UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ANA CAROLINA JUNGBLUTH
ISABELLA RODRIGUES GON

BIBLIOTECA SCIKIT-LEARN (SK-LEARN)

MEDIANEIRA - PR
2023

ANA CAROLINA JUNGBLUTH
ISABELLA RODRIGUES GON

BIBLIOTECA *SCIKIT-LEARN* (*SK-LEARN*)

Trabalho apresentado à disciplina de Sistemas Inteligentes Aplicados, do curso superior de bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná - UTFPR, como requisito parcial para avaliação semestral.

Professor: Jorge Aikes Junior, Dr.

MEDIANEIRA - PR

2023

1 Sobre o *dataSet* utilizado

O *ICDDoS2019* é um conjunto de dados destinado a fornecer uma representação precisa e atualizada de ataques de negação de serviço distribuídos (DDos) em redes. Seu objetivo principal é ajudar no desenvolvimento e na avaliação de métodos de detecção de ataques DDoS em tempo real com foco na minimização da carga computacional necessária.

Esse conjunto de dados abrange diversos tipos mais atualizados e comuns de ataques DDoS a partir da representação de dados reais em formato de pacotes de rede (PCAPs). Além disso, também traz a análise do tráfego de rede feita através do *CICFlowMeter-V3*, categorizando elementos como endereços IP de origem e destino, portas de origem e destino, horário de registro, protocolos utilizados e identificação de ataques.

2 Biblioteca *scikit-learn*

O *scikit-learn* é uma biblioteca *Python* desenvolvida sobre duas outras bibliotecas populares, *NumPy* e *SciPy*. Ela oferece ferramentas eficientes para análise de dados preditivos, abrangendo uma variedade de algoritmos para aprendizado supervisionado e não supervisionado. Desde regressão linear, *Support Vector Machine* (SVM) e árvores de decisão até métodos de agrupamento como *k-means* (SCIKIT-LEARN, 2023a).

Além disso, a biblioteca disponibiliza funcionalidade para fazer o pré-processamento de dados, o que permite a normalização, padronização e tratamento de valores nulos e *outliers*. Também oferece ferramentas para avaliação de modelos, com métricas como acurácia, precisão, e *recall*, que auxiliam na escolha do melhor modelo para uma tarefa. Outra característica interessante é o suporte da biblioteca à construção de *pipelines*, facilitando a criação de fluxos de trabalho organizados e reutilizáveis (SCIKIT-LEARN, 2023a).

3 Pré-processamento geral

Nesta seção será apresentado o pré-processamento feito para todos os algoritmos que utilizam o *dataSet Friday-WorkingHours-Afternoon-DDos.pcap_ISCX*. Para uma melhor visualização do código pode acessar o link do *Google Collab:* https://colab.research.google.com/drive/1HkDaPBnAOQcw14r5P9sY_PO7uyCoE3h4?usp=sharing.

3.1 Remoção dos valores nulos

Se os valores nulos não forem tratados adequadamente, isso pode introduzir vieses nos resultados ou levar a interpretações equivocadas durante a análise dos dados. Algoritmos de aprendizado de máquina muitas vezes requerem que todos os dados estejam presentes para aprender padrões precisos e fazer previsões confiáveis (ARAÚJO, 2022).

Por isso, foi feita a remoção das instâncias que continham qualquer valor nulo, utilizando a função *dropna().*

```python
dados = dados.dropna(how='any')
```

3.2 Transformação da classe do tipo *string* para binário

Como o objetivo era prever entre duas classes distintas, essa transformação também é para que o algoritmo de aprendizado de máquina possa calcular corretamente as métricas de desempenho, como precisão, *recall*, entre outros. Além de também converter os dados categóricos sem prejudicar a lógica de separação proporcionada pelo dado. (ROSAL, 2021).

Para isso foi criado um array *types_of_label* , esse array informa os valores que a classe possui. A partir disso, a função *replace_str()* funcionando como um laço de repetição, mapeia os valores e modifica-os. Nesse caso, como só existem dois valores, sendo eles, *BENIGN* e *DDos*, então *BENIGN* (0) e *DDos* (1).

```python
types_of_label = dados[' Label'].unique()
replace_str = {}
for i,value in enumerate(types_of_label, start=0):
  replace_str[value]=i
dados[' Label'] = dados[' Label'].map(replace_str)
```

3.3 Transformação dos dados do IP para númerico

Transformações aplicadas nos dados relacionados aos IPs que eram do tipo *string*, que foram transformados em numéricos. Essa transformação foi feita, principalmente para o uso do método *fit()*, que espera que os dados fornecidos sejam do tipo numérico. Esse método é usado após a criação de um modelo e é responsável por iniciar o processo de treinamento, onde o modelo aprende com os dados fornecidos (PACHECO, 2021).

3.3.1 *Source* IP, *Destination* IP e *Flow* ID

Para isso é criada uma nova coluna *Source IP Numerico*, que acessa a coluna *Source IP* e aplica a função *lambda* que converte endereços IP na forma de *string* para um formato numérico. E do mesmo modo foi feito com a coluna *Destination IP.* Após isso, foi realizado o *drop* das colunas *Source* IP e *Destination* IP. Adicionalmente, na coluna *Flow* ID também foi feito o *drop,* pois a mesma possui informações redundantes já apresentadas em outras colunas.

```python
dados['Source IP Numerico'] = dados[' Source IP'].apply(lambda
x: int(ipaddress.IPv4Address(x)) if isinstance(x, str) else x)
dados['Destination IP Numerico'] = dados[' Destination
IP'].apply(lambda x: int(ipaddress.IPv4Address(x)) if
isinstance(x, str) else x)

dados = dados.drop(' Source IP', axis=1)
```

```
dados = dados.drop(' Destination IP', axis=1)
dados = dados.drop('Flow ID', axis=1)
```

3.4 Selecionado atributos por correlação

Como haviam muitos atributos no *dataSet*, foi feito a análise da correlação dos atributos, a partir dela foi realizada uma seleção de atributos levando esse fator em consideração. Já que na correlação, basicamente é medido o quanto um atributo influencia o outro, ou seja, busca entender como dois atributos se movem juntos, se há uma relação linear entre eles, a direção (positiva ou negativa) e a força dessa relação. (SANTOS, 2021).

Para a análise da correlação entre os atributos foi utilizado o método *corr()* que calcula a matriz de correlação entre as colunas e armazena essa matriz em uma variável *correlation*, após isso é criada uma lista *low_correlations_coluns* que contém os rótulos das colunas, onde o valor da coluna *Label* está dentro do intervalo de -0.3 a 0.3. Após isso, é utilizado o *len(low_correlation_columns)* para calcular o número de colunas identificadas como tendo correlação baixa com a classe. E então essas colunas são excluídas gerando um novo conjunto de dados nomeado como *dados_sem_low_correlation*.

Python
```
correlation = dados.corr()
low_correlation_coluns = correlation.loc[(correlation[' Label']
< .3) & (correlation[' Label'] > -.3)].index
len(low_correlation_columns)

dados_sem_low_correlation =
dados.drop(columns=low_correlation_columns)
dados2 = dados_sem_low_correlation
```

3.5 Transformação do atributo *Timestamp*

Como mencionado anteriormente, o método *fit()* espera receber dados do tipo numérico. Então, foi aplicada uma transformação na coluna *Timestamp*, onde foi feita a separação de data e hora em duas colunas, sendo elas, *Date* e *Time*. Após isso foi transformada a coluna *Time* para o tipo numérico, e como as informações da coluna *Date* eram as mesmas para todas as instâncias a coluna foi deletada.

Para a transformação da coluna *Time* para numérico foi utilizada a função *time_to_seconds*, transformando a hora, minuto e segundos, em somente segundos, portanto a representação passa a ser uma representação do tempo, numérica em segundos.

```python
dados2['Date'] = pd.to_datetime(dados[' Timestamp']).dt.date
dados2['Time'] = pd.to_datetime(dados[' Timestamp']).dt.time
dados2 = dados2.drop(' Timestamp', axis=1)
dados2 = dados2.drop('Date', axis=1)
def time_to_seconds(time_obj):
  total_seconds = time_obj.hour * 3600 + time_obj.minute * 60 +
time_obj.second
  return total_seconds

dados2['Time_Numeric'] = dados2['Time'].apply(time_to_seconds)
dados2 = dados2.drop('Time', axis=1)
```

3.6 Remoção de *outliers*

Segundo Peres (2021) *outliers* são dados discrepantes em relação a um conjunto de dados. Este tipo de dado pode trazer distorções nas análises, e criar um viés falso ou pouco generalista da variação nas métricas e cálculos realizados pelos algoritmos de aprendizado de máquina.

Por esse fator foi feita a remoção dos *outliers* no *dataSet.* Para realizar essa remoção foi utilizado o algoritmo *IsolationForest* que é um algoritmo de detecção de *outliers* baseado em árvores de decisão que se destaca pela sua eficácia na

identificação de valores atípicos (*outliers*) em conjuntos de dados. (SANTANA, 2020; SCIKIT-LEARN, 2023b)

```python
model = IsolationForest()
model.fit(dados2)
outliers = model.predict(dados2)
dados_sem_outliers = dados2[outliers == 1]
```

4 Execução e apresentação dos resultados

Para execução dos resultados dos algoritmos, foi criada a função *execute_test*, que é um conjunto de processos, ou seja, o conjunto das funções *train_model* e *predict_with_model,* a função *train_model* treina o modelo usando os dados de treinamento (X_treinamento e y_treinamento) e retorna o modelo treinado (treinamento_model), a função *predict_with_model* recebe o modelo treinado (treinamento_model) e usa-o para fazer previsões nos dados de teste (x_teste). Ele retorna as previsões feitas pelo modelo (y_pred). Com isso, a função *execute_test* encapsula o processo completo. Ela recebe um modelo de machine learning, dados de treinamento (X_treinamento e y_treinamento - opcionalmente pode receber argumentos padrão se não forem especificados) e dados de teste (X_teste). Em seguida, utiliza as funções anteriores para treinar o modelo nos dados de treinamento e prever os resultados nos dados de teste. Finalmente, retorna as previsões feitas pelo modelo nos dados de teste (y_pred).

```python
def train_model(model, X_treinamento, y_treinamento):
  treinamento_model = model.fit(X_treinamento, y_treinamento)
  return treinamento_model
```

```python
def predict_with_model(treinamento_model, x_teste):
  y_pred = treinamento_model.predict(x_teste)
  return y_pred
```

```python
def execute_test(model, X_treinamento = x_treinamento,
y_treinamento= y_treinamento, X_teste= x_teste):
  treinamento_model = train_model(model, X_treinamento,
y_treinamento)
```

```
  y_pred = predict_with_model(treinamento_model, X_teste)
  return y_pred
```

Para a apresentação dos resultados dos algoritmos, foi criada a função *show_test*, essa função calcula e exibe várias métricas de avaliação do modelo de classificação, como a acurácia, calculando a precisão do modelo em prever corretamente as classes, também, o relatório de classificação em que apresenta métricas como precisão, recall e F1-Score para cada classe presente nos dados de teste, fornecendo uma visão mais detalhada do desempenho do modelo em cada classe. E também, a matriz de confusão, que mostra uma representação visual de uma tabela que compara as classes reais com as classes previstas pelo modelo. Isso permite a visualização dos acertos e erros do modelo para cada classe.

