Impacto de técnicas de pré-processamento em algoritmos de classificação

Luiza Ruivo Marinho [1]
Tiago José da Silva [2]

Resumo: Este artigo compara os resultados obtidos através das técnicas de classificação SVM e *Naive Bayes* antes e após o pré-processamento de dados. Foi utilizada uma base de dados disponibilizada pela NASA que classifica binariamente a possibilidade de um asteroide atingir a Terra. Por apresentar um desbalanceamento de classes, o objetivo do pré-processamento foi melhorar a métrica de Sensibilidade (*Recall*) das classificações propostas. Após a redução da dimensionalidade da base de dados com a remoção e alteração de algumas *features*, seleção de atributos por análise de variância, análise de correlação, extração e normalização, foi possível constatar o sucesso das técnicas de pré-processamento citadas elevando de 0% para 91% e 93% a métrica de Sensibilidade (*Recall*) dos classificadores *Naive Bayes* e SVM respectivamente.

***P*alavras-chave:** Pré-processamento de dados, SVM, Naive Bayes, Normalização, Extração, Matriz de confusão, Correlação, Variância, scikit-learn.

1 Introdução

O pré-processamento é uma etapa fundamental para o processo de mineração de dados, o qual envolve tarefas de preparação, organização e estruturação. É um processo determinante para a qualidade final dos dados e que pode impactar modelos de previsão e análise.

Iniciando este artigo, na Seção 2 são apresentados a origem dos dados, linguagem de programação e as ferramentas operadas para o estudo. A Seção 3 apresenta a discussão e a análise de todos os processos. Começa com a compreensão da base de dados, seguindo pela classificação antes do pré-processamento para futuras comparações e uma explanação sobre o funcionamento dos algoritmos SVM e *Naive Bayes*. Concluindo a Seção, foi dissertado algumas comparações dos resultados finais. A conclusão final deste artigo é expressada na Seção 4.

2 Metodologia

Para este estudo de caso, utilizou-se uma base com dados coletados pela API[3] do *CEO - Center for Near-Earth Object Studies*[4], da NASA, a qual foi disponibilizada no site Kaggle[5]. Python foi a linguagem de programação escolhida, operado dentro da ferrameta Jupyter Notebook[6]. Para treino e teste da base,

[1]Setor de Educação Profissional e Tecnológica, UFPR
{luizamarinho@ufpr.br}
[2]Setor de Educação Profissional e Tecnológica, UFPR
{tiago.silva1@ufpr.br}
[3]Application Programming Interface.
[4]Centro de Estudos de Objetos Próximos à Terra - Tradução Livre.
[5]Base de dados disponível em https://www.kaggle.com/shrutimehta/nasa-asteroids-classification
[6]https://jupyter.org/

foi utilizada a biblioteca Scikit-learn[7]. Outras bibliotecas utilizadas foram o Pandas[8] e o Numpy[9] para manipulação de dados, e Matplotlib[10] e Seaborn[11] para análise gráfica.

As técnicas de pré-processamento aplicadas visaram melhorar a métrica de Sensibilidade (*Recall*) nas classificações SVM e *Naive Bayes*. Para exemplificar isso, foi adotada uma metodologia própria com as seguintes etapas:

- Compreensão da base de dados;
- Classificação preliminar ao pré-processamento;
- Pré-processamento;
- Novas classificações;
- Análise dos resultados anteriores e posteriores ao pré-processamento.

3 Discussão e Análise de Resultados

3.1 Compreensão da base de dados

A base possui 4687 observações, com 40 atributos (Tabela 1), sendo 39 *features* e 1 classe. As observações estão divididas em duas classes, sendo "True" representando asteroides com risco de colisão e "False" representando asteroides sem risco de colisão. O atributo que representa a classe é o "Hazardous". A base não possui dados faltantes.

```
import pandas as pd

neo_df = pd.read_csv('nasa-asteroides-database.csv', sep = ',', engine = 'python',
                           parse_dates=['Close Approach Date', 'Orbit Determination
    Date'])
```

Código 1: Carregamento da base dados

Atributo	Definição	Tipo de dado
Neo Reference ID	ID de referência do Near Earth Object (NEO) para um asteroide (ou cometa) que se aproxima da Terra	Categórico
Name	Nome do asteroide (o mesmo que 'Neo Reference ID')	Categórico
Absolute Magnitude	Medida da luminosidade do asteroide (em H)	Contínuo
Est Dia in KM(min)	Diâmetro mínimo estimado do asteroide em KM	Contínuo
Est Dia in KM(max)	Diâmetro máximo estimado do asteroide em KM	Contínuo
Est Dia in M(min)	Diâmetro mínimo estimado do asteroide em Metros	Contínuo
Est Dia in M(max)	Diâmetro máximo estimado do asteroide em Metros	Contínuo
Est Dia in Miles(min)	Diâmetro mínimo estimado do asteroide em Milhas	Contínuo
Est Dia in Miles(max)	Diâmetro máximo estimado do asteroide em Milhas	Contínuo
Est Dia in Feet(min)	Diâmetro mínimo estimado do asteroide em Pés	Contínuo
Est Dia in Feet(max)	Diâmetro máximo estimado do asteroide em Pés	Contínuo

[7]https://scikit-learn.org/
[8]https://pandas.pydata.org/
[9]https://numpy.org/
[10]https://matplotlib.org/
[11]https://seaborn.pydata.org/

Atributo	Definição	Tipo de dado
Close Approach Date	Data em que o asteroide se aproxima da Terra	Contínuo
Epoch Date Close Approach	Data em que o asteroide se aproxima da Terra (em tempo de época)	Discreto
Relative Velocity km per sec	Velocidade do asteroide em relação à Terra em km por segundo	Contínuo
Relative Velocity km per hr	Velocidade do asteroide em relação à Terra em km por hora	Contínuo
Miles per hour	Velocidade do asteroide em relação à Terra em milhas por hora	Contínuo
Miss Dist.(Astronomical)	Distância pela qual o asteroide erra a Terra em medida astronômica	Contínuo
Miss Dist.(lunar)	Distância pela qual o asteroide erra a Terra em medida lunar	Contínuo
Miss Dist.(kilometers)	Distância pela qual o asteroide erra a Terra em km	Contínuo
Miss Dist.(miles)	Distância pela qual o asteroide erra a Terra em milhas	Contínuo
Orbiting Body	Nome do corpo orbitado	Categórico
Orbit ID	Id da órbita	Categórico
Orbit Determination Date	Data em que a órbita do asteroide foi determinada	Discreto
Orbit Uncertainity	Uma medida da incerteza (’erros de medição’) na órbita calculada	Discreto
Minimum Orbit Intersection	A distância mais próxima entre a Terra e o asteroide em suas respectivas órbitas (em unidades astronômicas)	Contínuo
Jupiter Tisserand Invariant	Um valor usado para diferenciar asteroides e cometas da família de Júpiter	Contínuo
Epoch Osculation	Instância de tempo em que a posição do asteroide e os vetores de velocidade são especificados	Discreto
Eccentricity	Um valor que especifica o quanto a órbita do asteroide se desvia de um círculo perfeito	Contínuo
Semi Major Axis	O raio mais longo de uma órbita elíptica; uma medida da distância média do asteroide ao Sol (asteroides orbitam o Sol)	Contínuo
Inclination	Mede a inclinação da órbita do asteroide em torno do Sol	Contínuo
Asc Node Longitude	Ângulo no plano eclíptico entre o eixo X da estrutura inercial e a linha que passa pelo nó ascendente	Contínuo
Orbital Period	Tempo necessário para o asteroide completar uma única órbita ao redor do Sol	Contínuo
Perihelion Distance	Distância do ponto na órbita do asteroide que está mais próximo do Sol	Contínuo
Perihelion Arg	O ângulo (no plano da órbita do corpo) entre a linha do nó ascendente e o periélio medido na direção da órbita do corpo	Contínuo
Aphelion Dist	Distância do ponto na órbita do asteroide que está mais distante do Sol	Contínuo
Perihelion Time	Período de tempo de passagem do asteroide pelo estágio do periélio	Contínuo
Mean Anomaly	O produto do movimento médio de um corpo em órbita e o tempo após a passagem do periélio	Contínuo

