3, TP3 = cm3.ravel()

def printConfusionMatrix (TN,FP,FN,TP):
  print(f"""
        P      N
        O      E
        S      G
  POS  {TP}   {FN}
  NEG  {FP}   {TN}
  """)

printConfusionMatrix(TN1,FP1,FN1,TP1)
```

```
##
##          P      N
##          O      E
##          S      G
##   POS  2021   2145
##   NEG  275   3961
##
```

```
printConfusionMatrix(TN2,FP2,FN2,TP2)
```

```
##
##          P      N
##          O      E
##          S      G
##   POS  2303   1863
##   NEG  496   3740
##
```

```
printConfusionMatrix(TN3,FP3,FN3,TP3)
```

```
##
##          P      N
##          O      E
##          S      G
##   POS  2450   1716
##   NEG  664   3572
##
```

Antes de mais importa realçar que a matriz de confusão objetiva em avaliar os resultados obtidos pelos modelos, face à realidade, em problemas de classificação binários. Pelo que, ela informa-nos acerca dos valores previstos acertados e errados, podendo assim ser possível extrair diversas métricas, entre as quais:

- Accuracy: de todos os valores previstos, qual percentagem foi prevista com sucesso.
- Negative predictive value: de todos os valores previstos com negativo, qual a percentagem que realmente tinha valor negativo.

É importante notar que existem dois tipos de erros, o primeiro, chamado erro do tipo 1 consiste num falso positivo, o segundo, chamado erro do tipo 2 consiste num falso negativo. Ora para o nosso problema, onde ser internado é 1 positivo e não ser é 0 negativo, uma vez que estamos a prever se um utente deve ou não ser internado com base no conjunto de sintomas que apresenta, um erro do tipo 2 é considerado extremamente grave, significando que o modelo preveu que o utente não deveria ser internado quando deveria, consequentemente agravando o estado da doença no utente.

As matrizes construídas e apresentadas de seguida seguem a seguinte legenda:

- TP# = número de valores previstos como positivo e que realmente eram positivo.
- FN# = número de valores previstos como negativo e que não eram negativo. (Erro do tipo 2)
- FP# = número de valores previstos como positivo e que não eram positivo. (Erro do tipo 1)
- TN# = número de valores previsos como negativo e que realmente eram negativo.

Poderemos ver pelas matrizes acima que os nossos modelos apresentam um valor de "FN#" relativamente superior ao valor de "FP#", pelo que, como o valor de "FN#" está diretamente ligado ao erro do tipo 2

e o valor de "FP#" diretamente erro do tipo 1, os nossos modelos apresentam uma tendência não muito desejada, indicando que a dificuldade em prever casos em que o utente deve ser internado.

Podemos observar que o modelo de decisionTree apresenta menos erros do tipo 2 em relação aos restantes, mas em contrapartida apresenta mais erros do tipo 1, realçando assim para o facto de a accuracy ser bastante semelhante entre modelos.

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_predTree))
```

```
##               precision    recall  f1-score   support
##
##          0.0       0.65      0.94      0.77      4236
##          1.0       0.88      0.49      0.63      4166
##
##     accuracy                           0.71      8402
##    macro avg       0.76      0.71      0.70      8402
## weighted avg       0.76      0.71      0.70      8402
```

```
print(classification_report(y_test, y_predForest))
```

```
##               precision    recall  f1-score   support
##
##          0.0       0.67      0.88      0.76      4236
##          1.0       0.82      0.55      0.66      4166
##
##     accuracy                           0.72      8402
##    macro avg       0.75      0.72      0.71      8402
## weighted avg       0.74      0.72      0.71      8402
```

```
print(classification_report(y_test, y_predAda))
```

```
##               precision    recall  f1-score   support
##
##          0.0       0.68      0.84      0.75      4236
##          1.0       0.79      0.59      0.67      4166
##
##     accuracy                           0.72      8402
##    macro avg       0.73      0.72      0.71      8402
## weighted avg       0.73      0.72      0.71      8402
```

Pelos relatórios de classificação apresentados de seguida, podemos confirmar tudo o que foi previsto na fase anterior. O valor da accuracy em ambos os modelos permaneceu muito semelhante ao apresentado na fase anterior, assim como a accuracy entre diferentes modelos não apresentou variabilidade significativa. O facto de todos os modelos estarem com uma accuracy muito próxima leva-nos a concluir que os modelos não estão sub-ajustados e que estão balanceados.

O desempenho de cada modelo pode ser avaliado pelo cálculo das métricas:

- sensibilidade ou recall ou "true positive rate", indicando a percentagem de casos que um utente foi internado que o modelo conseguiu prever, que conseguimos observar na coluna recall na linha 1.0, referente ao valor ser internado.