Python
```
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix

def show_test(y_pred, y_teste=y_teste):
  accuracy = accuracy_score(y_teste, y_pred)
  print("Acurácia:", accuracy*100,"%")

  report = classification_report(y_teste, y_pred,
output_dict=True)
  print("\n\t     Relatório de Classificação")

print("-----------------------------------------------------")
                                                    print(f"
{'Classe':<10}{'Precisão':<10}{'Recall':<10}{'F1-Score':<10}{'S
uporte':<10}")
```

```
print("-------------------------------------------------------")
  for cls, metrics in report.items():
    if cls.isdigit():
                                                        print(f"
{cls:<10}{metrics['precision']:<10.2f}{metrics['recall']:<10.2f
}{metrics['f1-score']:<10.2f}{metrics['support']:<10}")

print("-------------------------------------------------------")

  plt.figure(figsize=(8, 6))
     sns.heatmap(confusion_matrix(y_teste, y_pred), annot=True,
fmt='d', cmap='Purples')
  plt.xlabel('Predict')
  plt.ylabel('True')
  plt.title('Matriz de Confusão')
  plt.show()
```

5 Algoritmos

Nesta seção será apresentada uma breve descrição dos algoritmos utilizados no presente trabalho e para alguns casos um pré-processamento específico. Além dos resultados obtidos para cada algoritmo.

5.1 Árvore de Decisão *Classification And Regression Trees* (CART)

A árvore de decisão CART é um algoritmo de aprendizado de máquina usado para classificação e regressão. Destaca-se por criar modelos simples e interpretáveis, representados como árvores. As decisões são baseadas em divisões binárias nos dados, usando critérios para minimizar a impureza nos nós. A árvore é construída inicialmente e, em seguida, podada para evitar o *overfitting* (CLARKE; BITTENCOURT, 2003).

O código utilizado para fazer a classificação na árvore de decisão CART é apresentado abaixo.

```python
from sklearn import tree

tree = tree.DecisionTreeClassifier()
y_pred = execute_test(tree, x_treinamento, y_treinamento,
x_teste)
show_test(y_pred)
```

5.1.1 Resultados CART (padrão)

Para a árvore de decisão CART os resultados foram satisfatórios, com uma acurácia de 99,9945% e um *recall* de 1,00 para a classe de interesse *DDos*, sugerindo que o modelo foi capaz de aprender com grande precisão os padrões e relações nos dados fornecidos para fazer boas previsões.

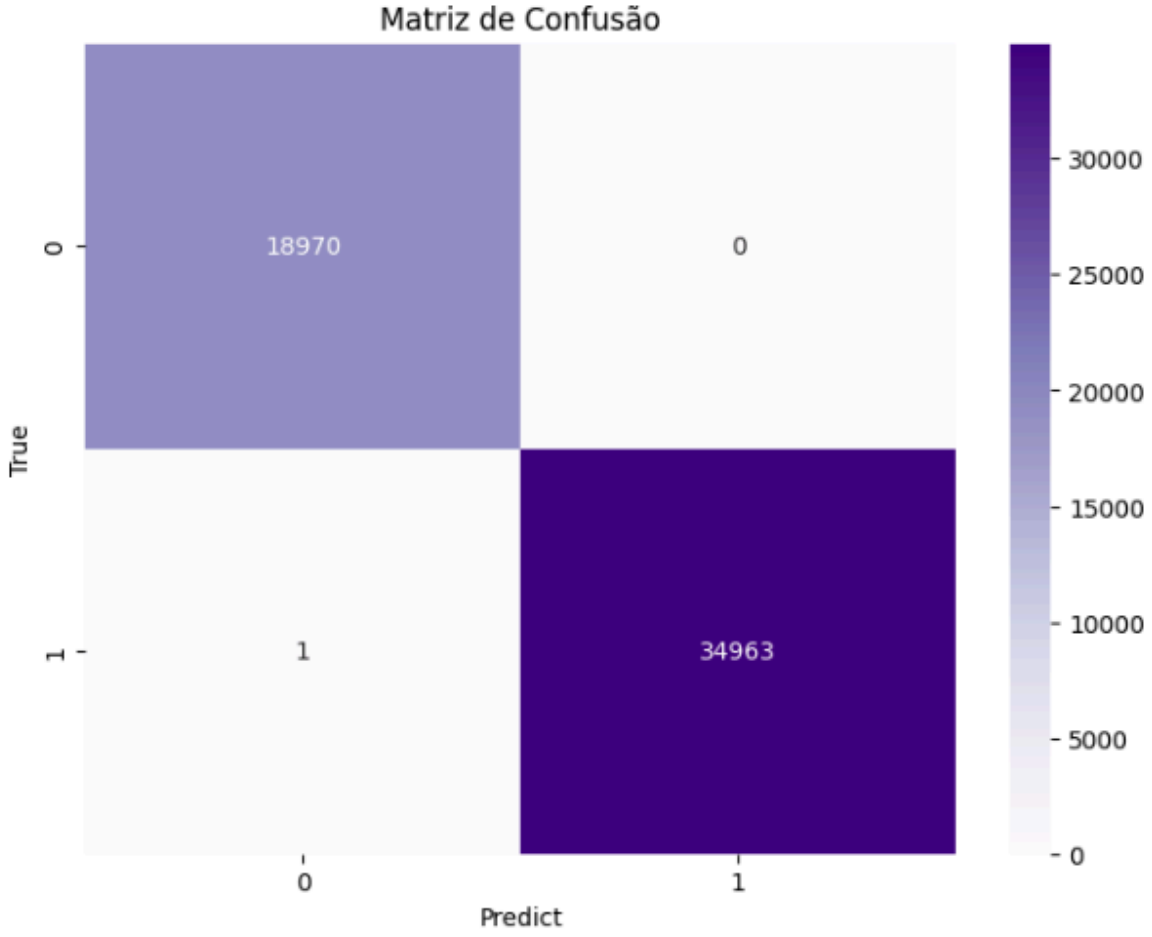
Figura 1 - Relatório de Classificação árvore de decisão CART

Relatório de Classificação

Classe	Precisão	Recall	F1-Score	Suporte
0	1.00	1.00	1.00	18970
1	1.00	1.00	1.00	34964

Fonte: Autoria própria (2023).

Figura 2 - Matriz de confusão árvore de decisão CART



Fonte: Autoria própria (2023).

5.2 *Support Vector Machine* (SVM)

Support Vector Machine (SVM) é um algoritmo de aprendizado de máquina utilizado para classificação e regressão. Seu objetivo principal é encontrar um hiperplano de decisão que maximize a margem entre classes, sendo os vetores de

suporte os pontos mais próximos à fronteira. O SVM pode lidar com dados linearmente separáveis e não separáveis, graças ao uso de *kernels* que transformam os dados em espaços de características de alta dimensão. Possui parâmetros, como o de regularização, e é eficaz em espaços de alta dimensão, resistente a *overfitting* (NOBLE, 2006).

O código utilizado para fazer a classificação no SVM é apresentado abaixo.

```python
from sklearn.svm import SVC
svm_model = SVC()
y_pred_svm = execute_test(svm_model,
X_treinamento=x_treinamento, X_teste=x_teste)
show_test(y_pred_svm)
```

5.2.1 Pré-Processamento específico

O SVM muitas vezes se beneficia da padronização dos dados devido à sua sensibilidade à escala das características. Isso porque este modelo busca encontrar um hiperplano de separação ideal entre as classes, e a escala das características pode afetar diretamente esse processo. Quando os dados não estão padronizados, características com magnitudes maiores podem ter um peso desproporcional na determinação do hiperplano de decisão, enquanto características com escalas menores podem ser subestimadas (IPNET, 2021).

Por isso foi feita a padronização dos atributos do dataSet utilizando a classe *StandardScaler*. Essa classe opera em cada coluna dos dados separadamente e transforma os dados de maneira que a média de cada coluna seja 0 e o desvio padrão seja 1.

```python
scaler = StandardScaler()
x_treinamento = scaler.fit_transform(x_treinamento)
x_teste = scaler.transform(x_teste)
```

5.2.2 Resultados SVM (padrão)

Os resultados obtidos com as configurações padrão do SVM foram notavelmente positivos. Com uma acurácia de 99,9981% e um *recall* de 1,00 para a classe DDos, o modelo demonstrou capacidade de compreender os padrões e relações nos dados, permitindo previsões precisas para a classe de interesse.

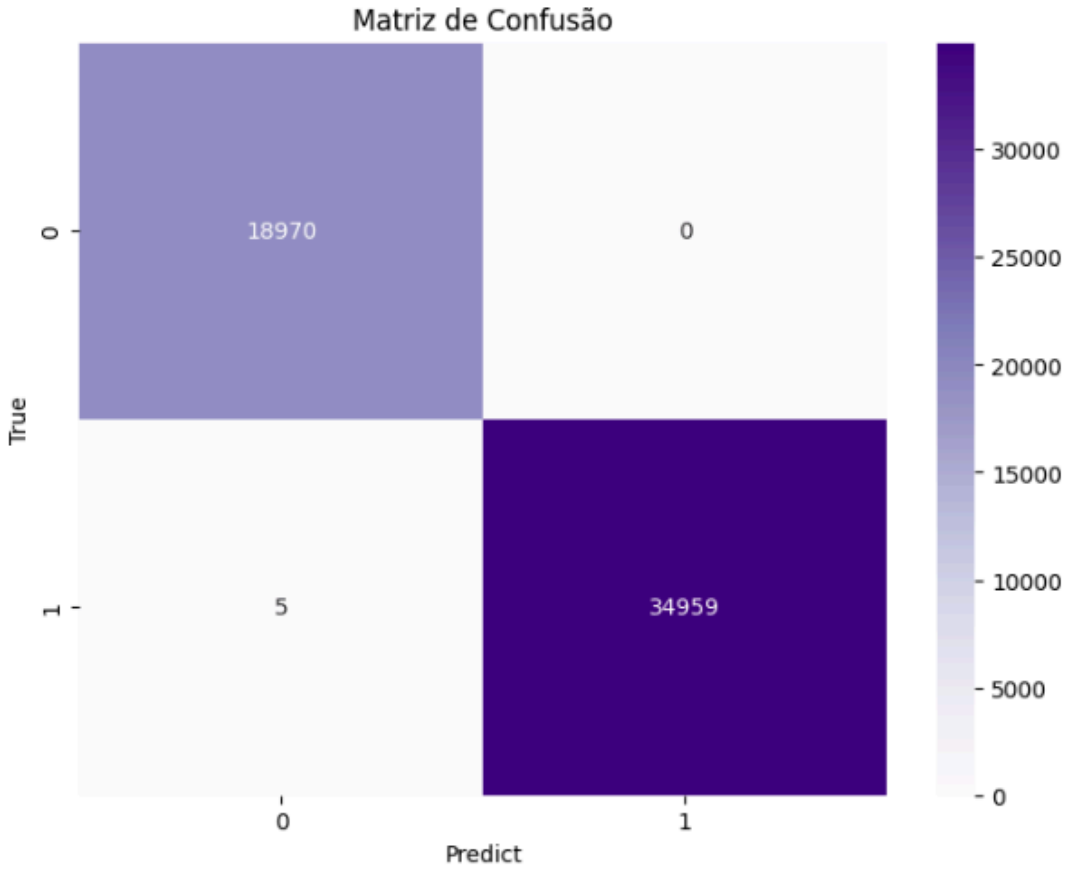
Figura 3 - Relatório de Classificação SVM

Relatório de Classificação

Classe	Precisão	Recall	F1-Score	Suporte
0	1.00	1.00	1.00	18970
1	1.00	1.00	1.00	34964

Fonte: Autoria própria (2023).

