Projeto de Classificação com Base PySUS

Projeto Final de *Data Mining*

BI-MASTER - Turma 2021.3

PUC-Rio

Professora

Manoela Kohler

Grupo

Eduardo Marques

Eleonora Weiner

Vinícius Guerra

Vinicius Mattoso

Sumário

Lista de Figuras

Lista de Tabelas

Link para o Github do projeto

https://github.com/vinicius-mattoso/Projeto_Final_DataMining

1. Introdução

As transformações tecnológicas que vêm ocorrendo em nossa sociedade estão facilitando o acesso aos dados das mais variadas fontes, além de catalisar o potencial destes dados para processos de tomada de decisão mais inteligentes. Os dados oriundos do Sistema Único de Saúde (SUS) proporcionam um bom exemplo deste potencial que está sendo desencadeado pela nova era tecnológica. Através desta base, é possível acessar os dados detalhados sobre a produção hospitalar e ambulatorial da rede credenciada ao SUS (pública e privada), dados sobre epidemiologia, oncologia, dentre outros.

Neste trabalho, será utilizada uma parte desta base de dados do SUS para o desenvolvimento de um modelo de *Machine Learning* de classificação. O objetivo do projeto é identificar aquele modelo com a melhor capacidade preditiva para o risco de mortalidade de pacientes internados em estabelecimentos credenciados à rede SUS.

Para realizar esta avaliação dos riscos de mortalidade de pacientes internados, este projeto considerou a base de morbidade hospitalar do SIH/SUS (Sistema de Informação Hospitalar do SUS). Para a resolução deste problema, foram aplicadas algumas técnicas de pré-processamento dos dados e métodos de avaliação de performance para identificar o modelo com o melhor erro preditivo.

Cientes de que este tipo de implementação poderia auxiliar tanto na otimização de processos de triagem clínica de novos pacientes quanto no desenvolvimento de modelos preditivos de seguradoras de vida, apresentamos a seguir o detalhamento das técnicas aplicadas bem como os resultados obtidos.

2. Preparação dos dados utilizados

O acesso às bases de dados do SUS é facilitado pelo portal *TabNet* do Ministério da Saúde e pode ser acessado no link: https://datasus.saude.gov.br/informacoes-de-saude-tabnet/. No entanto, algumas bibliotecas públicas nas linguagens Python e R viabilizam um acesso mais dinâmico a estas bases. Um exemplo é a biblioteca *PySUS*[1], que foi escolhida para a coleta de dados tratados das bases proporcionadas pelo SUS:

```
[ ]  !pip install PySUS
```

Figura 1 – Comando para instalar biblioteca que permite baixar a base de dados utilizada neste projeto.

Sendo o objetivo do projeto avaliar o risco de mortalidade para pacientes com internação em estabelecimentos credenciados da rede SUS, consideramos apenas o universo das bases RD (reduzidas) do SIH/SUS (Sistema de Informação Hospitalar do SUS). Para fins

[1] https://pysus.readthedocs.io/en/latest/

de simplificação, nosso grupo optou por utilizar os dados apenas do Rio de Janeiro reportados em Janeiro/2019. Esse recorte foi feito para restringir o volume de dados e também evitar efeitos de desbalanceamento oriundos do período da pandemia COVID-19 a partir do ano de 2020.

```
from pysus.online_data.SIH import download
df2 = download('RJ', 2019, month = 1)
df2.head()
```

Figura 2 – Comandos utilizados para selecionar e baixar a base de dados utilizada neste projeto.

A base de dados em questão contém 114 colunas e 62.412 registros para a UF escolhida, apenas no período selecionado para análise. Uma explicação mais detalhada de cada coluna pode ser encontrada no arquivo “IT_SIHSUS_1603.pdf”, que corresponde a um Informe Técnico proporcionado pela DIDIS (Divisão de Disseminação de Informações em Saúde) a respeito dos dados contidos na base objeto de estudo. Este documento também está salvo no *github* deste projeto.

3. Análise exploratória e filtro de dados inicial

Através da análise exploratória inicial sobre os dados, foi observado que existiam muitas variáveis não numéricas e sem expressão (códigos) na base de dados. Neste sentido, foi aplicado um primeiro filtro de pré-processamento para expurgar tais variáveis do conjunto a ser analisado.

Em seguida, foi realizada uma análise com o objetivo de identificar aqueles atributos que poderiam ser excluídos por não trazerem ganho de informação para o modelo. Em alguns casos, convertemos o tipo de *string* para numérico ou *float* e a partir do comando *describe* verificamos que existiam atributos com valor 0 em todos os seus registros. Estes atributos foram excluídos na sequência. Adicionalmente, também foram excluídos os atributos que possuíam valores vazios.

Além desses dados, também descartamos dados do tipo *object* com informações de nomes, dados pessoais, dados gerenciais dos hospitais e informações de datas que não eram relevantes para nossa análise ou que eram redundantes com outros dados que mantivemos.

Verificamos todas as colunas e os seus registros via os comandos *len* e *unique*, além de também relacionar graficamente alguns atributos com a classe. Descartamos as informações de acordo com a baixa relação.

Com os atributos restantes para descrever cada registro de paciente, fizemos uma matriz de correlação para identificar quaisquer relacionamentos mais significativos entre os atributos. Observando a correlação das variáveis com a variável *target* MORTE, podemos ver que não há nenhuma alta correlação específica. Assim, para melhor compreender os dados, se faz necessária uma melhor análise gráfica.