Atributo	Definição	Tipo de dado
Mean Motion	A velocidade angular necessária para um corpo fazer uma órbita em torno de uma elipse ideal com um semi-eixo principal específico	Contínuo
Equinox	Um padrão astronômico para medição (atualmente J2000.0)	Categórico
Hazardous	Campo binário (True ou False) que determina se o asteróide tem risco de colidir com a Terra.	Categórico

Tabela 1: Descrição da base de dados(1)

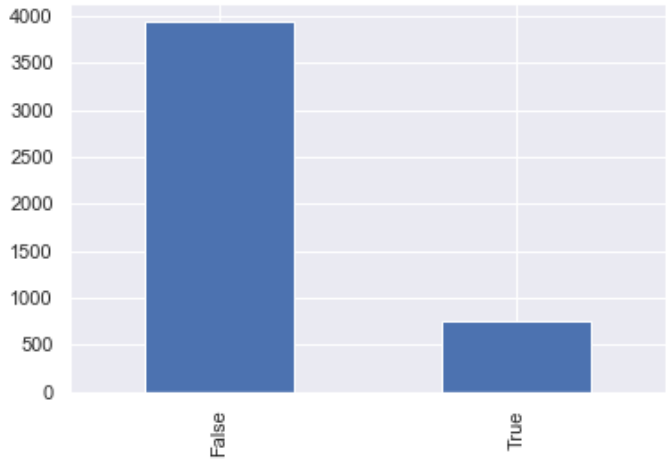

Figura 1: Distribuição da classe *Hazardous*

A base possui problema de desbalanceamento concentrando 84% das observações na classe "True"e os 16% restantes na classe "False".

3.2 Classificação preliminar ao pré-processamento

Por característica dos algoritmos de SVM e *Naive Bayes* implementados na biblioteca *scikit-learn*, e utilizados nesse estudo, algumas transformações de pré-processamento foram necessárias. Isso acontece porque as implementações em questão aceitam apenas *features* numéricas para a construção do modelo enquanto que a base de dados estudada possui dois campos categóricos no formato texto e dois campos contínuos que contém datas no padrão ISO 8601.

Orbit Determination Date	Close Approach Date	Orbiting Body	Equinox
2017-04-06 08:36:37	1995-01-01	Earth	J2000
2017-04-06 08:32:49	1995-01-01	Earth	J2000
2017-04-06 09:20:19	1995-01-08	Earth	J2000
2017-04-06 09:15:49	1995-01-15	Earth	J2000
2017-04-06 08:57:58	1995-01-15	Earth	J2000

Tabela 2: *Features* não numéricas presentes na base de dados

Uma vez que o objetivo desse estudo foi comparar os resultados de classificações com e sem pré-processamento de dados, não serão abordadas nessa etapa técnicas além das necessárias para a execução dos algoritmos de aprendizagem citados.

3.2.1 Codificação A codificação consiste em transformar dados presentes no formato texto em dados numéricos mantendo a categorização do dado. Essa técnica foi aplicada nas *features* "Orbiting Body"e "Equinox"da base.

```
orbiting_body_factor = pd.factorize(neo_df['Orbiting Body'])
neo_df['Orbiting Body'] = orbiting_body_factor[0]

equinox_factor = pd.factorize(neo_df['Equinox'])
neo_df['Equinox'] = equinox_factor[0]
```

Código 2: Codificação das *features* "Orbiting Body"e "Equinox"

As codificações feitas foram:

Feature	Valor Original	Valor Codificado
Orbiting Body	Earth	0
Equinox	J2000	0

Tabela 3: Resultado da codificação de "Orbiting Body"e "Equinox"

Percebe-se que ambas as *features* possuem apenas um valor possível, porém essa característica da base foi tratada na etapa dedicada ao pré-processamento uma vez que, como dito anteriormente, o objetivo desse tratamento prévio foi apenas permitir a execução dos algoritmos de classificação.

3.2.2 Exclusão de features de data Os campos de data, por serem pouco significativos, foram excluídos da base de dados.

```
neo_df.drop('Orbit Determination Date', 1, inplace = True)
neo_df.drop('Close Approach Date', 1, inplace = True)
```

Código 3: Exclusão das *features* de data

Após os tratamentos feitos e descritos, a base passa a possuir 37 *features* e uma classe. São elas:

	Feature
1	Neo Reference ID
2	Name
3	Absolute Magnitude
4	Est Dia in KM(min)
5	Est Dia in KM(max)
6	Est Dia in M(min)
7	Est Dia in M(max)
8	Est Dia in Miles(min)
9	Est Dia in Miles(max)
10	Est Dia in Feet(min)
11	Est Dia in Feet(max)
12	Epoch Date Close Approach
13	Relative Velocity km per sec

	Feature
14	Relative Velocity km per hr
15	Miles per hour
16	Miss Dist.(Astronomical)
17	Miss Dist.(lunar)
18	Miss Dist.(kilometers)
19	Miss Dist.(miles)
20	Orbiting Body
21	Orbit ID
22	Orbit Uncertainity
23	Minimum Orbit Intersection
24	Jupiter Tisserand Invariant
25	Epoch Osculation
26	Eccentricity
27	Semi Major Axis
28	Inclination
29	Asc Node Longitude
30	Orbital Period
31	Perihelion Distance
32	Perihelion Arg
33	Aphelion Dist
34	Perihelion Time
35	Mean Anomaly
36	Mean Motion
37	Equinox

Tabela 4: Lista de *features* após a remoção de "Orbiting Body"e "Equinox"

3.2.3 Separação de dados para treino e teste A base original com 4687 observações foi dividida em dois *datasets* cada um contendo 80% e 20% dos dados da base original, respectivamente. O propósito dessa divisão foi usar o primeiro subconjunto de dados (com 80% das observações da base original) no processo de treino de modelo e destinar o segundo subconjunto (com 20% das observações da base original) para os testes.

```
from sklearn.model_selection import train_test_split

# Cria um dataframe para as features
neo_features_df = neo_df.iloc[:,:-1]

# Cria um dataframe para as classes
neo_classes_df = neo_df.iloc[:,-1]

# Divide as observacoes entre treino e teste
features_train, features_test, classes_train, classes_test = \
train_test_split(neo_features_df, neo_classes_df, test_size = 0.2, random_state = 1,
    stratify = neo_classes_df)
```

Código 4: Separação de dados entre treino e teste

Por conta do desbalanceamento de classes a divisão de dados precisou respeitar a proporcionalidade de cada uma nos subconjuntos criados. A função *train_test_split* presente no *scikit-learn* e utilizada no código acima possui o atributo *stratify* que garante a manutenção da proporcionalidade original.