Figura 4 - Matriz de confusão SVM



Fonte: Autoria própria (2023).

5.2.3 Resultados SVM (mudando o *kernel* por outro)

O parâmetro *kernel* no SVM especifica o tipo de função *kernel* a ser usado no algoritmo. A função *kernel* é responsável por mapear os dados de entrada para um espaço de maior dimensionalidade, onde é mais fácil encontrar um hiperplano de separação entre as classes. No *scikit-learn*, o *kernel* padrão para SVM é o *kernel Radial Basis Function* (RBF). Para testar outra função foi utilizada a polinomial, que calcula o produto escalar dos vetores de características transformadas para encontrar um hiperplano de separação (SCIKIT-LEARN, 2023c).

Figura 5 - Relatório de Classificação SVM (Kernel = polinomial)

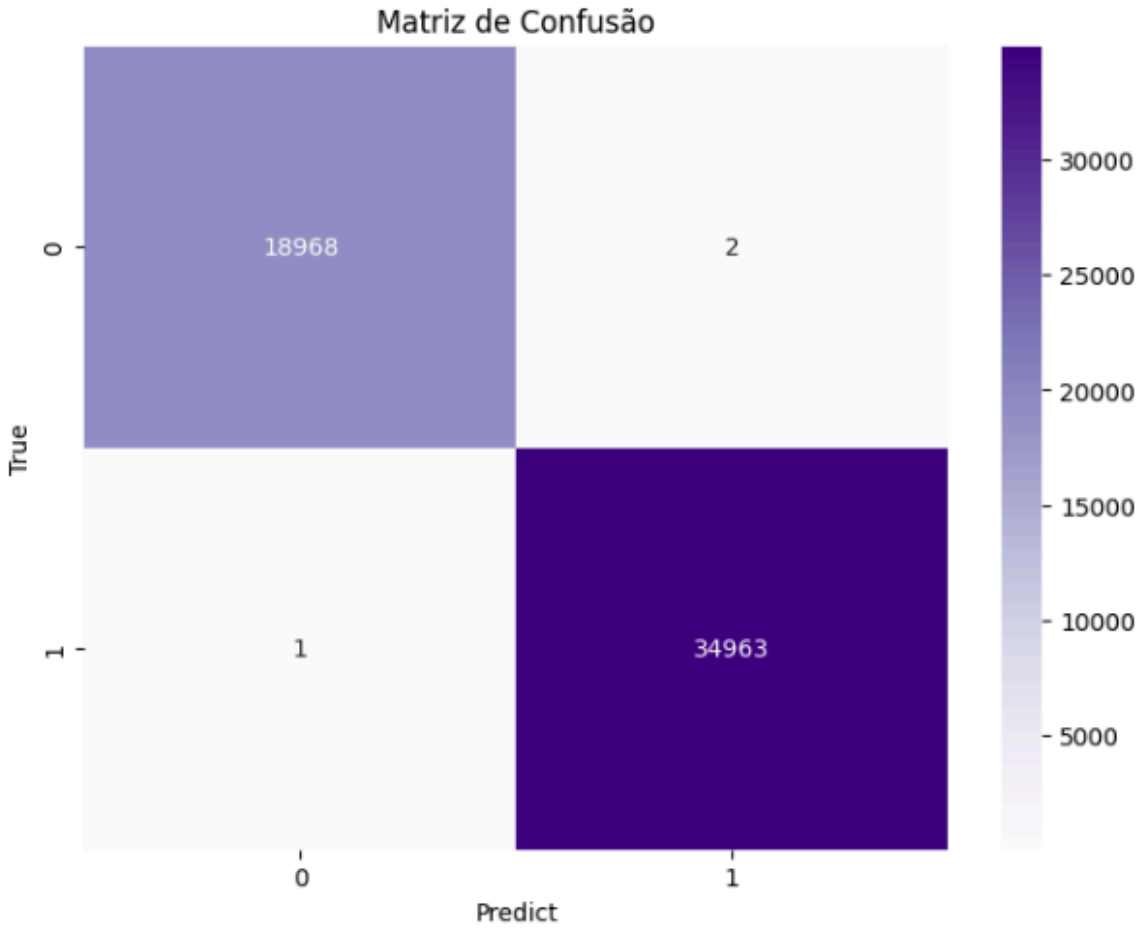
Relatório de Classificação

Classe	Precisão	Recall	F1-Score	Suporte
0	1.00	1.00	1.00	18970
1	1.00	1.00	1.00	34964

Fonte: Autoria própria (2023).

Para a SVM configurado com a função *kernel* polinomial, os resultados foram satisfatórios, com uma acurácia de 99,9945% um pouco menor que a do SVM com a configuração padrão, e um *recall* de 1,00 para a classe de interesse *DDos*, sugerindo que o modelo foi capaz de aprender com grande precisão os padrões e relações nos dados fornecidos para fazer boas previsões.

Figura 6 - Matriz de confusão SVM (Kernel = polinomial)



Fonte: Autoria própria (2023).

5.3 *k-Nearest Neighbors* (kNN)

O algoritmo kNN de aprendizado de máquina, classifica um dado com base em quão similar é a outros dados no espaço de características. O funcionamento é simples: calcular a distância entre o dado a ser classificado e todos os dados já classificados, selecionar os k-vizinhos mais próximos, verificar a classe predominante entre eles e atribuir essa classe ao dado a ser classificado. O cálculo da distância pode ser feito usando fórmulas como a Euclidiana (JOSÉ, 2018).

O código utilizado para fazer a classificação no algoritmo kNN é apresentado abaixo.

```python
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier()
y_pred_knn = execute_test(knn_model,
X_treinamento=x_treinamento, X_teste=x_teste)
show_test(y_pred_knn)
```

5.3.1 Pré-processamento específico

Do mesmo modo que que o SVM, para o kNN também foi feita a padronização de dados, uma vez que o kNN determina a similaridade entre pontos de dados com base em distâncias, buscando os vizinhos mais próximos para fazer previsões ou classificações. Quando os dados não estão padronizados, características com escalas maiores podem ter uma influência desproporcional na medida de distância, distorcendo a identificação dos vizinhos mais próximos. Isso pode resultar em um viés na seleção dos vizinhos, prejudicando a precisão das previsões (IPNET, 2021).

Por esse fator foi feita a padronização dos atributos, da mesma forma que no SVM subseção 5.2.1.

5.3.2 Resultados KNN (padrão)

Figura 7 - Relatório de Classificação kNN

Relatório de Classificação

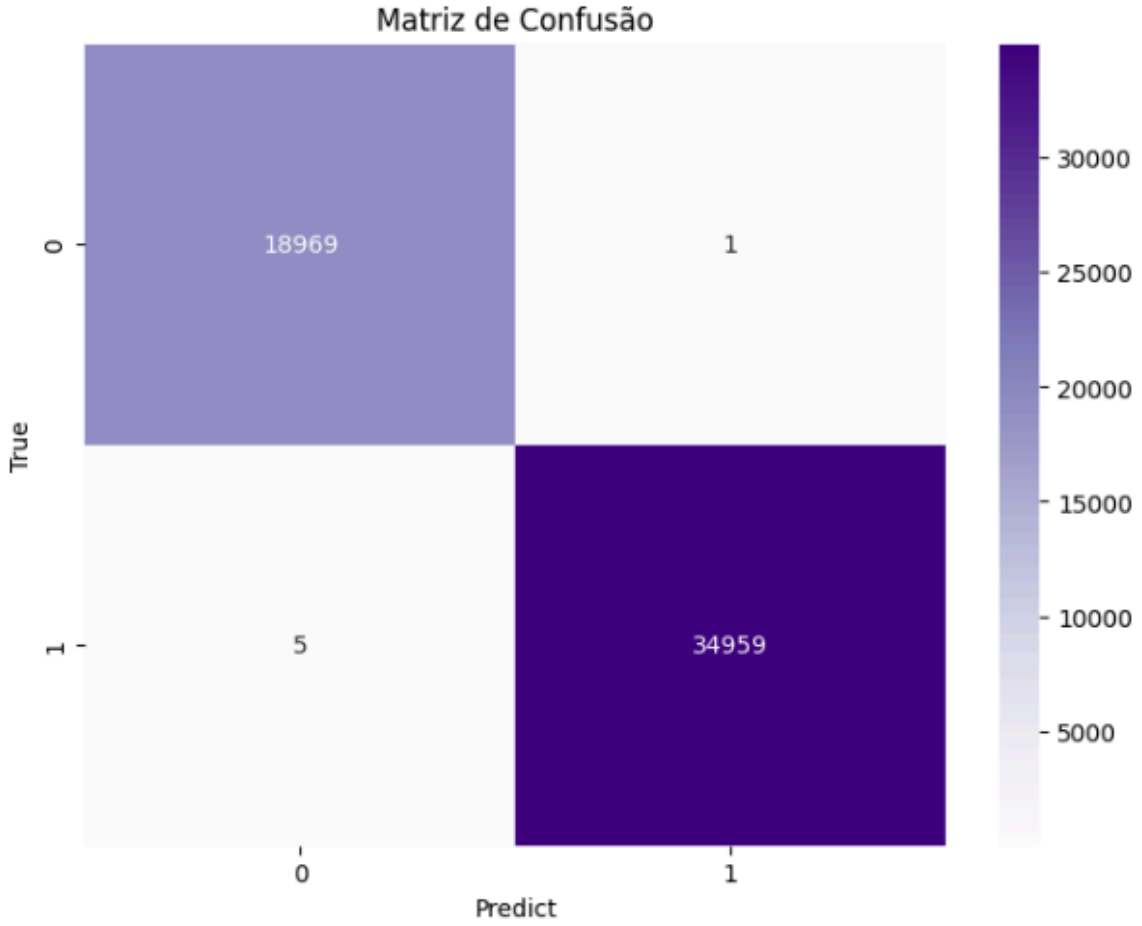
Classe	Precisão	Recall	F1-Score	Suporte
0	1.00	1.00	1.00	18970
1	1.00	1.00	1.00	34964

Fonte: Autoria própria (2023).