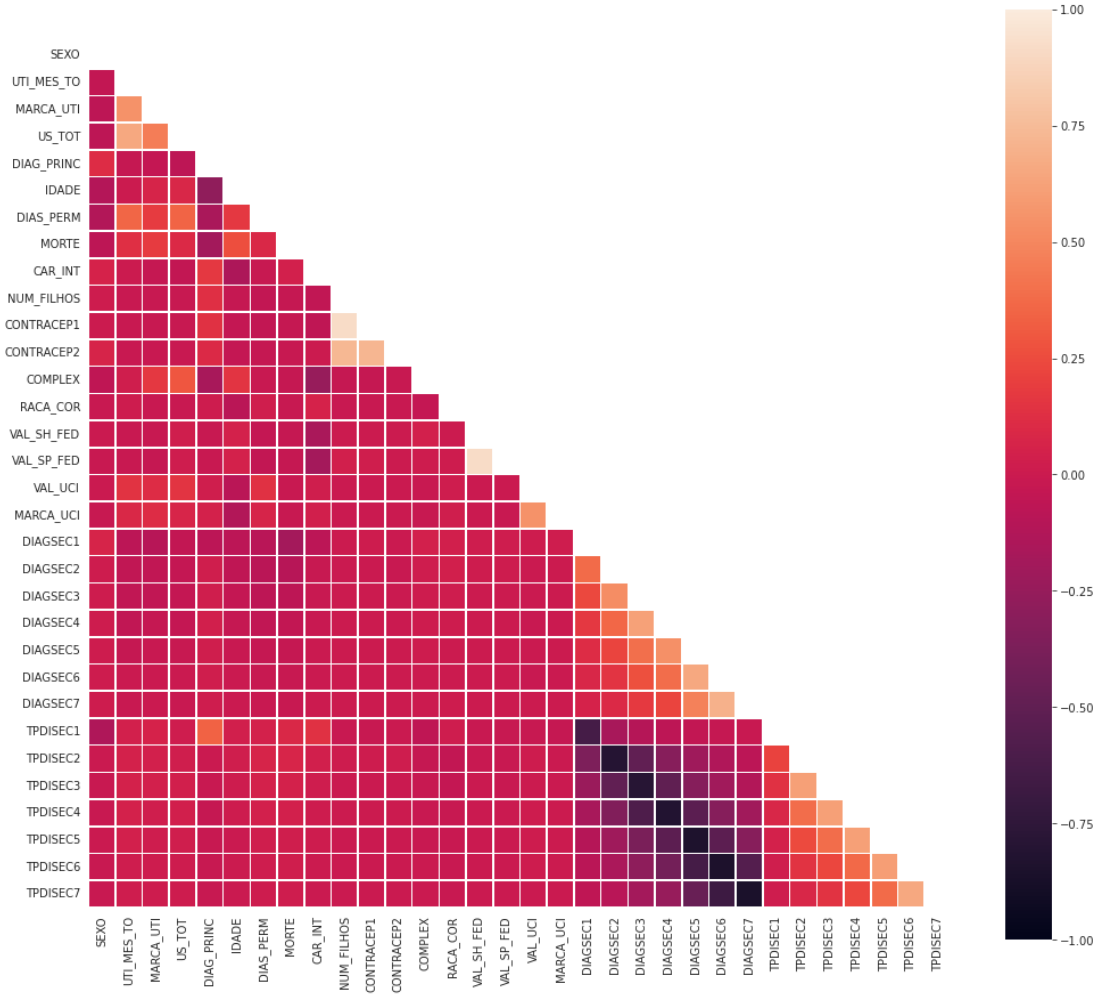

Figura 3 – Matriz de correlação dos atributos selecionados após as etapas prévias para filtrar atributos irrelevantes.

Uma ferramenta interessante que permite analisar correlações graficamente é o gráfico *pairplot.* Nesse gráfico é feita a análise da dispersão cruzada entre cada variável de interesse. Fizemos a avaliação cruzada entre os diferentes diagnósticos recebidos pelo paciente no intuito de avaliar se existe uma correlação linear entre os diagnósticos e se o paciente vive ou morre; assim, o gráfico foi segmentado por esta variável *target*.

```
df_DIAG=df2[['DIAGSEC1','DIAGSEC2','DIAGSEC3',\
             'DIAGSEC4','DIAGSEC5','DIAGSEC6',\
             'DIAGSEC7','MORTE']].copy()
sns.pairplot(df_DIAG, hue="MORTE")
```

Figura 4 – Comandos utilizados para gerar o pairplot.

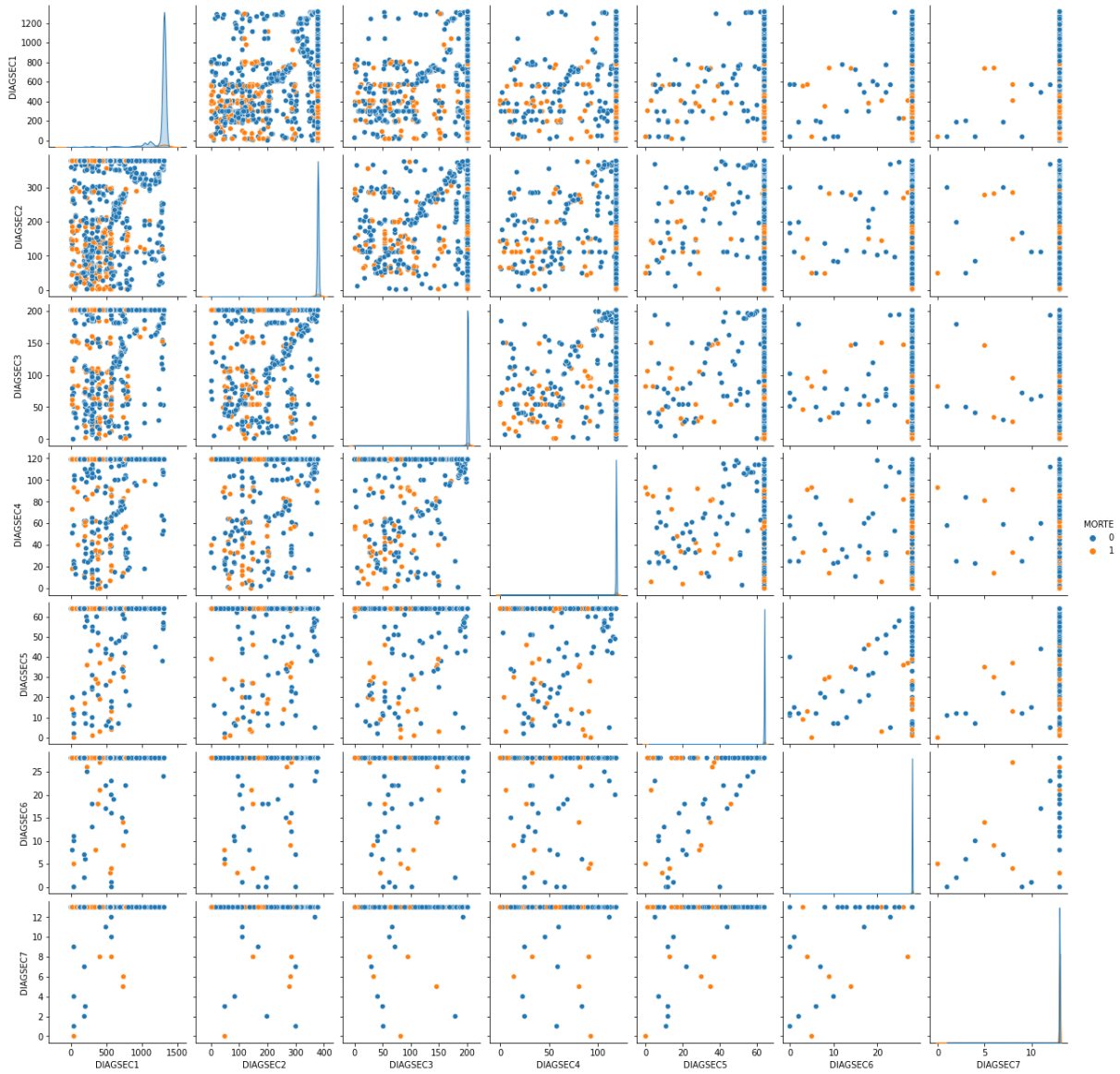

Figura 5 – Pairplot gerado que correlaciona os diagnósticos cadastrados por paciente e a variável alvo MORTE.