	Base original		Subconjunto de treino		Subconjunto de teste	
Classe	Quantidade	%	Quantidade	%	Quantidade	%
True	755	16	604	16	151	16
False	3932	84	3145	84	787	84

Tabela 5: Quantidades de observações em cada conjunto de dados

3.2.4 Classificação com SVM *Support Vector Machines*, ou simplesmente SVM, é um algoritmo de classificação supervisionada cujo objetivo consiste na separação ótima de classes (2). É a abordagem pré-fabricada mais popular para aprendizagem supervisionada e indicada para quando não há nenhum conhecimento prévio especializado sobre um domínio (3).

Em classificação binária a técnica SVM aprende a partir de um conjunto de *features* e classes correspondentes. Após o treino, o algoritmo se torna capaz de classificar novos dados.

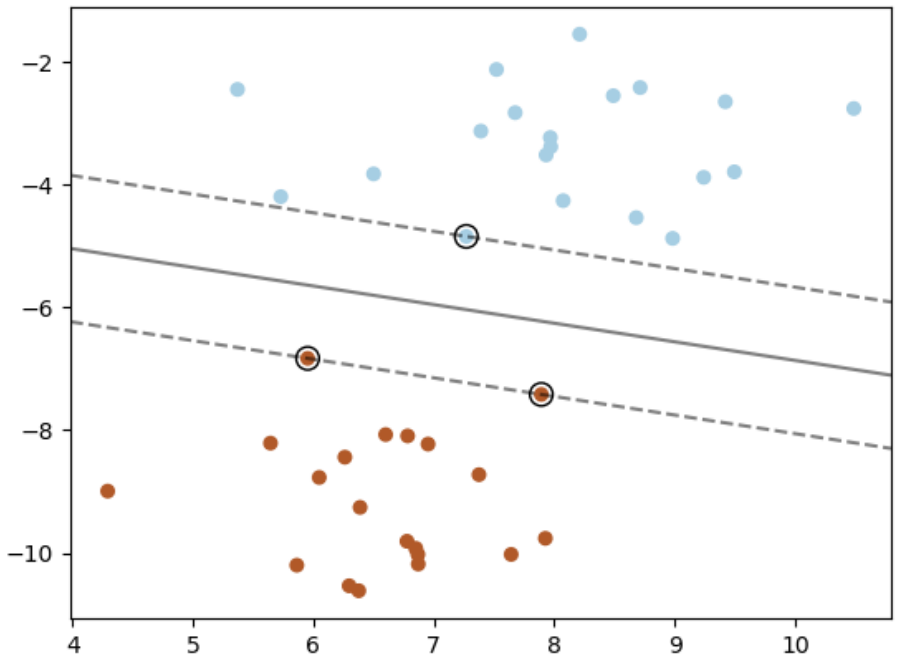

Figura 2: Exemplo de classificação SVM que separou a classe vermelha da classe azul. (4)

Após treino do modelo via SVM a classificação do subconjunto de testes alcançou os seguintes resultados:

```
from sklearn import svm

clf = svm.SVC()
clf.fit(features_train, classes_train)
```

```
5
6 predicted_svm = clf.predict(features_test)
```

Código 5: Classificador SVM

```
1  from sklearn.metrics import confusion_matrix
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import seaborn as sn
5
6  cm = confusion_matrix(classes_test, predicted_svm)
7
8  cm_df = pd.DataFrame(cm, columns = np.unique(classes_test), index = np.unique(
       classes_test))
9  cm_df.index.name = 'Valor real'
10 cm_df.columns.name = 'Valor predito'
11 plt.figure(figsize = (10,5))
12 sn.set(font_scale = 1.3)
13 sn.heatmap(cm_df, cmap="Blues", annot=True, annot_kws={"size": 16}, fmt='g')
```

Código 6: Geração da matriz de confusão da classificação SVM

```
1  from sklearn.metrics import accuracy_score
2  from sklearn.metrics import precision_score
3  from sklearn.metrics import recall_score
4  from sklearn.metrics import f1_score
5
6  print('Acuracia: %.2f'
7        % accuracy_score(classes_test, predicted_svm))
8  print('Precisao: %.2f'
9        % precision_score(classes_test, predicted_svm, zero_division = True))
10 print('Recall: %.2f'
11       % recall_score(classes_test, predicted_svm, zero_division = True))
12 print('F-sore: %.2f'
13       % f1_score(classes_test, predicted_svm, zero_division = True))
```

Código 7: Resumo das métricas do classificador SVM

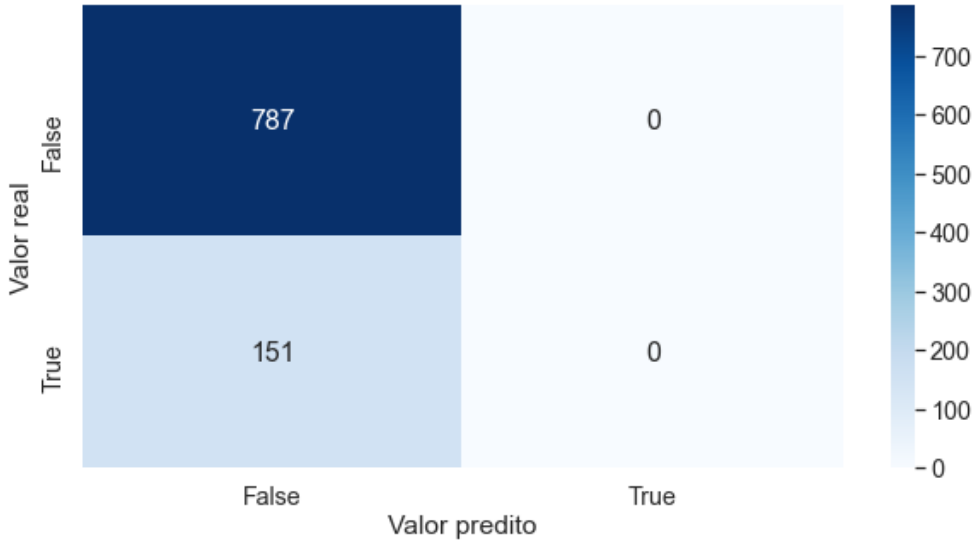

Figura 3: Matriz de confusão gerada a partir do classificador SVM

Métrica	Valor
Acurácia	84%
Precisão	100%
Sensibilidade (Recall)	0%
F-Score	0%

Tabela 6: Métricas alcançadas pelo classificador SVM

3.2.5 Classificação com *Naive Bayes* O classificador *Naive Bayes* se baseia no teorema de Bayes (criado por Thomas Bayes no século XVIII) e sua principal característica é assumir que as características do conjunto de dados são fortemente independentes entre si.(5). A partir das *features*, que são consideradas independentes, é calculado um modelo probabilístico que permite a classificação entre as classes disponíveis.