Para o kNN com as configurações padrões os resultados foram satisfatórios, com uma acurácia de 99,9963% e um *recall* de 1,00 para a classe de interesse

DDos, indicando que o modelo aprendeu os padrões e relações nos dados com precisão. O que permite que o modelo faça previsões precisas.

Figura 8 - Matriz de confusão kNN



Fonte: Autoria própria (2023).

5.3.3 Resultados kNN (mudando o valor de k)

Segundo Vaz (2021) o parâmetro k representa o número de vizinhos mais próximos que são considerados para tomar uma decisão de classificação ou regressão para um determinado ponto de dados. Ao aumentar o valor de k, o modelo considera mais vizinhos próximos para tomar uma decisão. Por outro lado, ao diminuir o valor de k, o modelo baseia-se em menos vizinhos próximos. No *scikit-learn* o k padrão é 5. Para o primeiro teste foi feita a escolha de um k menor 1.

Figura 9 - Relatório de Classificação kNN (k = 1)

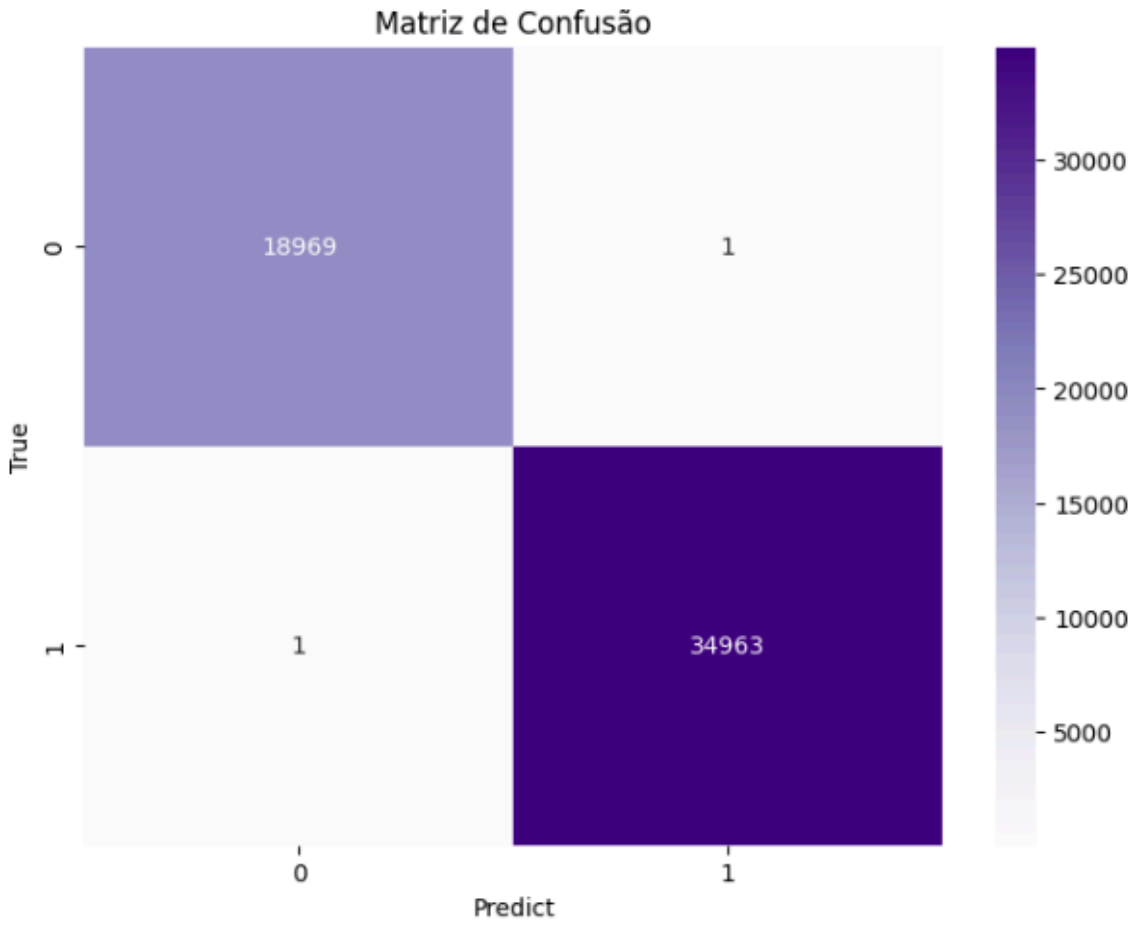
Relatório de Classificação

Classe	Precisão	Recall	F1-Score	Suporte
0	1.00	1.00	1.00	18970
1	1.00	1.00	1.00	34964

Fonte: Autoria própria (2023).

Para o kNN com k igual a 1 os resultados foram satisfatórios, com uma acurácia de 99,9927% um pouco menor do que a do kNN com as configurações padrão, e um *recall* de 1,00 para a classe de interesse *DDos*, sugerindo que o modelo foi capaz de aprender com grande precisão os padrões e relações nos dados fornecidos para fazer boas previsões.

Figura 10 - Matriz de confusão kNN (k = 1)



Fonte: Autoria própria (2023).

Já para um segundo teste foi feita a escolha de um k maior 14.

Figura 11 - Relatório de Classificação kNN (k = 14)

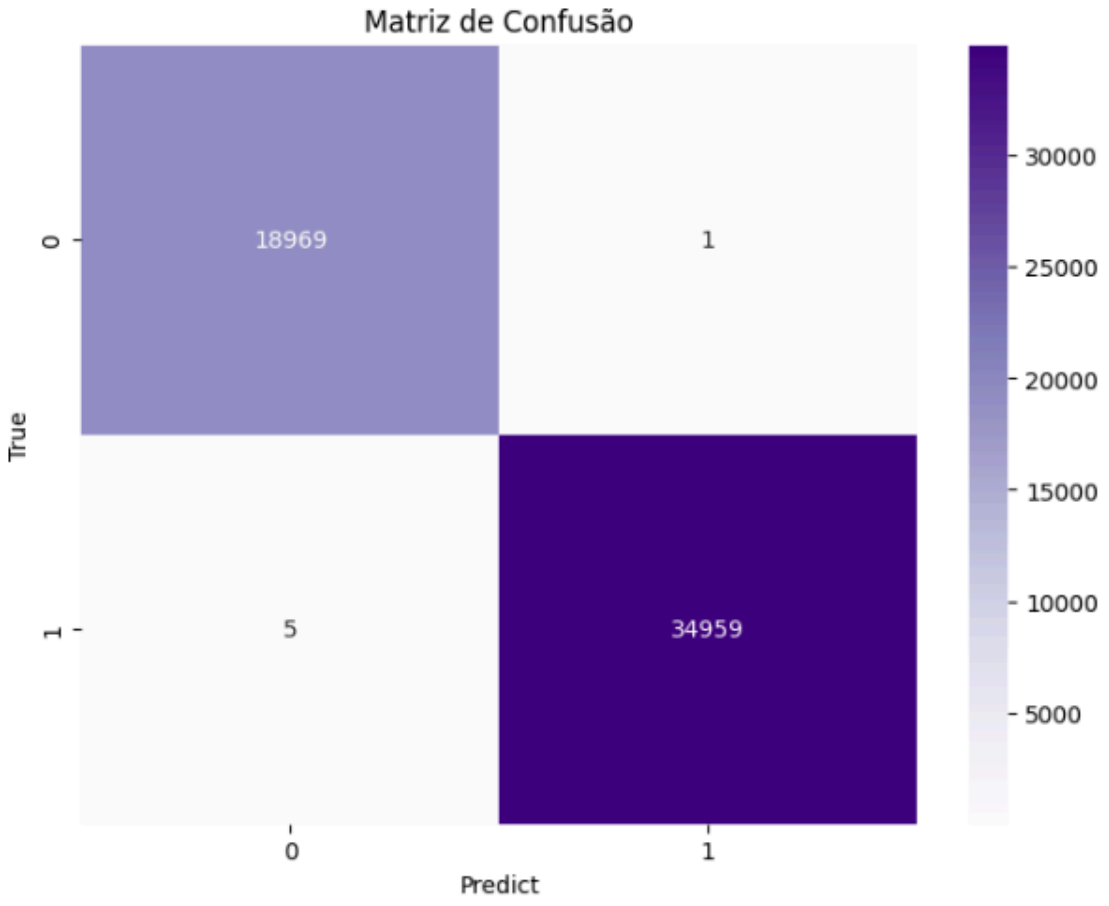
Relatório de Classificação

Classe	Precisão	Recall	F1-Score	Suporte
0	1.00	1.00	1.00	18970
1	1.00	1.00	1.00	34964

Fonte: Autoria própria (2023).

Para o kNN com k igual a 14 os resultados também foram satisfatórios, com uma acurácia de 99,9888%, e um *recall* de 1,00 para a classe de interesse *DDos*, indicando que o modelo aprendeu os padrões e relações nos dados com precisão. O que permite que o modelo faça previsões precisas.

Figura 12 - Matriz de confusão kNN (k = 14)



Fonte: Autoria própria (2023).

5.3.4 Resultados kNN (mudando a medida de distância)

A alteração do parâmetro de distância no algoritmo k-NN determina a métrica usada para calcular a distância entre os pontos de dados no espaço dimensional. A escolha da métrica de distância afeta diretamente a maneira como os vizinhos mais próximos são identificados e, consequentemente, influencia as decisões de classificação ou regressão do modelo (VAZ, 2021). A distância padrão no *scikit-learn* é a Euclidiana, para o primeiro teste foi escolhida a distância de *Manhattan* que mede a distância como a soma das diferenças absolutas entre as coordenadas de cada dimensão (VAZ, 2021).