Podemos ver na Figura 5 que há poucas correlações entre os diferentes diagnósticos. Existe fraca correlação entre os diagnósticos 2 e 3 e entre os diagnósticos 5 e 6 por exemplo. Além disso, parece haver mais mortes associadas aos diagnósticos 2, 3 e 4.

Para obter melhores inferências sobre os dados, também geramos alguns gráficos de dispersão segmentados. No primeiro gráfico, mostrado na Figura 6, relacionamos o atributo custo total do paciente em dólares e sua idade. Não há clara correlação entre estes dados, porém é possível ver que há uma concentração de indivíduos que morreram com idades maiores que 70 anos com variados custos.

Na Figura 7, relacionamos o atributo dias em que o paciente permaneceu internado e sua idade. Novamente, observamos baixa correlação entre os dados e que a concentração dos indivíduos que morreram possuíam mais de 70 anos.

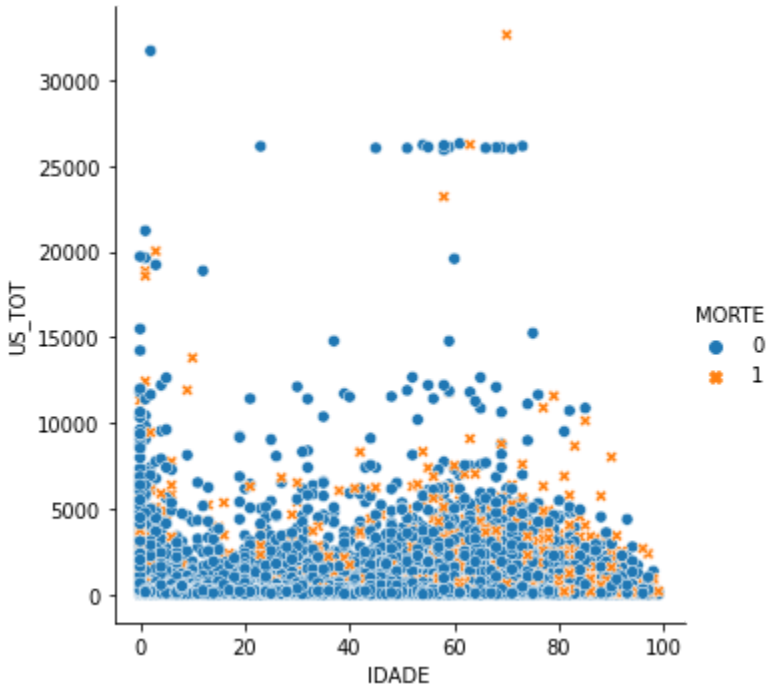

Figura 6 – Gráfico de dispersão relacionando o custo total em dólares do paciente e sua idade

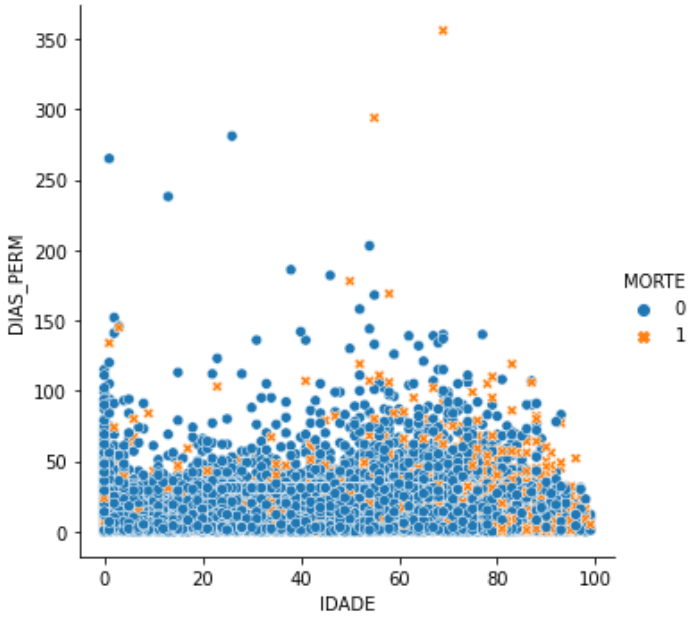

Figura 7 – Gráfico de dispersão relacionando quantos dias o paciente ficou internado e sua idade.

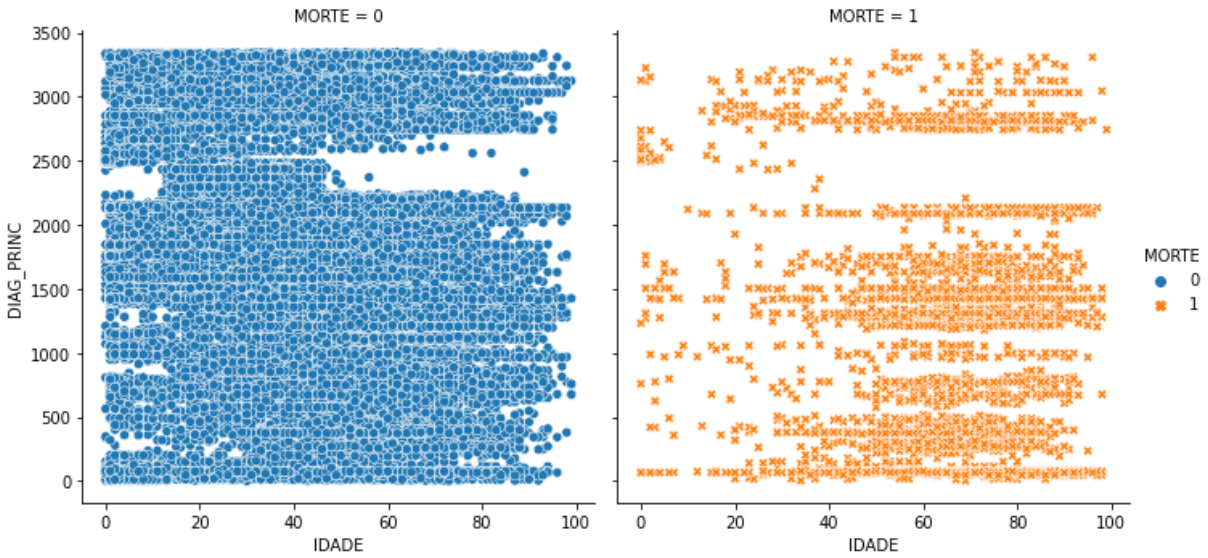

Figura 8 – Gráficos de dispersão relacionando o diagnóstico do paciente e sua idade. À esquerda, os registros em que o paciente viveu e à direita, os registros em que o paciente morreu.