Após treino do modelo via *Naive Bayes* a classificação do subconjunto de testes alcançou os seguintes resultados:

```
from sklearn.naive_bayes import GaussianNB

nb_classifier = GaussianNB()
nb_model = nb_classifier.fit(features_train, classes_train)
predicted_nb = nb_model.predict(features_test)
```

Código 8: Classificador *Naive Bayes*

```
cm = confusion_matrix(classes_test, predicted_nb)

cm_df = pd.DataFrame(cm, columns = np.unique(classes_test), index = np.unique(
    classes_test))
cm_df.index.name = 'Valor real'
cm_df.columns.name = 'Valor predito'
```

```
plt.figure(figsize = (10,5))
sn.set(font_scale = 1.3)
sn.heatmap(cm_df, cmap="Blues", annot=True, annot_kws={"size": 16}, fmt='g')
```

Código 9: Geração da matriz de confusão da classificação *Naive Bayes*

```
print('Acuracia: %.2f'
      % accuracy_score(classes_test, predicted_nb))
print('Precisao: %.2f'
      % precision_score(classes_test, predicted_nb, zero_division = True))
print('Recall: %.2f'
      % recall_score(classes_test, predicted_nb, zero_division = True))
print('F-sore: %.2f'
      % f1_score(classes_test, predicted_nb, zero_division = True))
```

Código 10: Resumo das métricas do classificador *Naive Bayes*

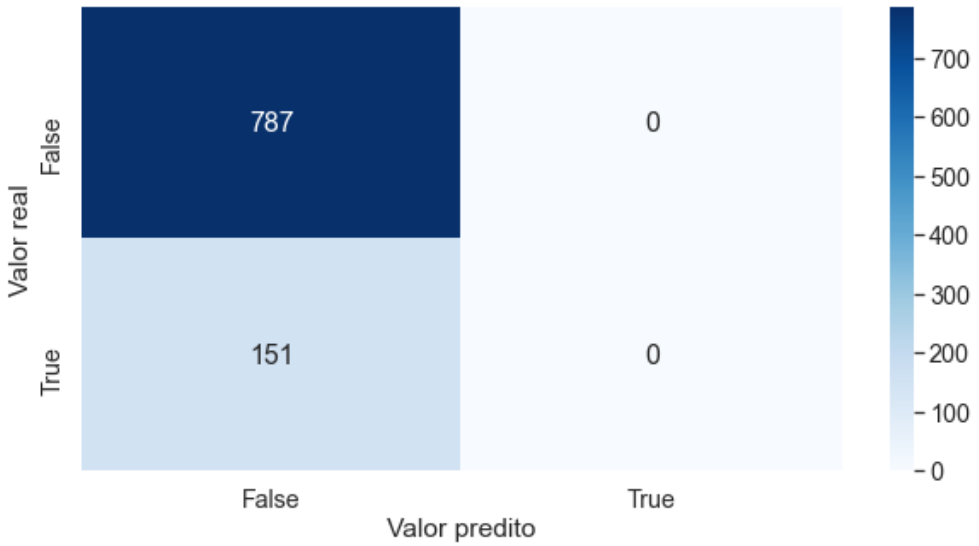

Figura 4: Matriz de confusão gerada a partir do classificador *Naive Bayes*

Métrica	Valor
Acurácia	84%
Precisão	100%
Sensibilidade (Recall)	0%
F-Score	0%

Tabela 7: Métricas alcançadas pelo classificador *Naive Bayes*

3.2.6 Análise de Resultados Ambos os classificadores apresentados aqui, SVM e *Naive Bayes*, apresentaram a mesma matriz de confusão com os mesmos valores para as métricas de Acurácia, Precisão, Sensibilidade (*Recall*) e F-Score.

Embora a acurácia e a precisão apresentem valores altos, 84% e 100% respectivamente, tais mé-

tricas não são significativas no modelo por conta do desbalanceamento de classes presente nos dados estudados.

Ambos os modelos conseguiram classificar todos os asteroides não perigosos (classe FALSE) com sucesso, justificando a precisão de 100%, porém se mostrou ineficiente ao não conseguir classificar nenhum asteroide perigoso, o que justificou uma sensibilidade (ou *recall*) de 0%.

Considerando a gravidade da classificação de falsos negativos (asteroides perigosos classificados como não perigosos), esse estudo buscou uma sensibilidade alta em detrimento da precisão.

3.3 Pré-processamento

O pré-processamento não é uma etapa isolada e conclusiva por si só, mas um processo necessário que pode ser repetido várias vezes durante um processo maior de mineração de dados.

Nesse estudo as primeiras técnicas de pré-processamento foram feitas como pré-requisito para executar as primeiras classificações. Foram elas a Codificação e a Exclusão de atributos (seções 3.2.1 e 3.2.2 respectivamente). Com o objetivo de melhorar as métricas alcançadas, outras técnicas de pré-processamento foram executadas e são descritas a seguir.

3.3.1 Redução de dimensionalidade Após a etapa de codificação e remoção de features descritas nas etapas anteriores a base de dados passou a possuir 37 *features* que tentam explicar a classe de cada observação.

A redução de dimensionalidade, também conhecida como redução de dados vertical, é uma tarefa do pré-processamento que vem ganhando importância por reduzir a quantidade de variáveis ou atributos aleatórios de acordo com o objetivo da tarefa (6). Esta redução se faz importante em casos onde as bases de dados possuem o número de atributos (características) muito maior do que a quantidade de instâncias (amostras) (7).

Destacam-se duas técnicas de redução de dimensionalidade:

- Seleção de atributos: Busca o subconjunto de *features* da base principal que melhor representam a totalidade.
- Extração de atributos: Gera uma nova *feature* com o poder de substituir duas ou mais *features* da base original.