Figura 13 - Relatório de Classificação kNN (distância = *Manhattan*)

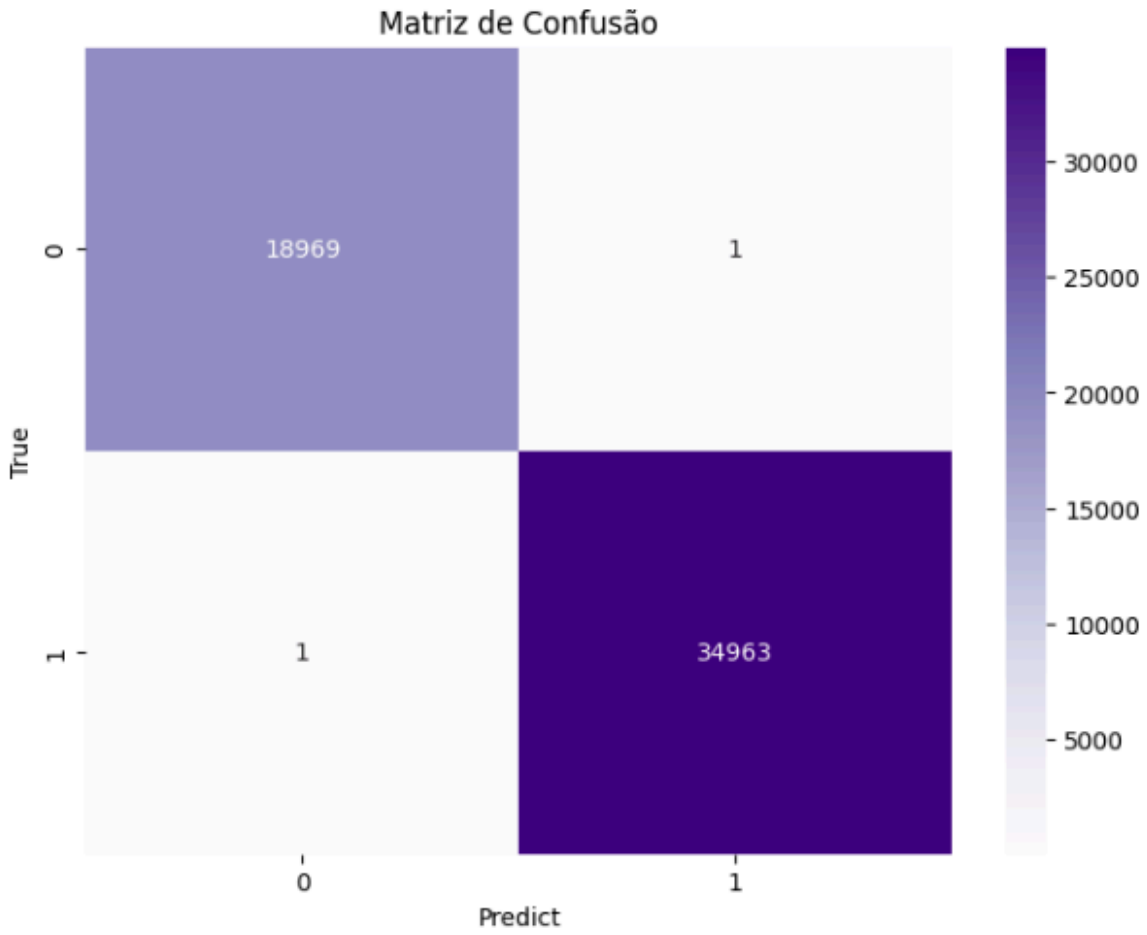
Relatório de Classificação

Classe	Precisão	Recall	F1-Score	Suporte
0	1.00	1.00	1.00	18970
1	1.00	1.00	1.00	34964

Fonte: Autoria própria (2023).

Para a distância *Manhattan* os resultados também foram satisfatórios, com uma acurácia de 99,9963% idêntica a do kNN com as configurações padrão, e um *recall* de 1,00 para a classe de interesse *DDos*, indicando que o modelo aprendeu os padrões e relações nos dados com precisão. O que capacita o modelo a realizar previsões acuradas.

Figura 14 - Matriz de confusão kNN (distância = *Manhattan*)



Fonte: Autoria própria (2023).

Para um segundo teste foi escolhida a distância de *Chebyshev* que calcula a distância máxima entre as coordenadas de cada dimensão (VAZ, 2021).

Figura 15 - Relatório de Classificação kNN (distância = *Chebyshev*)

Relatório de Classificação

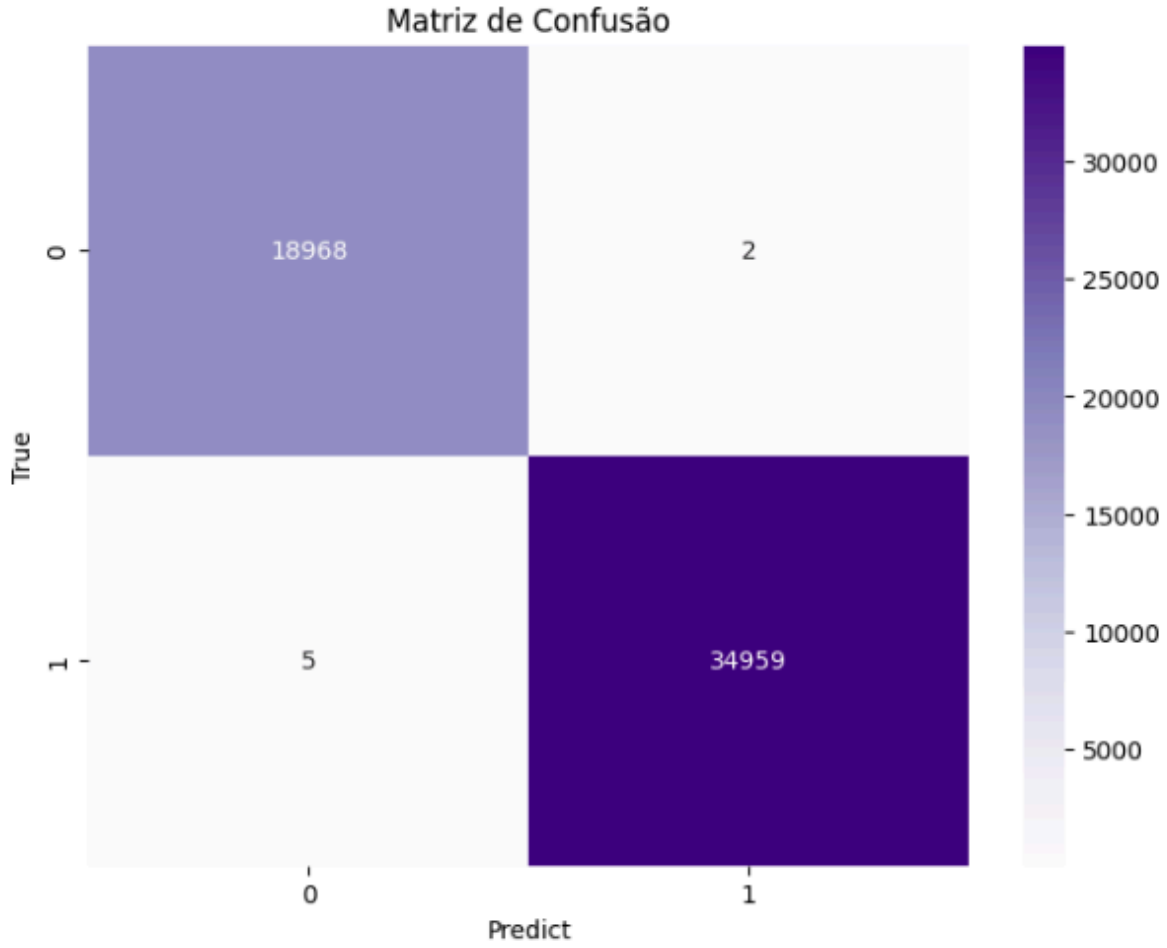
Classe	Precisão	Recall	F1-Score	Suporte
0	1.00	1.00	1.00	18970
1	1.00	1.00	1.00	34964

Fonte: Autoria própria (2023).

Para a distância *Chebyshev* os resultados também foram satisfatórios, com uma acurácia de 99,9981% um pouco maior do que os outros testes do kNN, e um

recall de 1,00 para a classe de interesse *DDos*, sugerindo que o modelo foi capaz de aprender com grande precisão os padrões e relações nos dados fornecidos, permitindo que ele faça previsões precisas.

Figura 16 - Matriz de confusão kNN (distância = *Chebyshev*)



Fonte: Autoria própria (2023).

5.4 *Gaussian Naive Bayes* (GNB)

O GNB é uma variante do algoritmo *Naive Bayes* que assume que os atributos contínuos associados a cada classe seguem uma distribuição normal. Durante o treinamento, o modelo calcula as médias e os desvios padrão de cada atributo para cada classe. Para classificar uma nova instância, o GNB utiliza o Teorema de *Bayes* para calcular a probabilidade condicional de pertencer a cada classe, dadas as observações dos atributos. A classe atribuída é aquela com a maior probabilidade condicional calculada (MAJUMDER, 2020).

O código utilizado para fazer a classificação no GNB é apresentado abaixo.

```python
from sklearn.naive_bayes import GaussianNB

naive_bayes_model = GaussianNB()
y_pred_gnb = execute_test(naive_bayes_model,
X_treinamento=x_treinamento, X_teste=x_teste)
show_test(y_pred_gnb)
```

5.4.1 Pré-processamento específico

O GNB assume que os dados de cada classe seguem uma distribuição gaussiana (também conhecida como distribuição normal). A padronização facilita para o modelo fazer suposições mais precisas sobre a distribuição dos dados para cada classe. Portanto, para esse algoritmo os dados também foram padronizados, como na subseção 5.2.1 (IPNET, 2021).