Finalmente, na Figura 8, em que relacionamos a idade do paciente com o seu diagnóstico principal, podemos observar que pessoas com idades mais avançadas têm maior incidência de óbitos para vários CIDs diferentes. Dado que não há claras correlações entre os atributos e a classificação do registro, seguimos com a preparação dos dados para aplicar os modelos, como é detalhado a seguir.

4. *Encoding* dos dados

Algumas variáveis da nossa base de dados trazem detalhes de diagnóstico dos pacientes através da nomenclatura CID (Classificação Internacional de Doenças)[2]. De maneira mais específica, observamos que alguns campos consideram a informação do código CID-10. Como este código contempla a combinação de letras e números, tais variáveis foram inicialmente identificadas com o tipo *object*. A transformação deste tipo de variável é fundamental para a aplicabilidade dos modelos de *Machine Learning* que adotaremos neste problema.

Neste sentido, algumas técnicas como o *label encoding* e o *one hot encoding* fornecem alternativas funcionais de pré-processamento já que ambas são utilizadas para converter dados de texto ou dados categóricos em dados numéricos.

As duas técnicas são provenientes da biblioteca *sklearn*, porém o nosso objetivo é evitar a criação de inúmeras novas colunas (tendo em vista a extensão da classificação proporcionada pelo CID-10). Sendo assim, priorizamos a técnica *label encoding*, que faz a transformação de *object* para número sem criar novas colunas.

[2] http://tabnet.datasus.gov.br/cgi/sih/mxcid10lm.htm

```
df2['TPDISEC7']=df2['TPDISEC7'].astype(str).astype(int)
df2['TPDISEC6']=df2['TPDISEC6'].astype(str).astype(int)
df2['TPDISEC5']=df2['TPDISEC5'].astype(str).astype(int)
df2['TPDISEC4']=df2['TPDISEC4'].astype(str).astype(int)
df2['TPDISEC3']=df2['TPDISEC3'].astype(str).astype(int)
df2['TPDISEC2']=df2['TPDISEC2'].astype(str).astype(int)
df2['TPDISEC1']=df2['TPDISEC1'].astype(str).astype(int)
from sklearn import preprocessing

LABEL_ENCODING_DIAG1 = preprocessing.LabelEncoder()
LABEL_ENCODING_DIAG2 = preprocessing.LabelEncoder()
LABEL_ENCODING_DIAG3 = preprocessing.LabelEncoder()
LABEL_ENCODING_DIAG4 = preprocessing.LabelEncoder()
LABEL_ENCODING_DIAG5 = preprocessing.LabelEncoder()
LABEL_ENCODING_DIAG6 = preprocessing.LabelEncoder()
LABEL_ENCODING_DIAG7 = preprocessing.LabelEncoder()

df2['DIAGSEC1']=LABEL_ENCODING_DIAG1.fit_transform(df2['DIAGSEC1'])
df2['DIAGSEC2']=LABEL_ENCODING_DIAG2.fit_transform(df2['DIAGSEC2'])
df2['DIAGSEC3']=LABEL_ENCODING_DIAG3.fit_transform(df2['DIAGSEC3'])
df2['DIAGSEC4']=LABEL_ENCODING_DIAG4.fit_transform(df2['DIAGSEC4'])
df2['DIAGSEC5']=LABEL_ENCODING_DIAG5.fit_transform(df2['DIAGSEC5'])
df2['DIAGSEC6']=LABEL_ENCODING_DIAG6.fit_transform(df2['DIAGSEC6'])
df2['DIAGSEC7']=LABEL_ENCODING_DIAG7.fit_transform(df2['DIAGSEC7'])

LABEL_ENCODING_DIAG_PRINC = preprocessing.LabelEncoder()
df2['DIAG_PRINC']=LABEL_ENCODING_DIAG_PRINC.fit_transform(df2['DIAG_PRINC'])
```

Figura 9 – Comandos utilizados para transformar os campos do tipo object de interesse para campos do tipo inteiro sem criar novas colunas.

5. *Baseline - Decision Tree Classifier*

Para segmentação da base de dados entre treino e teste, entrada (X) e saída (Y) aplicamos o método *train_test_split* da biblioteca *sklearn*. O tamanho do conjunto de teste corresponde a 20% dos registros. Inicialmente, utilizamos para o treino o modelo *Decision Tree Classifier* também da biblioteca *sklearn*. Este modelo supervisionado é muito utilizado para os problemas de classificação tendo em vista a facilidade de interpretação dos resultados proporcionados diante dos demais modelos mais comumente usados. Nossos resultados para este modelo *baseline* são mostrados abaixo:

1) **Treino:**
   - Acurácia: 0.9999399146788439
   - *Kappa*: 0.9995271655970807

- *F1*: 0.9995594066676458

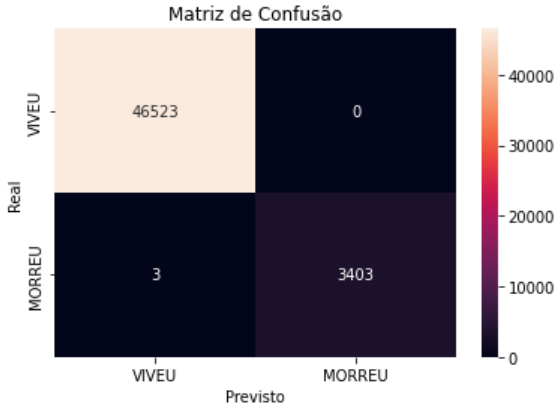

Figura 10 – Matriz de confusão do conjunto de treinamento com o modelo Baseline.

