Faculdade de Informática e Administração Paulista (FIAP)

Relatório Técnico: Aprimorando a Detecção de Fraudes na QuantumFinance

Eliézer Ramos Silveira Junior – RM (347876)
Daniel Ferreira Selga – RM (347845)
Fausto Santo Vito Delgado – RM (348252)
Renan Phelippe Eggers – RM (347812)

Prof: Felipe Souza Amaral

Dezembro/2023

RESUMO

Contexto: A QuantumFinance está comprometida em oferecer serviços financeiros seguros e eficientes para seus clientes. Uma das principais ameaças ao setor financeiro é a fraude. Sua missão é melhorar o sistema de detecção de fraudes da empresa usando técnicas avançadas de deep learning.

Tarefa: A tarefa principal é criar um modelo de deep learning para detectar transações fraudulentas. Você receberá um conjunto de dados de transações financeiras, que inclui recursos como valor da transação, horário, tipo de transação, entre outros. Algumas dessas transações são fraudulentas, enquanto outras são legítimas.

SUMÁRIO

Contexto

A QuantumFinance está focada em oferecer serviços financeiros seguros e eficientes, com a missão de melhorar seu sistema de detecção de fraudes usando técnicas avançadas de deep learning.

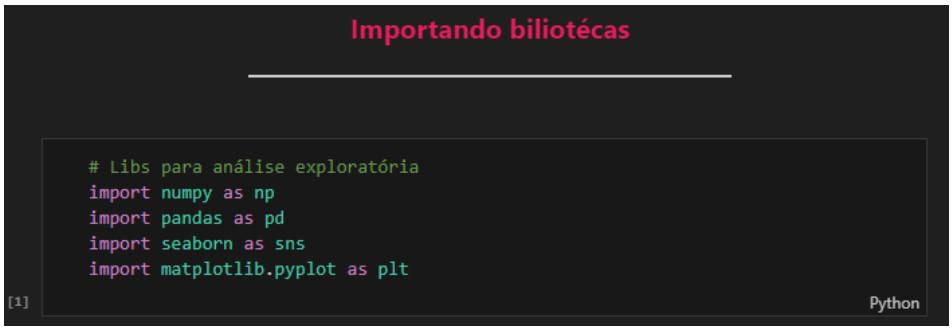
Objetivo

Criar um modelo de deep learning para detectar transações fraudulentas em um conjunto de dados fornecido, que inclui características como valor da transação, horário, tipo de transação, entre outros.

Análise Exploratória de Dados (EDA)

1.1 – Importação das bibliotecas

Para nossa análise exploratória dos dados (EDA), vamos utilizar as bibliotecas **pandas** (para manipulação dos dados), **seaborn** e **matplotlib** (para representação gráfica dos dados) e **numpy**.



1.2 – Carregamento dos dados

Vamos usar a base de dados `***dataset.csv***` que conta com as informações de supostas transações da Quantum Finance, com as informações abaixo:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Transaction ID	Date	Time	Card Type	Entry Mode	Amount	Transaction Type	Merchant Group	Transaction Country	Shipping Address	Billing Address	Gender	Age	Issuing Bank	Fraudulent
2	1	19/10/2023	05:31	MasterCard	Tap	391.56	POS	Clothing	Italy	61 Redwood St	726 Maple St	Female	55	XYZ Bank	No
3	2	16/10/2023	12:56	MasterCard	Tap	167.67	Online	Clothing	France	945 Pine St	252 Cedar St	Male	37	DEF Bank	No
4	3	29/09/2023	07:57	MasterCard	PIN	126.24	Online	Electronics	Spain	773 Oak St	177 Main St	Male	53	ABC Bank	No
5	4	13/10/2023	00:08	MasterCard	Tap	496.8	Online	Electronics	UK	436 Maple St	758 Main St	Male	21	DEF Bank	No
6	5	02/10/2023	23:19	MasterCard	Tap	446.88	POS	Grocery	France	887 Pine St	629 Main St	Male	52	XYZ Bank	No
7	6	03/10/2023	18:32	Visa	PIN	364.98	POS	Clothing	USA	512 Maple St	824 Birch St	Male	68	XYZ Bank	No
8	7	28/09/2023	03:51	MasterCard	Tap	392.7	Online	Electronics	Italy	748 Elm St	842 Main St	Female	48	ABC Bank	No
9	8	18/09/2023	21:25	Visa	PIN	488.92	POS	Grocery	Spain	141 Main St	259 Maple St	Male	65	ABC Bank	No
10	9	07/10/2023	13:17	Visa	Tap	266.52	Online	Food	Germany	459 Cedar St	331 Redwood St	Male	23	DEF Bank	No
11	10	10/10/2023	05:40	MasterCard	PIN	21.19	ATM	ATM Withdrawal	France	84 Pine St	215 Oak St	Female	29	DEF Bank	No

1.3 – Iniciando a análise exploratória (EDA)

Foi realizada uma análise exploratória básica dos dados, incluindo a visualização das primeiras linhas do conjunto de dados. Essa visualização mostra colunas como ID da transação, data, hora, tipo de cartão, modo de entrada, valor da transação, tipo de transação, grupo de comerciantes, país da transação, endereços de envio e cobrança, gênero, idade, banco emissor e se a transação é fraudulenta ou não.