3.3.2 Remoção de *Features* identificadoras (*Ids*) *Ids* são atributos que não representam características mas servem como identificadores e chaves de relacionamento entre dados. Por essa razão não possuem significância para a construção do modelo e podem ser excluídas. Na base estudada tais campos são:

- Neo Reference ID
- Name
- Orbit ID

```
neo_df = neo_df.drop(columns = ['Neo Reference ID', 'Name', 'Orbit ID'])
```

Código 11: Remoção de *Ids*

Ao final dessa remoção o conjunto de features passou a ser definido por:

	Feature
1	Absolute Magnitude
2	Est Dia in KM(min)
3	Est Dia in KM(max)
4	Est Dia in M(min)
5	Est Dia in M(max)
6	Est Dia in Miles(min)
7	Est Dia in Miles(max)
8	Est Dia in Feet(min)
9	Est Dia in Feet(max)
10	Epoch Date Close Approach
11	Relative Velocity km per sec
12	Relative Velocity km per hr
13	Miles per hour
14	Miss Dist.(Astronomical)
15	Miss Dist.(lunar)
16	Miss Dist.(kilometers)
17	Miss Dist.(miles)
18	Orbiting Body
19	Orbit Uncertainity
20	Minimum Orbit Intersection
21	Jupiter Tisserand Invariant
22	Epoch Osculation
23	Eccentricity
24	Semi Major Axis
25	Inclination
26	Asc Node Longitude
27	Orbital Period
28	Perihelion Distance
29	Perihelion Arg
30	Aphelion Dist
31	Perihelion Time
32	Mean Anomaly
33	Mean Motion
34	Equinox

Tabela 8: Lista de *features* após a remoção de *Ids*

3.3.3 Seleção de atributos por análise de variância *Features* com 0 variância representam dados constantes que não interferem no resultado da classificação, por isso foram excluídas do conjunto de dados.

```
neo_df.var().sort_values()
```

Código 12: Listagem da variância das *features*

Feature	Variância
Orbiting Body	0.000
Equinox	0.000
Minimum Orbit Intersection	0.008

Feature	Variância
Miss Dist.(Astronomical)	0.021
Eccentricity	0.033

Tabela 9: 5 features com menor variância

```
neo_df = neo_df.drop(columns = ['Orbiting Body',
                                'Equinox'])
```

Código 13: Remove as *features* "Orbiting Body"e "Equinox"

Ao final dessa remoção o conjunto de *features* passou a ser:

	Feature
1	Absolute Magnitude
2	Est Dia in KM(min)
3	Est Dia in KM(max)
4	Est Dia in M(min)
5	Est Dia in M(max)
6	Est Dia in Miles(min)
7	Est Dia in Miles(max)
8	Est Dia in Feet(min)
9	Est Dia in Feet(max)
10	Epoch Date Close Approach
11	Relative Velocity km per sec
12	Relative Velocity km per hr
13	Miles per hour
14	Miss Dist.(Astronomical)
15	Miss Dist.(lunar)
16	Miss Dist.(kilometers)
17	Miss Dist.(miles)
18	Orbit Uncertainity
19	Minimum Orbit Intersection
20	Jupiter Tisserand Invariant
21	Epoch Osculation
22	Eccentricity
23	Semi Major Axis
24	Inclination
25	Asc Node Longitude
26	Orbital Period
27	Perihelion Distance
28	Perihelion Arg
29	Aphelion Dist
30	Perihelion Time
31	Mean Anomaly
32	Mean Motion

Tabela 10: Lista de *features* após a remoção de "Orbiting Body"e "Equinox"

3.3.4 Seleção de atributos por análise de correlação Em Estatística correlação indica que duas variáveis estão relacionadas e se movem juntas. A correlação pode ser positiva, quando ambas se movem no mesmo sentido. A correlação também pode ser negativa, quando uma variável se move em sentido oposto à outra. A correlação é dita perfeita quando corresponde à 100%.

```
sn.set(rc={'figure.figsize':(30,20)})
sn.set(font_scale=1.2)
sn.heatmap(neo_df.corr(), vmin=-1, vmax=1, cmap="Spectral", annot=True)
plt.show()
plt.close()
```