2) Teste

- Acurácia: 0.9157253865256749
- *Kappa*: 0.3388437924012013
- *F1*: 0.3840749414519906

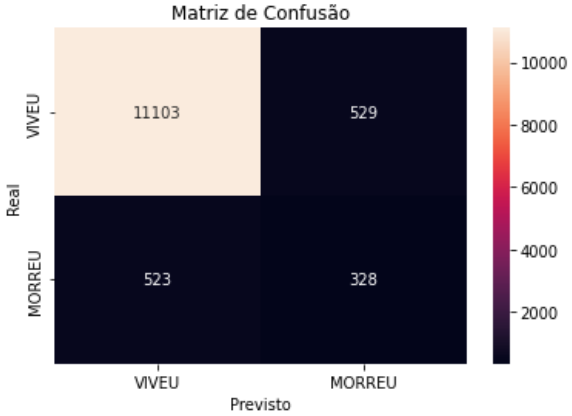

Figura 11 – Matriz de confusão do conjunto de teste com o modelo Baseline.

Embora o modelo *baseline* tenha apresentado excelentes resultados para o conjunto de treino, o mesmo não ocorre com as métricas de *Kappa* (0.338) e *F1-score* (0.3841) do conjunto de testes. Em uma análise mais detalhada sobre os resultados apresentados na matriz de confusão do nosso teste, é possível observar que o modelo errou cerca de 1050 casos entre Falsos Positivos (Erro do Tipo I) e Falsos Negativos (Erro do Tipo II).

Além disso, de um ponto de vista qualitativo sobre este problema, existe uma preocupação adicional em considerar um modelo onde existe um grande erro de previsão de sobrevivência (PREVISTO: “VIVEU”) para pacientes que acabam falecendo (REAL: “MORREU”). Neste sentido, as próximas etapas de pré-processamento e de calibragem irão levar em consideração a minimização dos erros de predição aqui mencionados.

6. Desbalanceamento e *oversampling*

A análise exploratória dos dados revelou que a quantidade de registros de mortes apresentou uma frequência muito inferior à classe de vivos. Esta diferença na frequência de registros se traduz em um problema de desbalanceamento da nossa variável independente (rótulo), ou seja, existe uma classe minoritária dentro da base de dados quando comparada às demais categorias.

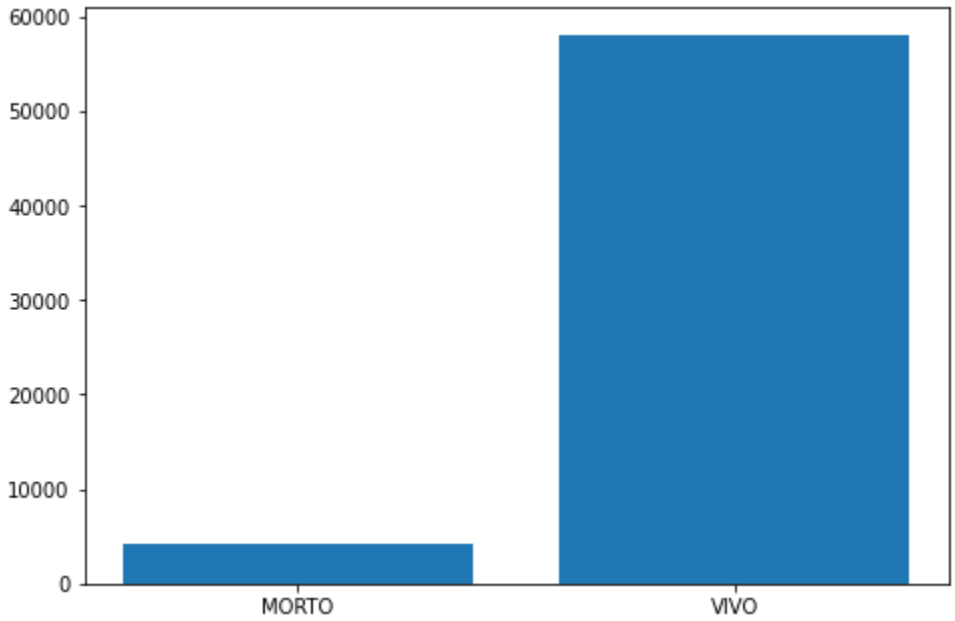

Figura 12 – Histograma dos registros do conjunto de dados de acordo com a classificação. É possível ver a disparidade entre a quantidade de registros em que o paciente morreu e a quantidade de registros em que o paciente viveu.

Existem algumas técnicas capazes de minimizar o impacto do desbalanceamento sobre uma base de dados. O *oversampling*[3], técnica empregada neste estudo, cria novas observações da classe com menor frequência com o objetivo de reduzir a disparidade da frequência entre classes.

[3] https://imbalanced-learn.org/stable/references/over_sampling.html

```
from imblearn.over_sampling import RandomOverSampler
# Instanciar o random over sampler
#   e definir que vamos aumentar até ficar 1 morto para cada 2 vivos
oversample = RandomOverSampler(sampling_strategy=0.5)

# Aplicar o fit_resample em nossa base de dados
X_over, y_over = oversample.fit_resample(X, y)
```

```
# fit and apply the transform
X_over, y_over = oversample.fit_resample(X, y)
```

Figura 13 – Comandos utilizados para realizar o oversampling necessário para balancear o conjunto de dados.

Via *oversampling*, aumentamos os registros de mortes em 50% e equilibramos o número das classes de nossa base de dados conforme a figura abaixo:

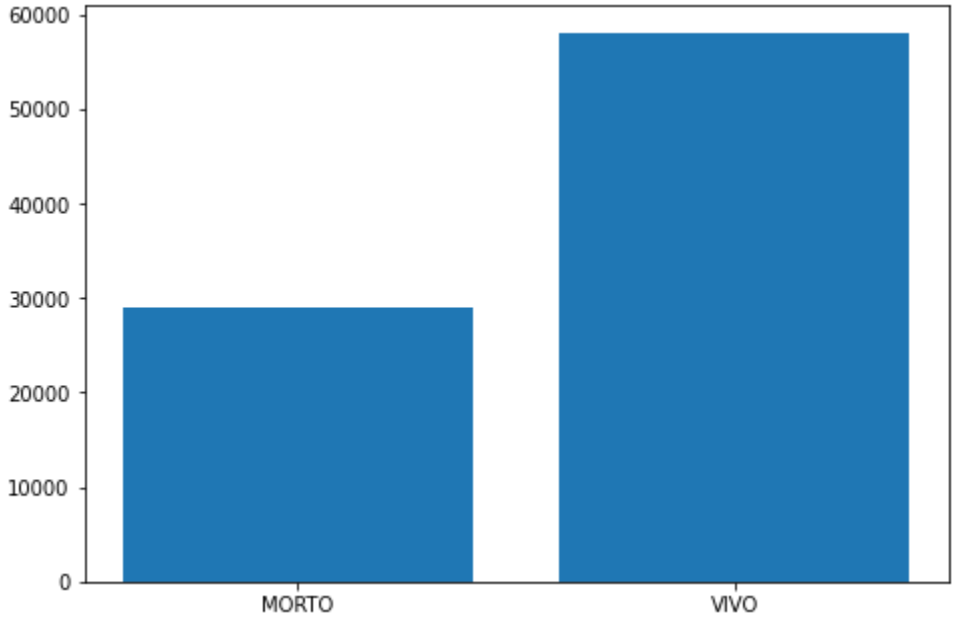

Figura 14 – Resultado do oversampling no conjunto de dados.