Nas células analisadas, o progresso no notebook inclui:

1. Processamento Adicional dos Dados: Foi adicionada uma nova coluna ao conjunto de dados, representando o mês da transação. Isso sugere uma tentativa de extrair mais características dos dados existentes.

```python
# Criando uma coluna com as informações de número e nome do mês
dados['Mes'] = pd.to_datetime(dados['Date']).dt.strftime('%m - %B')

dados.head()
```

Python

2. Verificação das Colunas Disponíveis: A listagem das colunas do conjunto de dados foi realizada, provavelmente para entender melhor os dados disponíveis e preparar para a modelagem.

```python
# Verificando as colunas disponíveis na base de dados
dados.columns
```

Python

```
Index(['Transaction ID', 'Date', 'Time', 'Card Type', 'Entry Mode', 'Amount',
       'Transaction Type', 'Merchant Group', 'Transaction Country',
       'Shipping Address', 'Billing Address', 'Gender', 'Age', 'Issuing Bank',
       'Fraudulent', 'Mes'],
      dtype='object')
```

3. Informações Básicas do Conjunto de Dados: Foi obtida uma visão geral do conjunto de dados, incluindo o número de entradas, a contagem de valores não nulos e os tipos de dados para cada coluna.

```python
# Obtendo informações básicas (metadados) da base de dados
infos = dados.info()
infos
```

Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Transaction ID       1000 non-null   int64
 1   Date                 1000 non-null   object
 2   Time                 1000 non-null   object
 3   Card Type            1000 non-null   object
 4   Entry Mode           1000 non-null   object
 5   Amount               1000 non-null   float64
 6   Transaction Type     1000 non-null   object
 7   Merchant Group       1000 non-null   object
 8   Transaction Country  1000 non-null   object
 9   Shipping Address     1000 non-null   object
 10  Billing Address      1000 non-null   object
 11  Gender               1000 non-null   object
 12  Age                  1000 non-null   int64
 13  Issuing Bank         1000 non-null   object
 14  Fraudulent           1000 non-null   object
 15  Mes                  1000 non-null   object
dtypes: float64(1), int64(2), object(13)
memory usage: 125.1+ KB
```

4. Resumo Estatístico das Variáveis Numéricas: Preparação para visualizar um resumo estatístico, que é uma etapa crucial para entender a distribuição e as características das variáveis numéricas.

```python
# Resumo estatístico das variáveis numéricas
resumo_var_numericas = dados.describe().round(2)
resumo_var_numericas
```

Python

	Transaction ID	Amount	Age
count	1000.00	1000.00	1000.00
mean	500.50	266.14	44.66
std	288.82	138.78	15.18
min	1.00	20.18	18.00
25%	250.75	147.90	31.00
50%	500.50	268.36	45.50
75%	750.25	380.53	57.00
max	1000.00	499.69	70.00

5. Análise das Variáveis Categóricas: Foi realizada uma contagem de valores únicos para variáveis categóricas e uma discussão sobre elas. Isso inclui variáveis como tipo de cartão, modo de entrada, tipo de transação, grupo comercial, país da transação, endereços de envio e cobrança, gênero, banco emissor e a classificação de fraudulência.

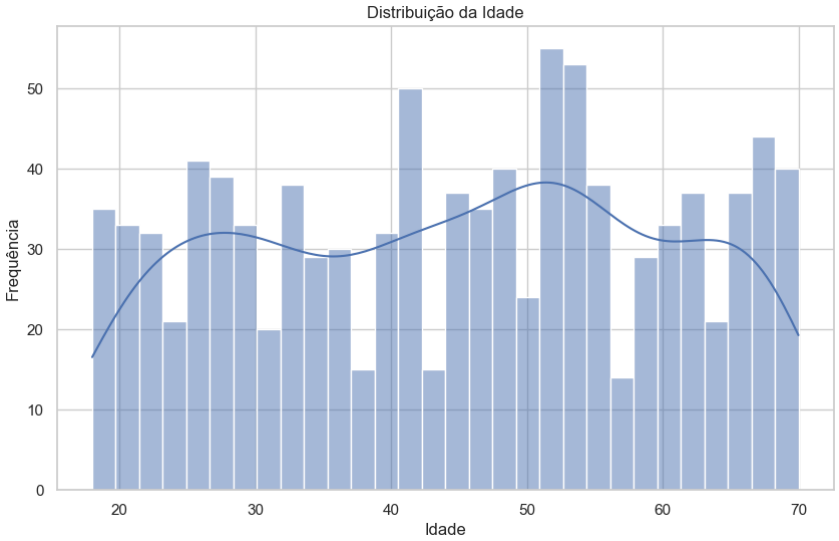
```python
# Contagem de valores únicos para variáveis categóricas
qtd_unica = dados.select_dtypes(include=['object', 'category']).nunique()
qtd_unica
```

Python