Código 14: Exibe mapa de calor da correlação entre as *features*

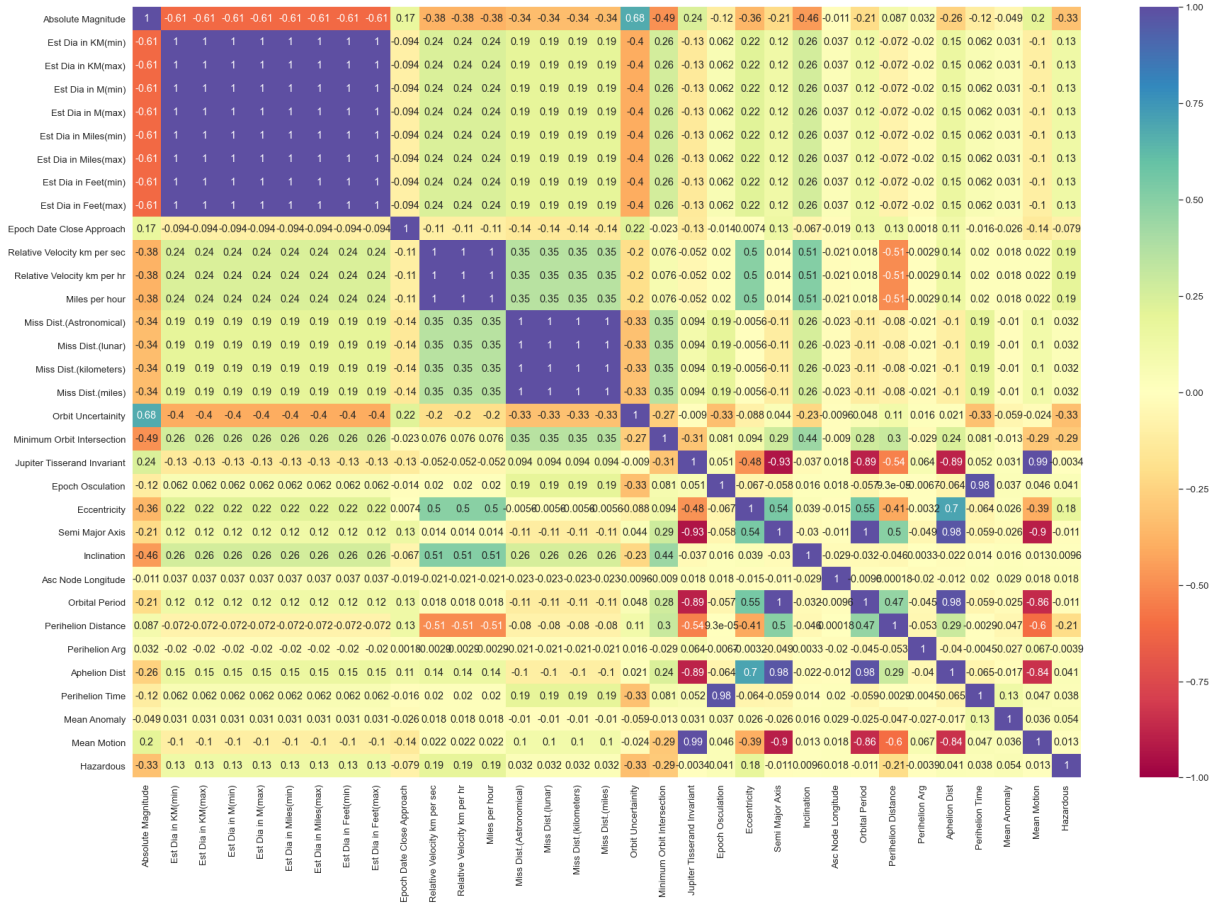

Figura 5: Mapa de calor referente à correlação de *features*

Ao destacar as variáveis com correlação positiva perfeita percebe-se que elas se referem às *features* de diâmetro estimado, distância e velocidade relativa presentes em mais de uma unidade de medida conforme exemplificado a seguir.

```
sn.set(rc={'figure.figsize':(40,30)})
sn.set(font_scale=3)
sn.heatmap(neo_df[[
```

```
    'Est Dia in KM(min)',
    'Est Dia in KM(max)',
    'Est Dia in M(min)',
    'Est Dia in M(max)',
    'Est Dia in Miles(min)',
    'Est Dia in Miles(max)',
    'Est Dia in Feet(min)',
    'Est Dia in Feet(max)',
    'Relative Velocity km per sec',
    'Relative Velocity km per hr',
    'Miles per hour',
    'Miss Dist.(Astronomical)',
    'Miss Dist.(lunar)',
    'Miss Dist.(kilometers)',
    'Miss Dist.(miles)']].corr(), vmin=-1, vmax=1, cmap="Spectral", annot=True)
plt.show()
plt.close()
```

Código 15: Exibe mapa de calor para a correlação entre as *features* selecionadas

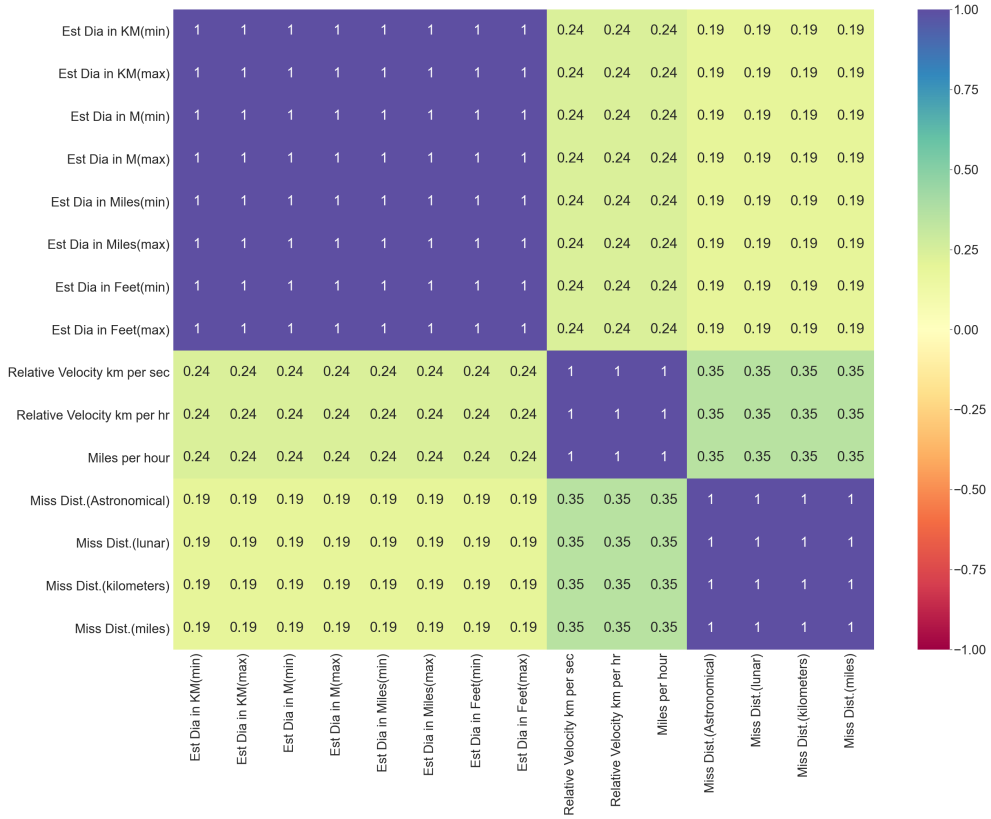

Figura 6: Mapa de calor referente às correlações positivas perfeitas

Por essa razão foi escolhida uma única unidade de medida excluindo as demais, diminuindo a dimensionalidade da base.

Abaixo mostra-se o que foi feito com cada uma das *features*:

Feature	Ação tomada
Est Dia in KM(min)	Mantida
Est Dia in KM(max)	Mantida
Est Dia in M(min)	Excluída
Est Dia in M(max)	Excluída
Est Dia in Miles(min)	Excluída
Est Dia in Miles(max)	Excluída
Est Dia in Feet(min)	Excluída
Est Dia in Feet(max)	Excluída
Relative Velocity km per hr	Mantida
Relative Velocity km per sec	Excluída
Miles per hour	Excluída
Miss Dist.(Astronomical)	Mantida
Miss Dist.(lunar)	Excluída
Miss Dist.(kilometers)	Excluída
Miss Dist.(miles)	Excluída

Tabela 11: Acão tomada para cada *feature* com correlação positiva perfeita

```
neo_df = neo_df.drop(columns = ['Est Dia in M(min)',
                                'Est Dia in M(max)',
                                'Est Dia in Miles(min)',
                                'Est Dia in Miles(max)',
                                'Est Dia in Feet(min)',
                                'Est Dia in Feet(max)',
                                'Relative Velocity km per sec',
                                'Miles per hour',
                                'Miss Dist.(lunar)',
                                'Miss Dist.(kilometers)',
                                'Miss Dist.(miles)'
                                ])
```

Código 16: Remove as *features* com correlação positiva perfeita

Ao final dessa etapa o conjunto de *features* passou a ser definido por:

	Feature
1	Absolute Magnitude
2	Est Dia in KM(min)
3	Est Dia in KM(max)
4	Epoch Date Close Approach
5	Relative Velocity km per hr
6	Miss Dist.(Astronomical)
7	Orbit Uncertainity
8	Minimum Orbit Intersection
9	Jupiter Tisserand Invariant
10	Epoch Osculation
11	Eccentricity
12	Semi Major Axis
13	Inclination
14	Asc Node Longitude
15	Orbital Period

	Feature
16	Perihelion Distance
17	Perihelion Arg
18	Aphelion Dist
19	Perihelion Time
20	Mean Anomaly
21	Mean Motion

Tabela 12: Lista de *features* após a remoção dos atributos com variância positiva perfeita