5.4.2 Resultados GNB (padrão)

Figura 17 - Relatório de Classificação GNB

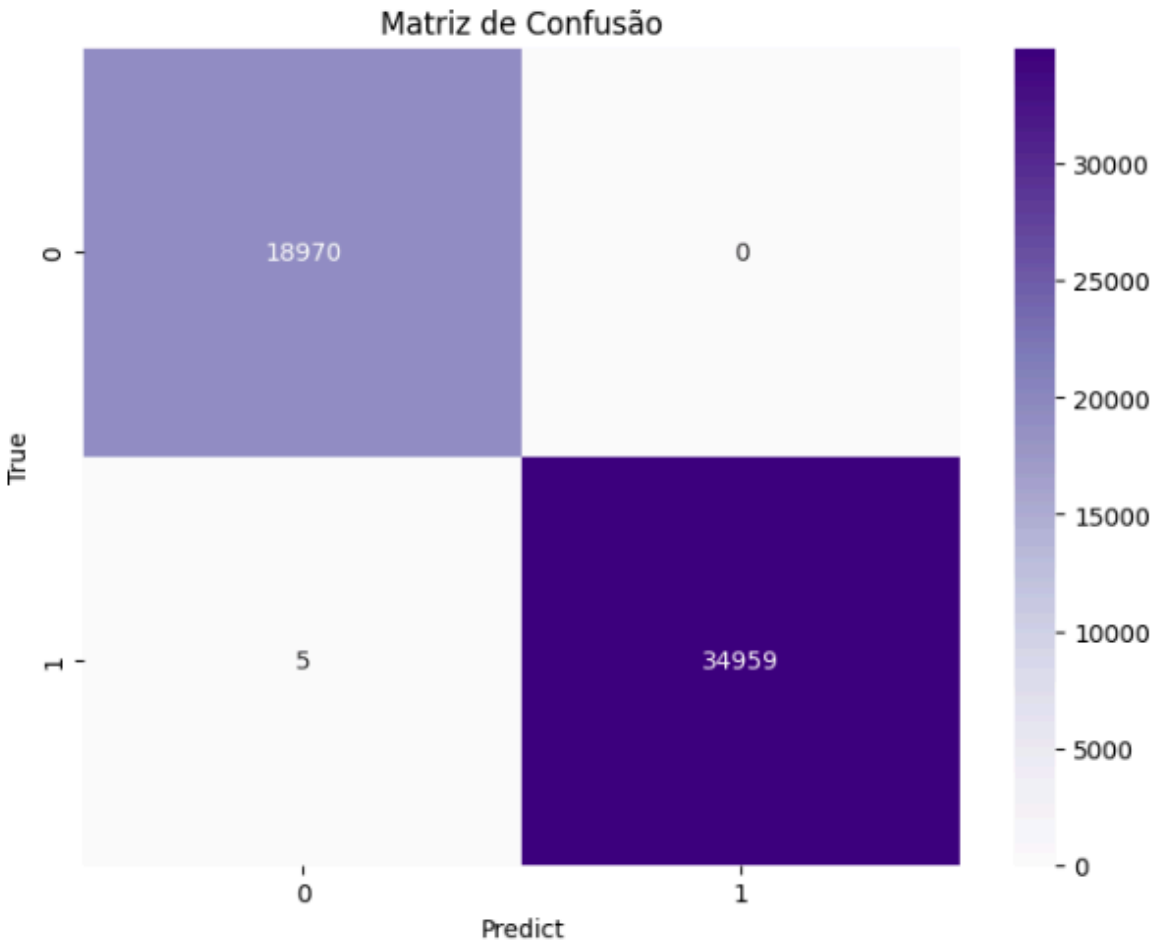
Relatório de Classificação

Classe	Precisão	Recall	F1-Score	Suporte
0	1.00	1.00	1.00	18970
1	1.00	1.00	1.00	34964

Fonte: Autoria própria (2023).

Para o GNB os resultados também foram satisfatórios, com uma acurácia de 99,9907% e um *recall* de 1,00 para a classe de interesse *DDos*, indicando que o modelo aprendeu os padrões e relações nos dados com precisão. O que capacita o modelo a realizar previsões acuradas.

Figura 18 - Matriz de confusão GNB



Fonte: Autoria própria (2023).

5.5 Categorical Naive Bayes (CategoricalNB)

Assim como o GNB, o *Categorical Naive Bayes* é uma variação do algoritmo *Naive Bayes,* mais é projetado para lidar com atributos categóricos, ou seja, dados que representam categorias discretas. Durante o treinamento, calcula probabilidades condicionais para cada valor categórico em relação a cada classe. Utiliza a distribuição multinomial para modelar essas probabilidades, mantendo a suposição de independência condicional entre atributos. Na classificação, determina a classe que maximiza a probabilidade a posteriori, dada a observação específica dos atributos (MAJUMDER, 2020; MATHWORKS, 2023).

O código utilizado para fazer a classificação no algoritmo *Categorical Naive Bayes* é apresentado abaixo.

```python
from sklearn.naive_bayes import CategoricalNB

categorical_nb_model = CategoricalNB()
y_pred_categorical_nb = execute_test(categorical_nb_model,
X_treinamento=x_treinamento_discretized,
X_teste=x_teste_discretized)
show_test(y_pred_categorical_nb, y_teste)
```

5.5.1 Pré-processamento específico

Para aplicação do algoritmo *Categorical Naive Bayes*, além do pré-processamento geral, foi feita a discretização, ela é necessária porque o *CategoricalNB* assume que os atributos são categóricos ou discretos. Quando alimentado com dados contínuos diretamente, o modelo não é capaz de interpretar ou aprender com esses valores, o que pode levar a resultados imprecisos. Resumidamente, quando se trata de dados contínuos ou numéricos, como valores que podem variar ao longo de um intervalo contínuo, é importante discretizar-los para que o *CategoricalNB* possa lidar com esses dados de maneira apropriada (TIMÓTEO, 2023; SCIKIT-LEARN, 2023d) .

Para a discretização foi utilizada a classe *KBinsDiscretizer* do scikit-learn, ela realiza a discretização das características numéricas em intervalos fixos de valores, nessa classe é possível definir o número de intervalos nos quais se deseja discretizar os dados (*n_bins*). Como a aplicação do algoritmo é feita em um *dataSet* que possui uma variação ampla, é preciso de mais bins para que essas variações sejam capturadas de uma forma melhor, portanto, o número de bins configurado para a discretização foi de 10, que apresentou resultados melhores em relação a números mais baixos.

```python
discretizer = KBinsDiscretizer(n_bins=10, encode='ordinal',
strategy='uniform')
```

```
x_treinamento_discretized =
discretizer.fit_transform(x_treinaux)
x_teste_discretized = discretizer.transform(x_testaux)
```

5.5.2. Resultados *CategoricalNB* (padrão)

Figura 19 - Relatório de Classificação *CategoricalNB* (*n_bins* = 10)

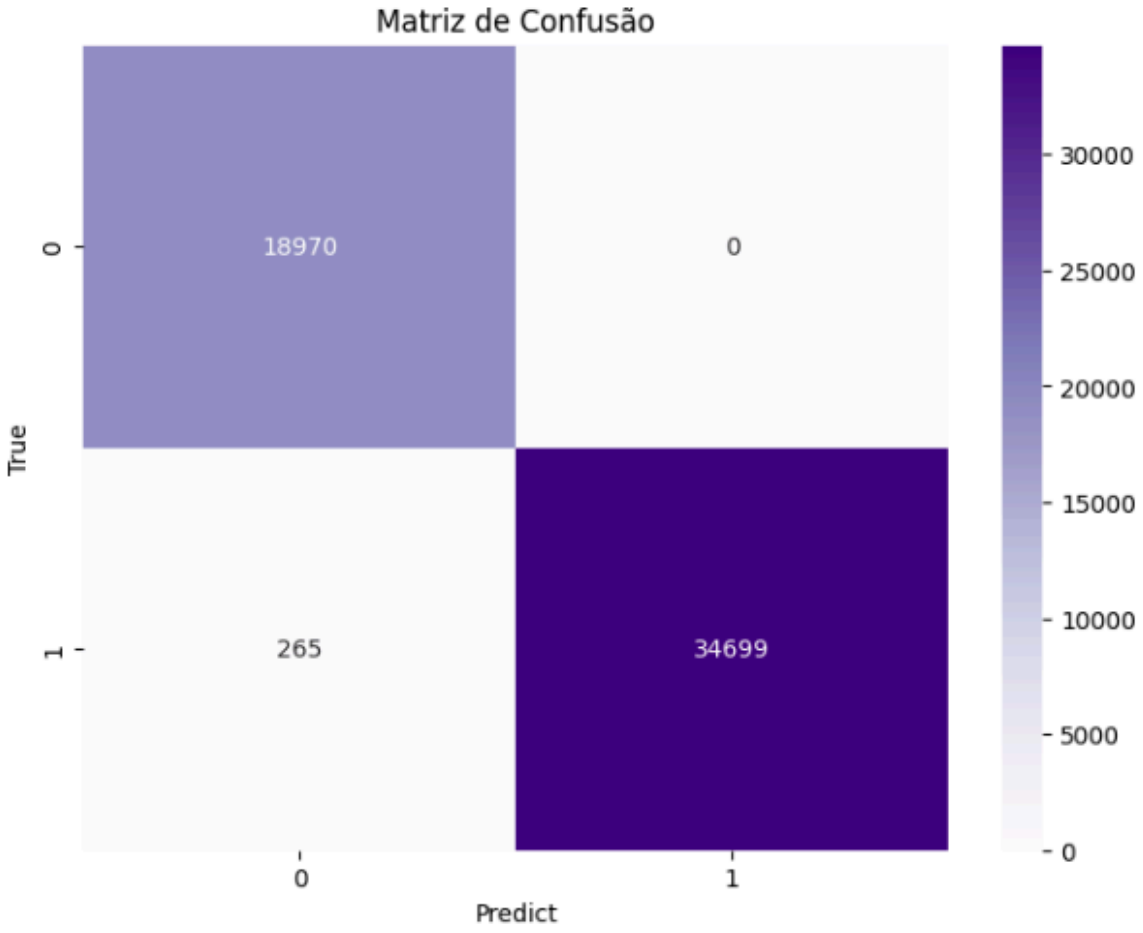
Relatório de Classificação

Classe	Precisão	Recall	F1-Score	Suporte
0	0.99	1.00	0.99	18970
1	1.00	0.99	1.00	34964

Fonte: Autoria própria (2023).

Para o *CategoricalNB* os resultados também foram satisfatórios, para o *n_bins* configurado em 10, obteve-se uma acurácia de 99,5086% e um *recall* de 0,99 que foi um pouco menor comparado aos outros teste realizados com o mesmo *dataSet,* para a classe de interesse *DDos*, mas que ainda foi o suficiente para que o modelo pudesse aprender os padrões e relações nos dados com precisão. O que capacita o modelo a realizar previsões boas.

Figura 20 - Matriz de confusão *CategoricalNB* (*n_bins* = 10)



Fonte: Autoria própria (2023).

Para justificar a escolha do número 10 para o parâmetro *n_bins*, também foi testado um valor mais baixo para o mesmo, em que o *n_bins* na discretização, foi configurado em 5.

Figura 21 - Relatório de Classificação *CategoricalNB* (*n_bins* = 5)