```
Date                    36
Time                   716
Card Type                2
Entry Mode               2
Transaction Type         3
Merchant Group           6
Transaction Country      7
Shipping Address       938
Billing Address        939
Gender                   2
Issuing Bank             3
Fraudulent               2
Mes                      2
dtype: int64
```

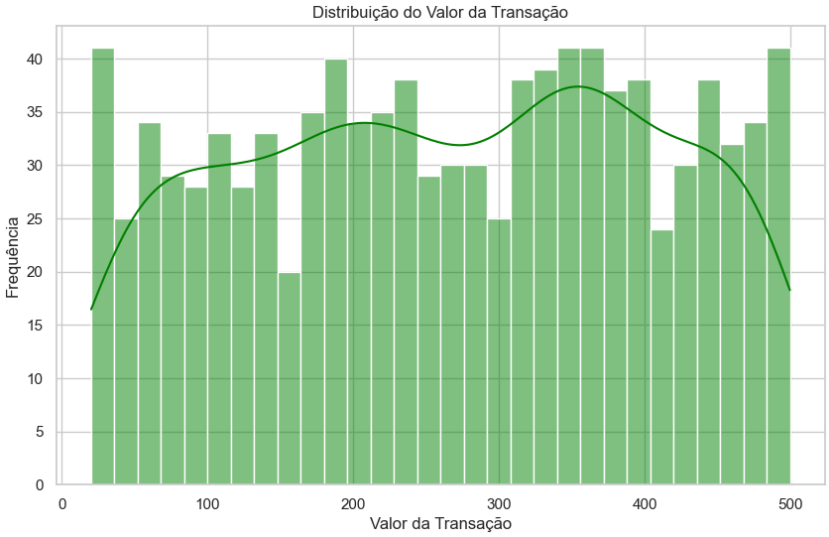
6. Vemos 3 distribuições na nossa EDA.
   • Distribuição de Idade
   • Distribuição de **Amount** (Valor da Transação)
   • Boxplot do valor de transação por transação fraudulenta

Distribuição por Idade



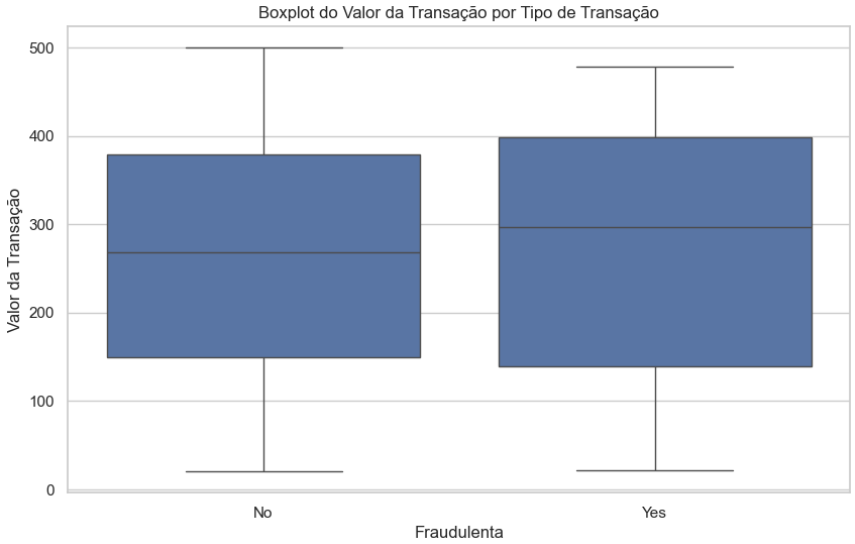
A idade dos clientes varia bastante, com uma distribuição relativamente uniforme entre 18 e 70 anos. A maioria dos clientes está na faixa dos 30 a 60 anos.

Distribuição por Amount (Valor da Transação)



O valor das transações apresenta uma distribuição ampla. A maioria das transações parece estar abaixo de 400, com uma distribuição relativamente uniforme.

Boxplot do valor da Transação por tipo de transação



O boxplot mostra a distribuição do valor das transações para casos fraudulentos e não fraudulentos.
Parece haver uma sobreposição significativa na distribuição de valores entre transações fraudulentas e legítimas. No entanto, transações fraudulentas tendem a ter uma ligeira concentração em valores mais baixos.

Essas etapas são fundamentais na fase de pré-processamento e análise exploratória de dados. Para completar o relatório técnico, ainda preciso identificar detalhes sobre a modelagem, escolha de arquitetura, hiperparâmetros e resultados de desempenho.

1.4 – Conclusão da EDA

- A análise exploratória revela um conjunto de dados com uma variedade de características numéricas e categóricas.

- A idade dos clientes e os valores das transações mostram distribuições variadas, o que pode ser útil para identificar padrões de transações fraudulentas.

- Existe um desequilíbrio significativo entre transações fraudulentas e legítimas, o que é um fator crítico a considerar ao treinar modelos de detecção de fraudes.

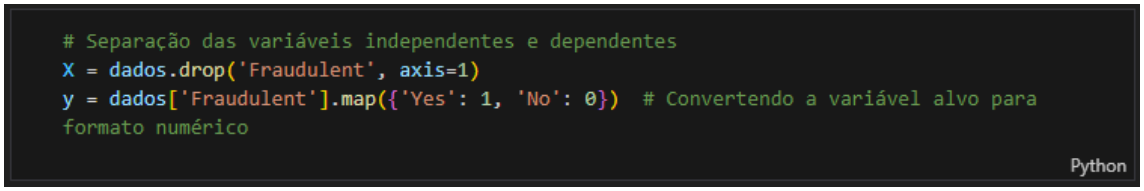
Pré-processamento de Dados

2.1 – Importação das bibliotecas

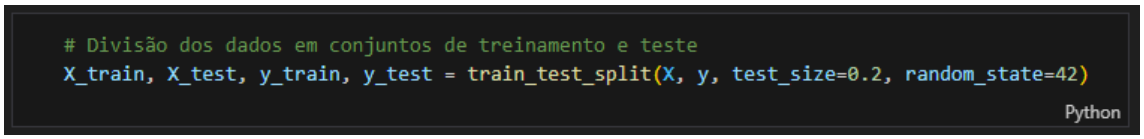
Para realizar nosso pré-processamento, vamos usar as libs:
'classification_report', 'confusion_matrix', 'roc_auc_score', 'StandardScaler', 'OneHotEncoder', 'train_test_split', 'ColumnTransformer', 'Pipeline'
Todas essas libs fazem parte do pacote ***'sklearn'***

2.2 – Variáveis independentes e dependentes

Separamos nossas variáveis em dois grupos. Independentes e dependentes. No nosso caso, apenas a variável ***'Fraudulent'*** é dependente e ficará sendo nosso eixo Y. Todas as outras variáveis estarão no eixo X e serão independentes.



Também vamos dividir nosso conjunto de treinamento e teste.
Mantendo o conjunto de treinamento com 80% e o conjunto de teste com 20% da nossa base de dados.



Depois disso fizemos uma seleção das colunas numéricas e categóricas.