- especifidade ou true negative rate, indicando a percentagem de casos que um utente não foi internado que o modelo conseguiu prever, que conseguimos observar na coluna recall na linha 0.0, referente ao valor de não ser internado.
- precisão ou positive predictive value, indicando da totalidade dos casos que o modelo previu como categoria linha referente qual a percentagem de acerto.
- f1 score, indicando um média de sensibilidade e precisão.

A "accuracy" também está presente, porém é uma medida mais suscetível a induzir a uma conclusão errada sobre o desempenho do modelo por isso o focos está nas medidas citadas acima.

As métricas nesta fase de validação tiveram valores bem equilibrados entre modelos mas não ótimos para os resultados desejados. Onde a "accuracy" ficou em torno de 71% a 72%, a especifidade ficou em média superior a 80% indicando que os modelos são bons a prever se um utente não vai ser internado, no entanto a sensibilidade ronda os 50% demonstrando a horrível capacidade de prever que um utente vai ser internado, a métrica de precisão na linha 1.0 como ronda os 80% aos 90% demonstra-nos que os algoritmos em média quando preveem que um utente deve ser internado então normalmente estão corretos.

Para a escolha de melhor resultado levamos em consideração que cada métrica tem suas peculiaridades que devem ser levadas em consideração na escolha de como o modelo de classificação será avaliado. De modo que não se consideramos uma como melhor ou pior que a outra de maneira geral, e sim, iremos analisar o contexto do problema para escolher a que melhor se adapta ao caso.

O algoritmo Decision Tree apresenta os maiores valores de especificidade e precisão de todos os gerados, de modo que, se a proposta for buscar um modelo o mais preciso possível (dentro dos testados) em suas classificações de necessidade de não internamento em UTI, que direcione de forma mais assertiva possível essa indicação no sentido em que um paciente que não necessite de internamento, assim não seja, para levar a otimização da utilização dos recursos médicos e esses recursos não sejam provisionados em casos não necessários e sejam utilizados em casos de real necessidade. Então esse algoritmo melhor se adequa, uma vez que os falsos positivos sao considerados mais prejudiciais que os falsos negativos nesse caso.

Entretanto se a proposta for buscar um modelo mais preciso possível em suas classificações de necessidade de internamento em UTI, que direcione de forma mais assertiva possível essa indicação no sentido em que um paciente que necessite de internamento, assim seja, para levar a diminuição de riscos para o paciente e encontre o maior numero de pacientes com necessidade de internamento mesmo que classifique alguns sem necessidade e assim diminuir a otimização da utilização dos recursos médicos, uma vez que esses recursos podem ser provisionados.

Então o algoritmo AdaBoost melhor se adequa, uma vez que os falsos negativos são considerados mais prejudiciais que os falsos positivos nesse caso. Sendo o que apresenta o maior valor de sensibilidade entre os gerados, a medida nesse caso mais adequada.

Levando em consideração que o objetivo do trabalho visa otimizar a utilização de vagas e recursos de UTI sobretudo em tempos de grande concorrência e escassez desses recursos, de modo que o modelo apoie a decisão medica na gerência de vagas e que casos de real necessidade utilizem tais recursos, então a melhor escolha será o Decision Tree neste caso.

Possiveis Alterações/Melhorias

Uma das alterações que pode ser feita, e que poderá alterar o comportamento de todos os modelos efetuados, é o pré-processamento do conjunto de dados original, nomeadamente, no processo de tratamento de valores nulos, que foi a maior alteração feita ao dataset, não utilizar o valor médio e sim outra heurística ou até mesmo não trata-los visto que alguns classificadores mais recentes já não assim o necessitam.

Também poderão ser feitos ajustes nos hiper parâmetros, poderá ser utilizado um conjunto de dados maior, no entanto esta medida em princípio não será muito eficaz visto que a partir dos resultados semelhantes em todos os modelos foi possível prever que o problema convergiu, outra alteração é reverificar os atributos que demonstram ser mais importantes não eliminando tantas colunas, aplicar a padronização ao invés da normalização, assim como aplicar one-hot encoding, diversas outras.

Todas as modificações irão gerar variabilidade no desempenho dos algoritmos, no entanto, apenas aplicando-as será possível saber quais as mais benéficas.

Representação do conhecimento

Arvore resultante

Como o modelo Decision Tree apresentou valores considerados bons, a árvore resultante permite-nos perceber qual o processo de decisão tomado pelo modelo para calcular a previsão de internação de um utente. A partir desta é possível observar o conhecimento obtido pelo nosso modelo, no entanto, devido ao elevado número de atributos, de seguida só se encontram apresentados os três primeiros níveis, visto que mais níveis a tornaria ilegível.

Pela árvore podemos retirar que dos atributos utilizados para mais facilmente detetar se o utente vai ser internado ou não temos primeiramente a sua necessidade de suporte de ventilação o que é esperado por ser a coluna com maior correlação ao nosso target, seguida do resultado da sua tomografia e vacina.