Os resultados do *Decision Tree Classifier* após o balanceamento dos dados foram os seguintes:

<u>Modelo *Decision Tree*:</u>

1) **Treino**
   - Acurácia: 0.9998996919108691
   - *Kappa*: 0.9997743205445172
   - *F1*: 0.9998495561907627

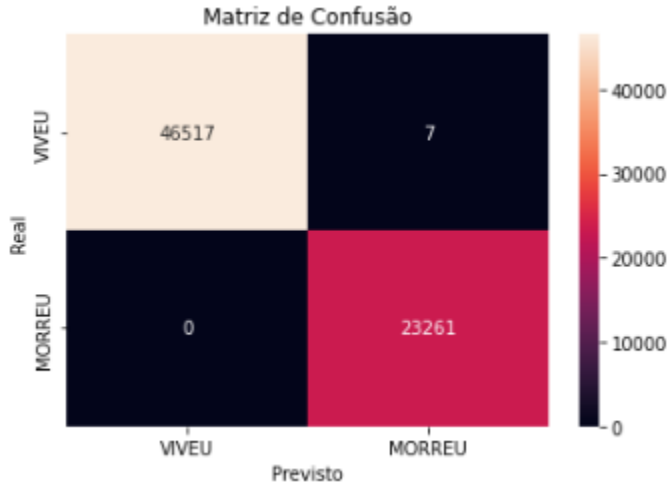

Figura 15 – Matriz de confusão do conjunto de treinamento com o mesmo modelo Baseline após o balanceamento dos dados.

2) Teste

- Acurácia: 0.9666991459849831
- *Kappa*: 0.9268577265608803
- *F1*: 0.9523731453397819

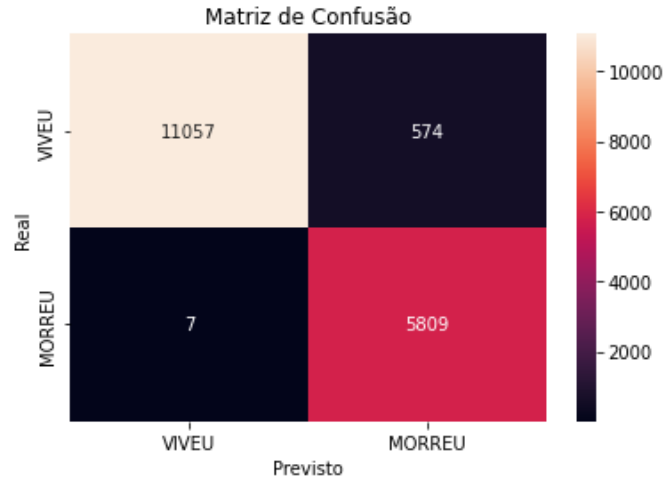

Figura 16 – Matriz de confusão do conjunto de teste com o mesmo modelo Baseline após o balanceamento dos dados.

Nesta etapa, também aplicamos outros dois modelos de *Machine Learning* para solução deste problema de classificação supervisionada: *SVM* e *RandomForest*. Os resultados destes modelos sobre a base já balanceada seguem abaixo:

Modelo SVM:

1) **Treino**
   - Acurácia: 0.781973203410475
   - Kappa: 0.46867532396216893
   - F1: 0.6136854132283863

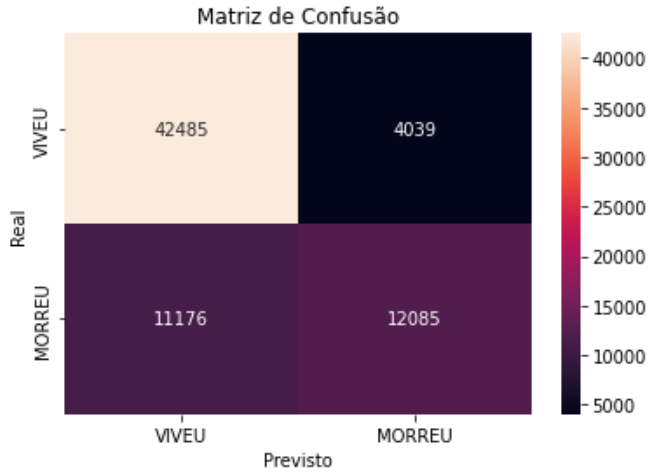

Figura 17 – Matriz de confusão do conjunto de treinamento com o modelo SVM com dados balanceados.

2) **Teste**
   - Acurácia: 0.7829999426835559
   - Kappa: 0.4734237915583417
   - F1: 0.619038035822097

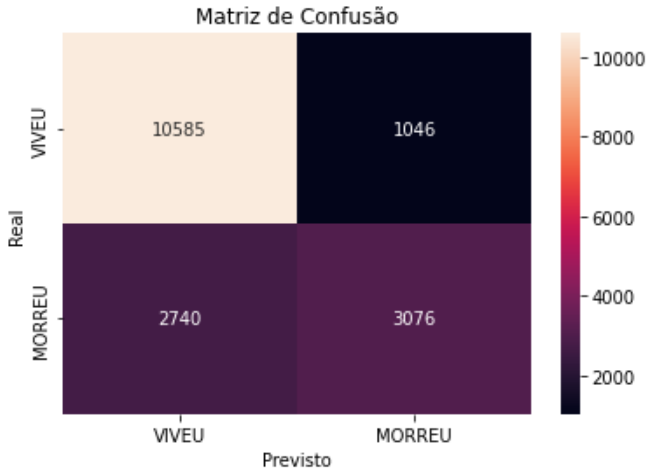

Figura 18 – Matriz de confusão do conjunto de teste com o modelo SVM com dados balanceados.