3.3.5 Extração de atributos O diâmetro estimado está representado em duas *features*, a primeira que descreve o diâmetro mínimo - *Est Dia in KM (min)* - e a segunda que descreve o diâmetro máximo - *Est Dia in KM (max)*. Ambas podem ser substituídas por uma nova *feature* que represente a média simples entre elas.

```
# Calcula e adiciona a media ao conjunto de dados.
neo_df['Est Dia in KM(mean)'] = neo_df[['Est Dia in KM(min)', 'Est Dia in KM(max)']].
    mean(axis = 1)

# Reposiciona os elementos do dataframe a fim de manter a classe na ultima posicao.
neo_df = neo_df[['Absolute Magnitude',
                 'Est Dia in KM(mean)',
                 'Epoch Date Close Approach',
                 'Relative Velocity km per hr',
                 'Miss Dist.(Astronomical)',
                 'Orbit Uncertainity',
                 'Minimum Orbit Intersection',
                 'Jupiter Tisserand Invariant',
                 'Epoch Osculation',
                 'Eccentricity',
                 'Semi Major Axis',
                 'Inclination',
                 'Asc Node Longitude',
                 'Orbital Period',
                 'Perihelion Distance',
                 'Perihelion Arg',
                 'Aphelion Dist',
                 'Perihelion Time',
                 'Mean Anomaly',
                 'Mean Motion',
                 'Hazardous'
                ]]
```

Código 17: Extração de *feature*

Ao final dessa extração o conjunto de *features* passou a ser definido por:

	Feature
1	Absolute Magnitude
2	Est Dia in KM(mean)
3	Epoch Date Close Approach
4	Relative Velocity km per hr
5	Miss Dist.(Astronomical)
6	Orbit Uncertainity
7	Minimum Orbit Intersection
8	Jupiter Tisserand Invariant

	Feature
9	Epoch Osculation
10	Eccentricity
11	Semi Major Axis
12	Inclination
13	Asc Node Longitude
14	Orbital Period
15	Perihelion Distance
16	Perihelion Arg
17	Aphelion Dist
18	Perihelion Time
19	Mean Anomaly
20	Mean Motion

Tabela 13: Lista de *features* após a extração do campo *Est Dia in KM(mean)*

3.3.6 Remoção de atributos com baixa correlação com a classe Ao analisar a correlação entre todos os atributos com a respectiva classe é possível inferir a importância de cada um para a classificação. Foram ignoradas todas as *features* com correlação menor que 0.1 (10%).

```
neo_df.corr()[['Hazardous']].abs().sort_values(by = 'Hazardous')
```

Código 18: Lista o valor de correlação de cada *feature* com a classe

Feature	Correlação com Hazardous
Jupiter Tisserand Invariant	0.003
Perihelion Arg	0.004
Inclination	0.01
Semi Major Axis	0.011
Orbital Period	0.011
Mean Motion	0.013
Asc Node Longitude	0.018
Miss Dist.(Astronomical)	0.032
Perihelion Time	0.038
Aphelion Dist	0.041
Epoch Osculation	0.041
Mean Anomaly	0.054
Epoch Date Close Approach	0.079
Est Dia in KM(mean)	0.132
Eccentricity	0.183
Relative Velocity km per hr	0.192
Perihelion Distance	0.207
Minimum Orbit Intersection	0.289
Absolute Magnitude	0.326
Orbit Uncertainity	0.329

Tabela 14: Correlação entre as features e a classe

```
neo_df = neo_df[['Est Dia in KM(mean)',
```

```
                'Eccentricity',
                'Relative Velocity km per hr',
                'Perihelion Distance',
                'Minimum Orbit Intersection',
                'Absolute Magnitude',
                'Orbit Uncertainity',
                'Hazardous'
               ]]
```

Código 19: Seleciona as *features* mais correlacionadas com a classe

Ao final dessa etapa o conjunto de *features* foi resumido a:

	Feature
1	Est Dia in KM(mean)
2	Eccentricity
3	Relative Velocity km per hr
4	Perihelion Distance
5	Minimum Orbit Intersection
6	Absolute Magnitude
7	Orbit Uncertainity

Tabela 15: Lista de *features* após a seleção das mais correlacionadas com a classe

3.3.7 Normalização Normalização consiste em transformar atributos de diferentes valores para uma escala menor e padrão (5).

Nessa etapa foi utilizada a normalização *Standard* presente no framework *Scikit-Learn*.

```
from sklearn.preprocessing import scale

neo_df['Est Dia in KM(mean)'] = scale(neo_df['Est Dia in KM(mean)'])
neo_df['Eccentricity'] = scale(neo_df['Eccentricity'])
neo_df['Relative Velocity km per hr'] = scale(neo_df['Relative Velocity km per hr'])
neo_df['Perihelion Distance'] = scale(neo_df['Perihelion Distance'])
neo_df['Minimum Orbit Intersection'] = scale(neo_df['Minimum Orbit Intersection'])
neo_df['Absolute Magnitude'] = scale(neo_df['Absolute Magnitude'])
neo_df['Orbit Uncertainity'] = scale(neo_df['Orbit Uncertainity'])
```

Código 20: Executa normalização *Standard* nas *features*

Essa etapa não gerou atualização na lista de *features*.

3.4 Análise de Resultados

Após as tarefas de pré-processamento novas classificações foram feitas afim de comparar as novas métricas com as alcançadas previamente ao pré-processamento.

Antes das novas predições a lista de *features* foi atualizada e novamente foi feita a divisão da base de dados em subconjuntos de treino e teste.

```
# Atualiza a lista de features
features = neo_df.columns.tolist()
features = features[0:-1]

# Cria dataframes para as features e para as classes
```

```
neo_features_df = neo_df[features]
neo_classes_df = neo_df.iloc[:,-1]

# Divide a base entre treinamento e teste
features_train, features_test, classes_train, classes_test = \
train_test_split(neo_features_df, neo_classes_df, test_size = 0.2, random_state = 1,
    stratify = neo_classes_df)
```

Código 21: Divisão da base para treino e teste

3.4.1 Classificação com SVM

Os resultados alcançados foram:

```
clf = svm.SVC()
clf.fit(features_train, classes_train)
predicted_svm_final = clf.predict(features_test)
```

Código 22: Execução da classificação SVM após o pré-processamento

```
cm = confusion_matrix(classes_test, predicted_svm_final)

cm_df = pd.DataFrame(cm, columns = np.unique(classes_test), index = np.unique(
    classes_test))
cm_df.index.name = 'Valor real'
cm_df.columns.name = 'Valor predito'
plt.figure(figsize = (10, 5))
sn.set(font_scale = 1.3)
sn.heatmap(cm_df, cmap="Blues", annot=True, annot_kws={"size": 16}, fmt='g')
```

Código 23: Geração da matriz de confusão da classificação SVM após o pré-processamento

```
print('Acuracia: %.2f'
      % accuracy_score(classes_test, predicted_svm_final))
print('Precisao: %.2f'
      % precision_score(classes_test, predicted_svm_final, zero_division = True))
print('Recall: %.2f'
      % recall_score(classes_test, predicted_svm_final, zero_division = True))
print('F-sore: %.2f'
      % f1_score(classes_test, predicted_svm_final, zero_division = True))
```