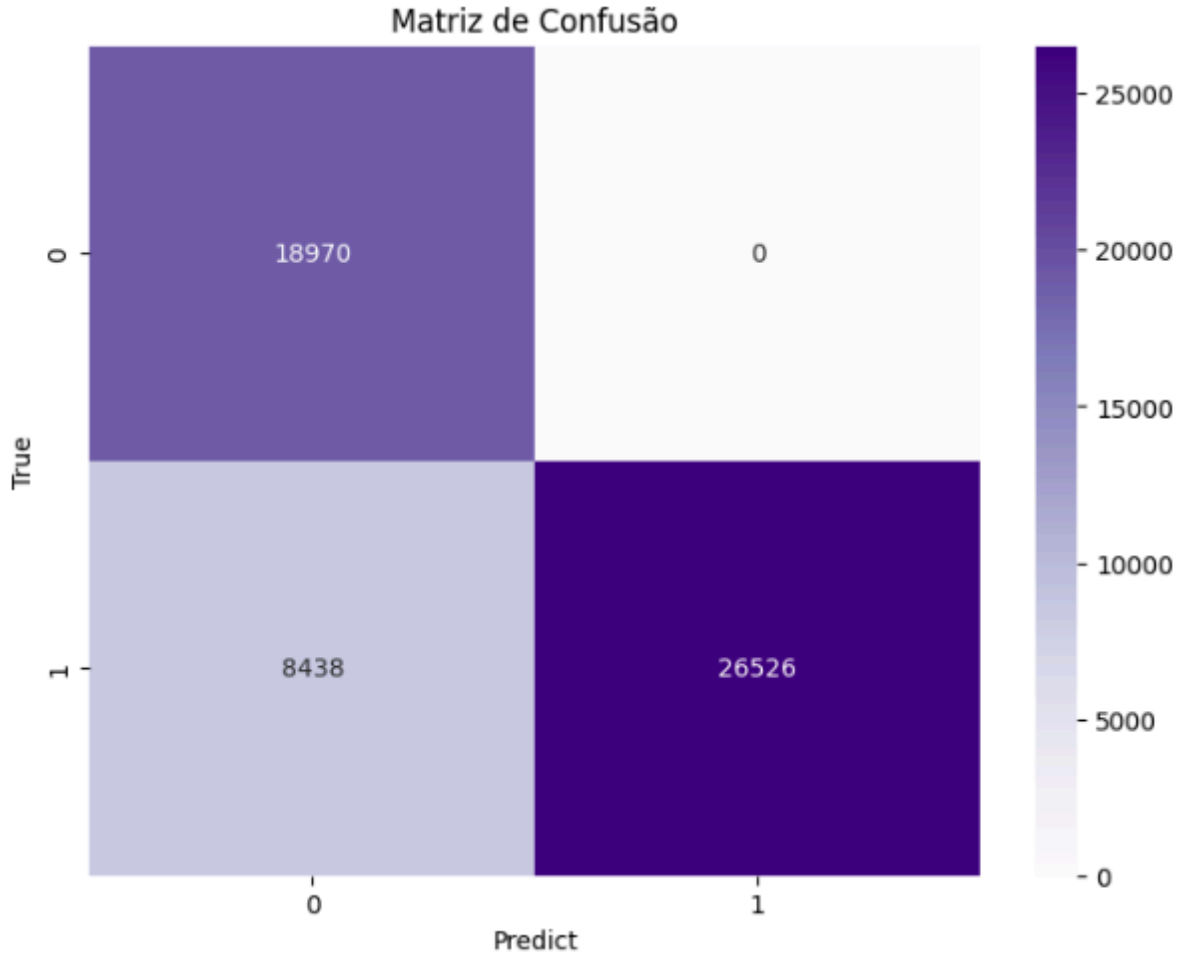
Relatório de Classificação

Classe	Precisão	Recall	F1-Score	Suporte
0	0.69	1.00	0.82	18970
1	1.00	0.76	0.86	34964

Fonte: Autoria própria (2023).

Para *n_bins* igual a 5 o algoritmo obteve uma acurácia de 84,3549%, e um *recall* de 0,76 para a classe de interesse, o que significa que o desempenho desse algoritmo em específico, considerando o conjunto de dados utilizado, é melhor quando se tem um maior número de intervalos na discretização, devido a ampla variação dos dados.

Figura 22 - Matriz de confusão *CategoricalNB* (*n_bins* = 5)



Fonte: Autoria própria (2023).

5.6 Regressão Linear

A Regressão Linear por Mínimos Quadrados é uma técnica estatística para modelar relações lineares entre variáveis. Inicia formulando uma equação linear $y = ax + b$, e busca ajustar os coeficientes m e b para minimizar a soma dos quadrados dos resíduos. Durante o ajuste, são feitas estimativas iniciais, os resíduos são calculados e os coeficientes são iterativamente refinados. Os coeficientes m e b têm

interpretações específicas. Após o ajuste, o modelo é utilizado para fazer previsões com base em novos dados (SILVA; ALMEIDA, 2019; FILHO et al., 2011).

Para a aplicação do algoritmo de regressão linear em específico, foi utilizado um outro *dataSet*, sendo ele o *dataSet BostonHousing* que pode ser acessado no link https://github.com/selva86/datasets/blob/master/BostonHousing.csv. Esse conjunto de dados é geralmente utilizado para problemas de regressão, principalmente na área acadêmica, ele consiste em informações coletadas nos anos 1970 pelo Serviço de Censo dos EUA sobre áreas residenciais em Boston. Com 13 atributos distintos, como taxa de criminalidade e quantidade de quartos, o objetivo principal é prever o valor mediano das casas ocupadas pelos proprietários em milhares de dólares. Permite explorar e compreender as relações entre características urbanas e os preços das casas, embora sua idade possa limitar a representação atual do mercado imobiliário (CARLOS, 2019; ROSA, 2020).

O código utilizado para fazer a regressão é apresentado abaixo.

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

modelo_regressao = LinearRegression()

modelo_regressao.fit(x_treinamenReg, y_treinamenReg)

y_pred = modelo_regressao.predict(x_testeReg)

erro_quadratico = mean_squared_error(y_testeReg, y_pred)
print(f"Erro Quadrático Médio: {erro_quadratico}")
```

5.6.1 Pré processamento específico

Para o conjunto de dados *BostonHousing*, considerando a aplicação do algoritmo de regressão linear, o pré-processamento realizado foi somente a remoção de *outliers*. Isso porque, *outliers* podem distorcer a estimativa dos parâmetros do

modelo, especialmente em modelos sensíveis a valores extremos, como a regressão linear (PERES, 2021). Eles podem ter um efeito desproporcional na estimativa dos coeficientes, levando a resultados imprecisos. A remoção de *outliers* foi feita também utilizando o algoritmo *IsolationForest* que é um algoritmo de detecção de *outliers* baseado em árvores de decisão que se destaca pela sua eficácia na identificação de valores atípicos (*outliers*) em conjuntos de dados.

```python
model = IsolationForest()
model.fit(dadosReg)
outliers = model.predict(dadosReg)
dados_sem_outliers = dadosReg[outliers == 1]
```

5.6.2 Resultados Regressão Linear - Mínimos Quadrados (padrão)

Aplicando o algoritmo de regressão linear, obteve-se o erro quadrático médio de 10,4182. Considerando que os valores da variável de interesse (MEDV - valor mediano das casas) estão na faixa de milhares de dólares, um MSE de 10,4182 sugere que, em média, as previsões do modelo estão errando em torno de $10.418 ao prever os preços das casas. Em contextos nos quais a precisão é crucial, como previsão de preços de imóveis, um erro quadrático médio de 10,4182 pode ser considerado razoável.

5.7 *Perceptron*

O *Perceptron* é um algoritmo de aprendizado de máquina que representa a unidade básica de uma rede neural. Funciona por meio de um neurônio artificial que recebe múltiplas entradas, cada uma ponderada por um peso correspondente, e aplica uma função de ativação para produzir uma saída binária. Durante o treinamento supervisionado, ele ajusta iterativamente seus pesos e termo de viés para minimizar a diferença entre as saídas previstas e as saídas desejadas (ARAUJO, 2020).

O código utilizado para fazer a classificação no *Perceptron* é apresentado abaixo.

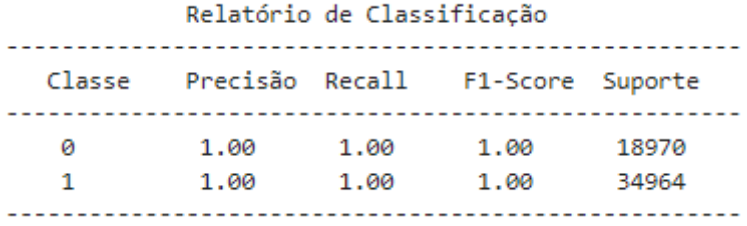
```python
from sklearn.linear_model import Perceptron

perceptron_model = Perceptron()

y_pred_percep = execute_test(perceptron_model,

X_treinamento=x_treinamento, X_teste=x_teste)