Modelo *Random Forest*:

1) **Treino**
   - Acurácia: 0.9998996919108691

- Kappa: 0.9997743205445172
- F1: 0.9998495561907627

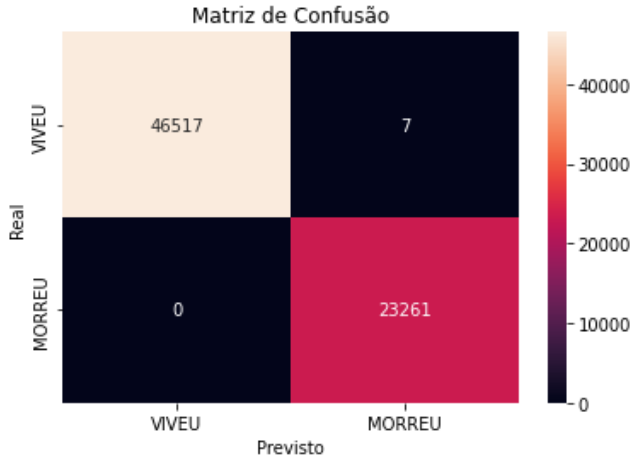

Figura 19 – Matriz de confusão do conjunto de treinamento com o modelo Random Forest com dados balanceados.

2) Teste

- Acurácia: 0.9840660285435892
- Kappa: 0.9645457960301247
- F1: 0.9766229397914564

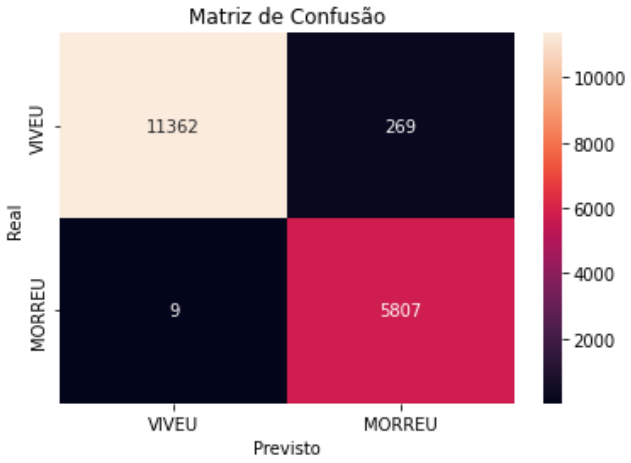

Figura 20 – Matriz de confusão do conjunto de teste com o modelo Random Forest com dados balanceados.

A partir dos resultados acima, identificamos que o modelo *Random Forest* apresentou excelentes resultados, superando os outros dois modelos avaliados até então. Os resultados acima, não só otimizaram a acurácia geral dos dados como contribuíram para uma redução do erro no quadrante mais crítico de nossa matriz de confusão (quando a previsão é de “VIVEU” e o resultado real é “MORREU”). Nas próximas etapas estaremos avaliando a utilização das

técnicas de normalização e de *Grid Search* para avaliar se o resultado acima pode ser superado.

7. Normalização dos dados

A normalização tem como objetivo modificar os dados das colunas numéricas para uma escala comum onde não ocorra distorção nos intervalos de valores. Não são todos os dados que precisam ser normalizados para um modelo de *Machine Learning*, apenas aqueles que apresentam parâmetros com intervalos bem diferentes.

No âmbito dos nossos dados, aplicamos a normalização dos *inputs* via método *StandardScaler* visando impactar positivamente os resultados do modelo através de dados com intervalo de valores comuns. Esta normalização é particularmente importante aqui ao considerar que há milhares de CIDs diferentes considerados nos registros da base, enquanto a idade, por exemplo, varia de 0 a 99 anos.

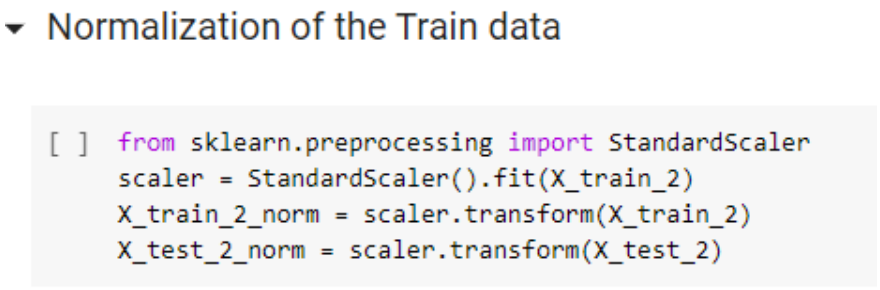

Figura 21 – Comandos utilizados para normalizar os dados.

Após aplicada a normalização sobre a base com melhor balanceamento, observamos que o modelo *Random Forest* continuou apresentando resultados mais satisfatórios diante dos outros dois modelos. As métricas dos modelos após normalização e *oversampling* são resumidas na Tabela 1.

Tabela 1 – Comparação dos modelos Decision Tree, SVM e Random Forest em relação às métricas de acurácia, kappa e F1.

Métricas	**Conjunto de Dados**	**Modelos**		
		Decision Tree	**SVM**	***Random Forest***
Acurácia	Treino	0.9998853621838504	0.8287740918535502	**0.9998996919108691**
	Teste	0.9703673984066028	0.8305725912764372	**0.9839513956554136**
Kappa	Treino	0.9997420833942013	0.6093321714546549	**0.9997743205445172**

	Teste	0.9347723899937403	0.6142564131730381	**0.9673237455580395**
F1-score	Treino	0.9998280679131744	0.7358461368409418	**0.9998495561907627**
	Teste	0.9574380505474603	0.7397429124845922	**0.9784366576819408**

Sobre os resultados apresentados acima, foi possível observar que o modelo *Random Forest* continuou superando os demais ao minimizar o erro geral de suas predições, em especial no que diz respeito aos erros obtidos na base de teste. Sendo assim, elegemos este modelo para o processo de otimização de hiperparâmetros através do *Grid Search*, que será discutido a seguir.