Código 24: Resumo das métricas do classificador SVM após o pré-processamento

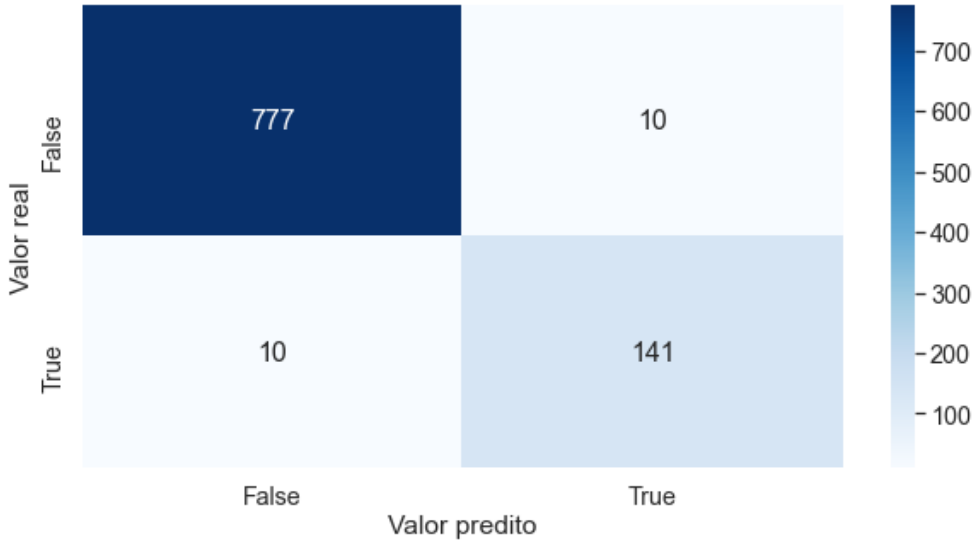

Figura 7: Matriz de confusão gerada a partir do classificador SVM após o pré-processamento

Métrica	Valor
Acurácia	98%
Precisão	93%
Sensibilidade (Recall)	93%
F-Score	93%

Tabela 16: Métricas alcançadas pela classificação SVM após o pré-processamento

3.4.2 Classificação com *Naive Bayes*

Os resultados alcançados foram:

```
gnb = GaussianNB()
model = gnb.fit(features_train, classes_train)
predicted_gnb_final = model.predict(features_test)
```

Código 25: Execução da classificação *Naive Bayes* após o pré-processamento

```
cm = confusion_matrix(classes_test, predicted_gnb_final)

cm_df = pd.DataFrame(cm, columns = np.unique(classes_test), index = np.unique(
    classes_test))
cm_df.index.name = 'Valor real'
cm_df.columns.name = 'Valor predito'
plt.figure(figsize = (10, 5))
sn.set(font_scale = 1.3)
sn.heatmap(cm_df, cmap="Blues", annot=True, annot_kws={"size": 16}, fmt='g')
```

Código 26: Geração da matriz de confusão da classificação *Naive Bayes* após o pré-processamento

```
print('Acuracia: %.2f'
      % accuracy_score(classes_test, predicted_gnb_final))
print('Precisao: %.2f'
      % precision_score(classes_test, predicted_gnb_final, zero_division = True))
```

```
print('Recall: %.2f'
      % recall_score(classes_test, predicted_gnb_final, zero_division = True))
print('F-sore: %.2f'
      % f1_score(classes_test, predicted_gnb_final, zero_division = True))
```

Código 27: Resumo das métricas do classificador *Naive Bayes* após o pré-processamento

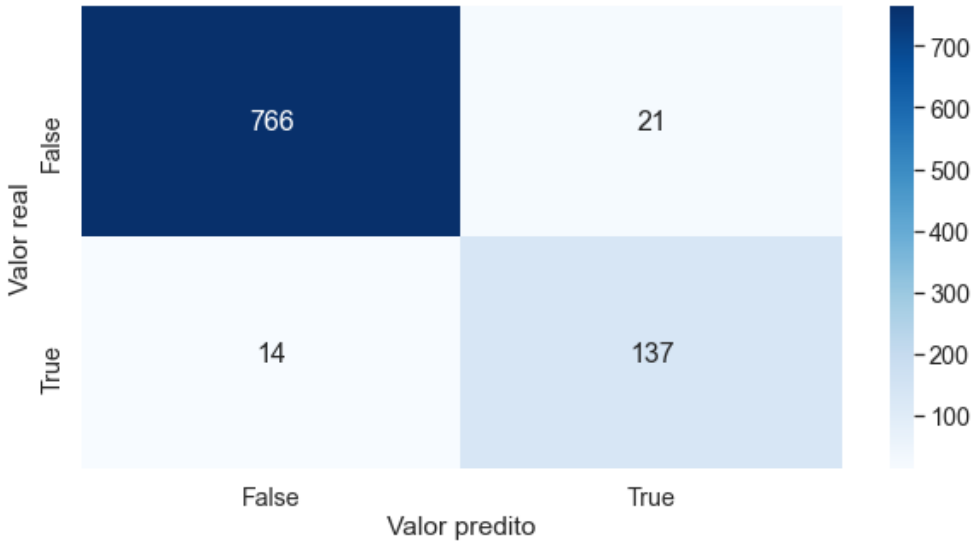

Figura 8: Matriz de confusão gerada a partir do classificador *Naive Bayes* após o pré-processamento

Métrica	Valor
Acurácia	96%
Precisão	87%
Sensibilidade (Recall)	91%
F-Score	89%

Tabela 17: Métricas alcançadas pela classificação *Naive Bayes* após o pré-processamento

4 Conclusões

As matrizes de confusão de ambos os algoritmos de classificação usados, SVM e *Naive Bayes*, mostram um melhor resultado após as etapas de pré-processamento exemplificadas nesse estudo de caso. Uma maior sensibilidade (*recall)* mostra que os dois algoritmos conseguiram prever asteroides perigosos com maior assertividade. É importante citar que para esse estudo os autores não contaram com a contribuição de especialistas na base de dados estudada, e escolheram atributos puramente a partir de técnicas de pré-processamento sem compreender a fundo o significado de cada *feature* no seu campo de estudo original. Embora não seja o indicado para situações do mundo real, a ausência de especialistas mostrou que as etapas de pré-processamento sozinhas são capazes de melhorar a predição e os resultados de algoritmos de classificação. No entanto, o estudo tenderia a apresentar resultados melhores, ou no mínimo, as análises teriam sido mais simples, se a ajuda de especialistas na base tivesse sido observada.

Referências

1 KAGGLE. Disponível em: <https://www.kaggle.com/kaggleuser654/is-the-asteroid-hazardous>.

2 CORTES, C.; VAPNIK, V. Support-vector networks, machine learning. *Springer Nature*, -, n. 20, p. 273–297, 1995.

3 RUSSELL, S.; NORVIG, P. Inteligência Artificial. *GEN LTC*, v. 3, 2003.

4 SCIKIT-LEARN. Disponível em: <https://scikit-learn.org/stable/modules/svm.html>.

5 ASSUNçãO, J. V. C. *Uma breve introdução à Mineração de Dados*. [S.l.]: Novatec Editora, 2021. ISBN 9786586057508.

6 FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, v. 17, p. 37, 1996.

7 HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The Elements of Statistical Learning*. [S.l.]: Springer, 2001. (New York: Springer series in statistics). ISBN 9786586057508.