show_test(y_pred_percep)
```

5.7.1 Pré-processamento específico

A padronização dos dados pode ajudar o Perceptron a convergir mais rapidamente durante o treinamento, evitando que certos atributos dominem outros simplesmente por causa das escalas diferentes em que estão representados, o que pode acelerar a convergência do algoritmo, portanto, é comum padronizar e normalizar os dados para ajudar a melhorar o desempenho do modelo, especialmente se houver diferenças significativas nas escalas ou variâncias entre os atributos. Isso pode facilitar o processo de treinamento e melhorar a eficiência do algoritmo, podendo ser benéfica em termos de estabilidade e consistência do modelo. Sendo assim, a padronização dos dados também foi aplicada, como na subseção 5.2.1 (IPNET, 2021).

5.7.2 Resultados *Perceptron* (padrão)

Figura 23 - Relatório de Classificação *Perceptron*



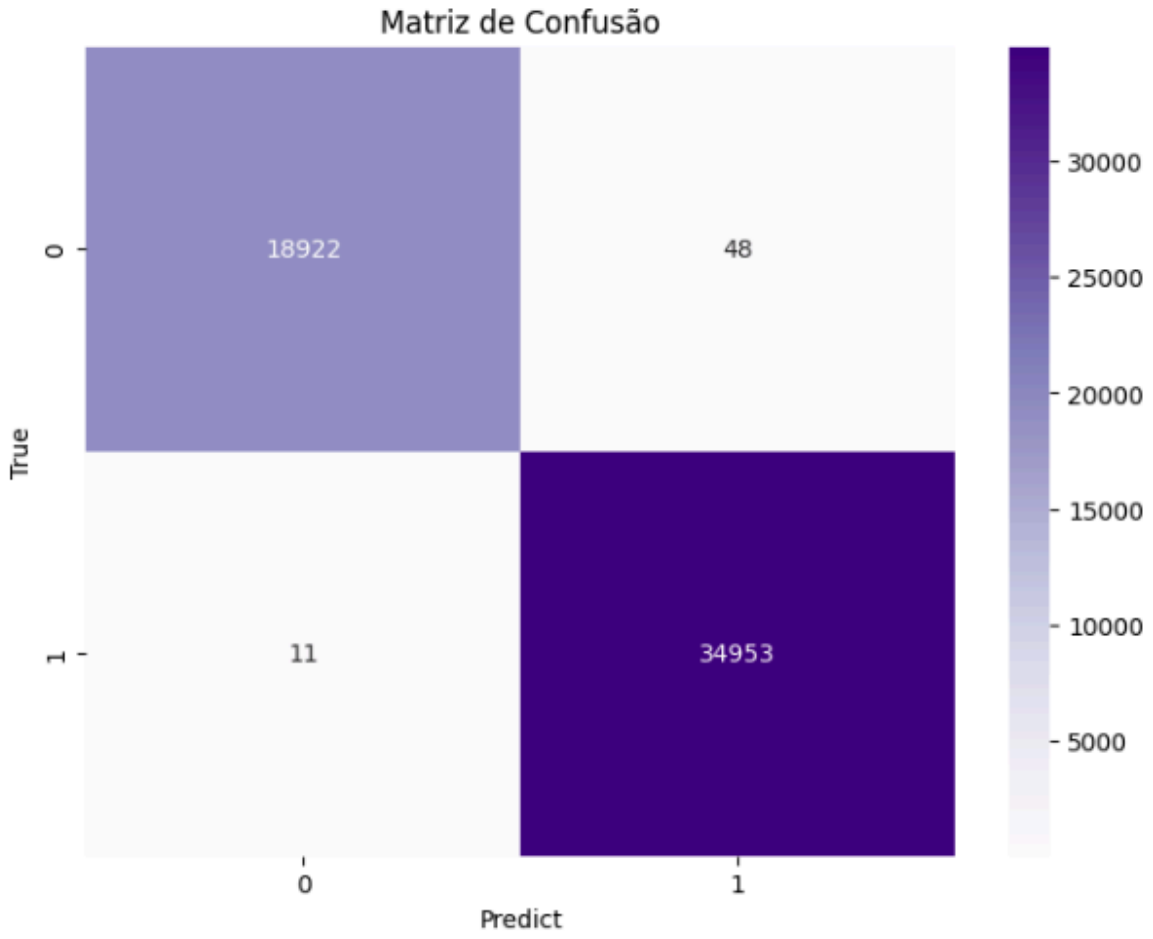
Relatório de Classificação

Classe	Precisão	Recall	F1-Score	Suporte
0	1.00	1.00	1.00	18970
1	1.00	1.00	1.00	34964

Fonte: Autoria própria (2023).

Para o *Perceptron* os resultados também foram satisfatórios, com uma acurácia de 99,8906% menor que outros teste realizados com a mesmo *dataSet*, e um *recall* de 1,00 para a classe de interesse *DDos*, porém ainda indicando que o modelo aprendeu os padrões e relações nos dados com precisão. O que capacita o modelo a realizar previsões acuradas.

Figura 24 - Matriz de confusão *Perceptron*



Fonte: Autoria própria (2023).

6 Considerações finais

Após realizados todos os testes com o *dataSet Friday-WorkingHours-Afternoon-DDos.pcap_ISCX*, foi possível observar que todos obtiveram resultados satisfatórios, mesmo alterando alguns parâmetros em alguns algoritmos. O recall da classe de interesse *DDos* em quase todos os testes ficou igual a 1, o que é satisfatório, já que é uma métrica que mede a quantidade de instâncias positivas classificadas de forma correta em relação ao total de instâncias. Outra métrica observada foi a acurácia, que é uma medida estatística que serve para avaliar a precisão dos algoritmos. Nos testes realizados todos obtiveram uma acurácia de 99% e alguns quebrados, o que é positivo. Portanto, os resultados dos modelos de algoritmo de aprendizado parecem satisfatórios, porém é fundamental destacar que a análise se concentrou nos valores de acurácia e recall, reconhecendo que há outros aspectos a serem considerados para uma avaliação abrangente do desempenho dos modelos.

Já em relação ao teste de regressão no *dataSet BostonHousing*, o resultado do erro quadrático em relação à previsão de valores de imóveis foi de 10,4182. Esse erro pode ser considerado um resultado razoável uma vez que falando em preços de imóveis ele estaria errando em $10.418 dólares ao prever preços de imóveis o que não é muito bom, por isso é considerado um erro razoável.

.

7 Referências bibliográficas

ARAUJO, R. Perceptrons. **Medium [online]**, Estados Unidos da América, nov 2020. Tecnologia. Disponível em: https://medium.com/brasil-ai/knn-k-nearest-neighbors-1-e140c82e9c4e. Acesso em: 01 dez. 2023.

ARAÚJO, Raquel. LIMPEZA DE DADOS EM UM DATASET REAL – DADOS DO TITANIC. **Hashtag [online]**, Rio de Janeiro, nov 2022. Tecnologia. Disponível em: https://www.hashtagtreinamentos.com/limpeza-de-dados-em-um-dataset-real-ciencia-dados. Acesso em: 03 dez. 2023.

CARLOS, Robert. Regressão Linear e Métricas com Python. **Medium [online]**, Estados Unidos da América, fev 2019. Tecnologia. Disponível em: https://datalivre.medium.com/regressao-linear-metricas-com-python-953af0a5dd74. Acesso em: 03 dez. 2023.

CLARKE, R. T.; BITTENCOURT, H. R. Uso de árvores de decisão na classificação de imagens digitais. **Anais XI SBSR, Belo Horizonte, Brasil**, p. 05–10, 2003. Disponível em:https://www.researchgate.net/publication/43653082_Uso_de_arvores_de_decisao_na_classificacao_de_imagens_digitais. Acesso em: 30 nov. 2023.

FILHO, D. F. et al. O que fazer e o que não fazer com a regressão: pressupostos e aplicações do modelo linear de mínimos quadrados or-dinários (mqo). **Revista Política Hoje**, v. 20, n. 1, 2011. Disponível em:https://d1wqtxts1xzle7.cloudfront.net/34605306/FIGUEIREDO__et_all_-_When_statistical_significance-libre.pdf?1409656513=&response-content-disposition=inline%3B+filename%3DO_que_Fazer_e_o_que_Nao_Fazer_com_a_Regr.pdf&Expires=1701626111&Signature=cIo1kzidtV-FXwZKqfgtbEwpsxUre3X8YFcJqYmcOcXJapmSr9wmvHiIpCjHSO0us7qryXkEJsi2x4p-tsPlGsgXRqNaFYmahQsdcM7w9e~Rg~rxPimEMV-9od9NCIP-F5CSrwCvHcjZpj9DCjWhPgBaos~bGCQEopVlk8YBlsn2kYS0tSvF38WTulT3OzlBL8orgEJ4toviBT7L4-6XIc556fKTvIVzeluyvayqM43SCtazELNdB-2485xj~4Zq-ij9fi2iFc70hn~OU2xTmOoIQxSBq32K-XrQgNSruwN2Cdh9z0pwtVbMWPcYeJFiMssfAxSLjoOMBPzPy3AmfQ__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA. Acesso em: 01 dez. 2023.

IPNET, G. P. A importância da normalização e padronização dos dados em Machine Learning. **Medium [online]**, Estados Unidos da América, fev 2021. Tecnologia. Disponível em:https://medium.com/ipnet-growth-partner/padronizacao-normalizacao-dados-machine-learning-f8f29246c12. Acesso em: 03 dez. 2023.

JOSÉ, I. Knn (k-nearest neighbors). **Medium [online]**, Estados Unidos da América, jul 2018. Tecnologia. Disponível em: https://medium.com/brasil-ai/knn-k-nearest-neighbors-1-e140c82e9c4e. Acesso em: 30 nov. 2023.

MAJUMDER, P. Gaussian naive bayes. **OpenGenus [online]**, Estados Unidos da América, fev 2020. Tecnologia. Disponível em: https://iq.opengenus.org/gaussian-naive-bayes/. Acesso em: 01 dez. 2023.

MATHWORKS, E. d. **Naive Bayes Classification**. 2023. Disponível em: https://www.mathworks.com/help/stats/naive-bayes-classification.html. Acesso em: 01 dez. 2023.

NOBLE, W. S. What is a support vector machine? **Nature biotechnology**, Nature Publishing Group UK London, v. 24, n. 12, p. 1565–1567, 2006. Disponível em: https://www.nature.com/articles/nbt1206-1565. Acesso em: 30 nov. 2023.

PACHECO, André. **Introdução ao Scikit-learn - Parte 1: uma visão geral**. 2021. Disponível em: https://computacaointeligente.com.br/outros/intro-sklearn-part-1/. Acesso em: 01 dez. 2023.

PERES, Luca. Como detectar e tratar outliers com Python. **Medium [online]**, Estados Unidos da América, set 2021. Tecnologia. Disponível em: https://medium.com/@lucapqg/como-detectar-e-tratar-outliers-com-python-ca2cf088c160. Acesso em: 03 dez. 2023.

ROSA, Diego. Prevendo o preço de imóveis com Machine Learning. **Linkedin [online]**, Estados Unidos da América, mar 2020. Tecnologia. Disponível em: https://www.linkedin.com/pulse/prevendo-o-pre%C3%A7o-de-im%C3%B3veis-com-machine-learning-diego-rosa/?originalSubdomain=pt. Acesso em: 03 dez. 2023.

ROSAL, Iury. Transformando dados categóricos em dados numéricos. **Medium [online]**, Estados Unidos da América, mar 2021. Tecnologia. Disponível em: https://medium.com/data-hackers/engenharia-de-features-transformando-dados-categ%C3%B3ricos-em-dados-num%C3%A9ricos-e5d3991df715. Acesso em: 03 dez. 2023.

SANTANA, Lamartine. Detecção de Fraude com Isolation Forest. **Medium [online]**, Estados Unidos da América, oct 2020. Tecnologia. Disponível em: https://medium.com/@lamartine_sl/detec%C3%A7%C3%A3o-de-fraude-com-isolation-forest-9ea50e4fe01b. Acesso em: 03 dez. 2023.

SANTOS, Gustavo. Estatística para Seleção de Atributos. **Medium [online]**, Estados Unidos da América, jan 2021. Tecnologia. Disponível em: https://medium.com/data-hackers/estat%C3%ADstica-para-sele%C3%A7%C3%A3o-de-atributos-81bdc274dd2c. Acesso em: 03 dez. 2023.

SCIKIT-LEARN, E. d. d. **Getting Started**. 2023a. Disponível em: https://scikit-learn.org/stable/getting_started.html. Acesso em: 01 dez. 2023.

SCIKIT-LEARN, E. d. d. **sklearn.ensemble.IsolationForest**. 2023b. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html. Acesso em: 01 dez. 2023.

SCIKIT-LEARN, E. d. d. **Support Vector Machines**. 2023c. Disponível em: https:https://scikit-learn.org/stable/modules/svm.html.html. Acesso em: 01 dez. 2023.

SILVA, C. A. P. d.; ALMEIDA, D. F. S. Métodos dos mínimos quadrados e regressão linear com ambiente jupyter e linguagem python. n. 12, 2019. Disponível em: https: 48//www.uft.edu.br/matematicaaraguaina/includes/eventos/2019/cc/CC9.pdf. Acesso em: 01 dez. 2023.

SCIKIT-LEARN, E. d. d. **sklearn.naive_bayes.CategoricalNB**. 2023d. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.CategoricalNB.html. Acesso em: 04 dez. 2023.

TIMÓTEO, Ângelo. **Tratamento de Dados Discretização: A Discretização no Tratamento de Dados**. 2023. Disponível em: https://awari.com.br/tratamento-de-dados-discretizacao-a-discretizacao-no-tratamento-de-dados/?utm_source=blog&utm_campaign=projeto+blog&utm_medium=Tratamento%20de%20Dados%20Discretiza%C3%A7%C3%A3o:%20A%20Discretiza%C3%A7%C3%A3o%20no%20Tratamento%20de%20Dados. Acesso em: 04 dez. 2023.

Vaz, Arthur L. KNN —K-Nearest Neighbor, o que é?. **Medium [online]**, Estados Unidos da América, mar 2021. Tecnologia. Disponível em: https://medium.com/data-hackers/knn-k-nearest-neighbor-o-que-%C3%A9-aeebe0f833eb. Acesso em: 03 dez. 2023.