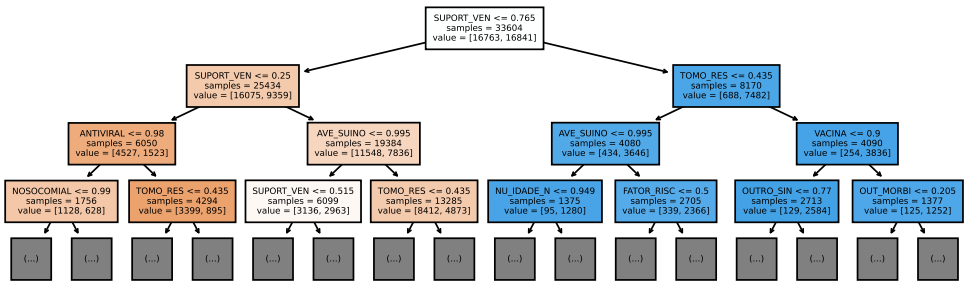
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

clf = DecisionTreeClassifier(random_state=2022,criterion='gini')
clf.fit(X_train,y_train)

#plt.close('all')
```

```
## DecisionTreeClassifier(random_state=2022)
```

```
fig = plt.figure(figsize=(10,3))
_ = tree.plot_tree(clf,feature_names=list(X_train.columns),filled=True,
                   max_depth=3,impurity = False,proportion=False,fontsize=5)
plt.show()
```



Objetivo

A equipa considera com total desagrado que não foram atingidos os objetivos inicialmente delineados na sua totalidade, uma vez que apenas 70% de accuracy num problema binário, como dito anteriormente, é bastante pouco e deixa muito a desejar.

Mesmo com o nosso melhor modelo, a equipa não considera a possibilidade da sua utilização em hospitais, principalmente porque a internação de um utente é uma atividade que custa bastante dinheiro ao hospital e a não internação, quando mal decidida, põe em risco a vida do utente, a área de medicina é um setor que exige muita rigorosidade e atenção, pela qual demanda somente práticas e métodos muito precisos e corretos.

Talvez o problema em si esteja errado, não sendo possível determinar se um utente deve ou não ser internado com base nos atributos e informação recolhidos, exigindo dados diferentes mais específicos ou resoluções diferentes para o mesmo objetivo, o que se revela um facto muito interessante.

Conclusões

Em conclusão consideramos que este trabalho se revelou muito construtivo para as nossas aprendizagens acerca das diversas técnicas, cuidados, métodos e processos de Mineração de dados.

O contacto com este dataset que contém uma enorme quantidade de dados e atributos forçou-nos a surgir com novas metodologias ao qual nunca necessitamos de recorrer na resolução deste tipo de problemas nomeadamente a ferramenta MS Excel que se demonstrou bastante eficiente e facilitou imenso o tratamento de dados.

A avaliação da necessidade de internamento em UTI, sobretudo em tempos há pouco vivenciados de grande concorrência e escassez das vagas mostra-se de grande importância pois quanto mais rápido e eficiente foi feita a identificação da necessidade de internar ou não, melhor se pode agir para preservar a vida do paciente e também para otimizar a utilização das vagas.

Assumimos que para trabalhos futuros, ainda existem diversas atividade possíveis para atingir melhores resultados e consequentemente métricas, de forma adicional também seria importante a adição de resultados de exames laboratoriais ou de imagem, como raio-x, tomografias e outros realizados ainda quando em triagem para ganharmos maior detalhamento do estado do paciente.

Adicionalmente, o aumento de hiper parâmetros e quantidade de dados utilizados, uma vez que apesar do dataset ter mais de 1 milhão de linhas, apenas pouco mais de 40 mil foram utilizadas por razão de limitação de poder de processamento. Também pode ser necessário trabalhar em mais variações de tratamento dos dados para então atingir melhores métricas.

Todavia acreditamos que uma vez muito mais trabalhado e estudado, o modelo resultante na resposta do problema pode ser um contributo fundamental para o desenvolvimento de um sistema de apoio a decisão médica.

Participação - Projeto e Artigo

Como pedido pelas regras da entrega do trabalho prático, nesta secção será introduzido a participação de cada elemento em ambos os trabalhos da cadeira de mineração de dados.

A equipa reconhece que todos os seus membros contribuíram com uma carga de trabalho de forma semelhante na realização deste projeto e na apresentação do artigo e desejamos que todos os membros da equipa sejam avaliados de forma igual.

- Nayana PG39294 - Fonte do conjunto de dados, preparação via MS excel, análise exploratória e pré-processamento, avaliação, relatório e slides de apresentação.
- Carlos PG47087 - Análise exploratória, pré-processamento, mineração de dados, avaliação, relatório e apresentação.
- Joel PG47347 - Relatório, avaliação, representação do conhecimento, resumo do artigo.
- Júlio PG47390- Relatório, resumo, guião e apresentação completa do artigo.

.