8. Aplicação do *Grid Search* no modelo *Random Forest*

O *Grid Search* é a técnica que utilizamos para obter valores ótimos de hiperparâmetros para o modelo, no nosso caso o *Random Forest*. Essa técnica tem a *sklearn* como biblioteca de origem e permite testar todas as combinações de hiperparâmetros pré-definidos, uma vez que são plotados de forma cartesiana e selecionados através de um problema de minimização de erros.

```
from sklearn.model_selection import GridSearchCV

#dicionario com os parametros de tunning
tuned_parameters = [{'n_estimators': [50, 100, 150, 200],
                     'max_depth': [30, 50, 80, 100],
                     'criterion':['gini','entropy']}]

# Executar o grid search
model_RFC_GS = GridSearchCV(RandomForestClassifier( random_state=seed),\
                            tuned_parameters, scoring='f1')
model_RFC_GS.fit(X_train_2_norm, y_train_2);
```

Figura 22 – Comandos utilizados para executar o Grid Search no melhor modelo obtido.

```
model_RFC_GS.best_params_
```

```
{'criterion': 'gini', 'max_depth': 50, 'n_estimators': 100}
```

Figura 23 – Seleção dos melhores hiperparâmetros após aplicação do Grid Search.

Tabela 2 – Comparação do modelo Random Forest sem e com Grid Search em relação às métricas de acurácia, kappa e F1.

Métricas	**Conjunto de Dados**	**Modelo *Random Forest***	
		Sem *Grid Search*	**Com *Grid Search***
Acurácia	Treino	0.9998996919108691	0.9998853621838504
	Teste	0.9839513956554136	0.9857282054221356
Kappa	Treino	0.9998996919108691	0.9997420833942013
	Teste	0.9839513956554136	0.9682077692816657
F1-score	Treino	0.9998495561907627	0.9998280679131744
	Teste	0.9764587186816882	0.9790139064475348

Após aplicado o método *Grid Search* sobre os parâmetros do modelo *Random Forest,* observamos que os resultados finais foram muito próximos ao *Random Forest* anterior. Este resultado é compreensível quando avaliamos a performance geral do modelo anterior, tanto do ponto de vista do treino quanto do ponto de vista do teste, pois os resultados obtidos já haviam sido muitos satisfatórios.

Como o método *Grid Search* foi setado para buscar aquele conjunto de parâmetros que minimizam o erro sob a perspectiva do *F1-Score*, observamos que o modelo calibrado pelo *Grid Search* obteve um ganho residual através da métrica de performance do *F1-Score* em detrimento da métrica de *Kappa*. De uma maneira geral, ambos os resultados são satisfatórios.

Ainda é importante avaliar se os resultados obtidos são satisfatórios sob o ponto de vista do quadrante mais crítico para o problema, isto é, quando existe predição de “VIVEU” contra o rótulo real de “MORREU”. Abaixo, ilustramos a matriz de confusão para os resultados do modelo *Random Forest* ajustado pelo *Grid Search*, onde é possível observar que este ponto foi muito bem endereçado pelo modelo proposto.

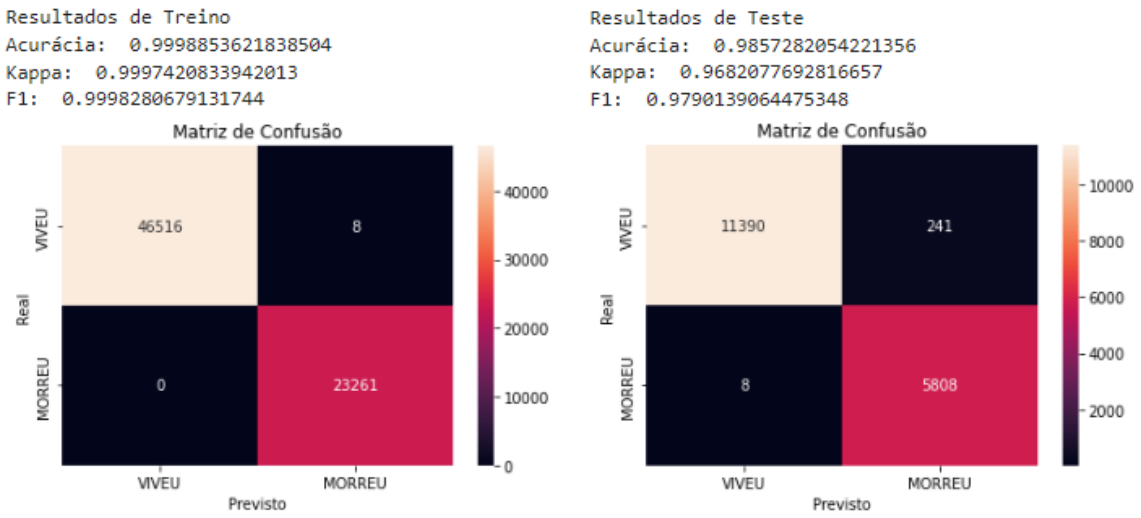

Figura 24 – Resultados obtidos após aplicação do método Grid Search sobre modelo Random Forest.

Os resultados acima estão em linha com o esperado, tendo em vista a performance anteriormente indicada para os parâmetros *Kappa* e *F1-Score*.

9. Conclusões

Este projeto buscou modelar um problema de predição do risco de morte de pacientes internados em estabelecimentos credenciados ao SUS. Neste sentido, foi utilizada a base de dados de morbidades do SIH/SUS (Sistema de Informações Hospitalares do SUS).

Ao longo do processo de modelagem, foi demonstrada a importância da análise exploratória para o processo de refino das variáveis que compõem o universo explicativo de cada amostra. Também foi abordada a importância de outras etapas de pré-processamento, como o *oversampling* e a normalização. Para demonstrar o ganho de eficiência que estas etapas proporcionam ao modelo, foi criado um modelo *baseline* anterior ao pré-processamento. Os resultados obtidos após esta etapa comprovam a relevância do tempo despendido com o pré-processamento da base de dados.

De maneira a minimizar os riscos de predição para o problema proposto, foram testados três modelos distintos de classificação supervisionada: modelo *Decision Tree*, modelo *SVM* e modelo *Random Forest*. Para escolha do melhor modelo, foram avaliadas três métricas de performance sobre as predições de cada um, tanto sobre a base de treino quanto sobre a base teste: acurácia, *Kappa* e *F1-Score*. Após avaliados os resultados destas métricas, foi observado que o modelo *Random Forest* apresentou a melhor performance.

Finalmente, realizamos um teste de calibragem sobre os parâmetros do melhor modelo (*Random Forest*) através do método de *Grid Search*. Os resultados finais obtidos se assemelham aos resultados anteriores à aplicação do método, não havendo ganho substancial.

Cabe observar que, os resultados antes e depois do *Grid Search* foram muito satisfatórios para o problema proposto de predição da mortalidade para pacientes com internações na rede SUS.

Estes resultados ilustram a relevância dos modelos de *Machine Learning* para problemas de classificação supervisionada e lançam luz sobre a oportunidade de utilização desse ferramental para resolução de problemas complexos.