```python
# Seleção de colunas numéricas e categóricas
colunas_numericas = X_train.select_dtypes(include=['int64', 'float64']).columns
colunas_categoricas = X_train.select_dtypes(include=['object']).columns
```

Python

Criamos um transformador de coluna para processamento das colunas numéricas e categóricas.

```python
# Normalização para colunas numéricas
transformador_numerico = Pipeline(steps=[
    ('scaler', StandardScaler())
])

# Codificação one-hot para colunas categóricas
transformador_categorico = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Combinando transformadores em um único transformador de coluna
preprocessor = ColumnTransformer(
    transformers=[
        ('num', transformador_numerico, colunas_numericas),
        ('cat', transformador_categorico, colunas_categoricas)
    ])
```

Python

As colunas numéricas passam por uma normalização usando StandardScaler para garantir que todas as características numéricas contribuam igualmente para o modelo.

As colunas categóricas são transformadas usando a codificação one-hot através do ***'OneHotEncoder'***, permitindo que o modelo processe informações categóricas.

2.3 – Aplicação do pré-processamento dos dados

O transformador de coluna foi aplicado aos conjuntos de dados de treinamento e teste.

```python
# 5. Aplicando o pré-processamento aos dados de treinamento e teste
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)
```

Python

Após o pré-processamento, os conjuntos de treinamento e teste têm 800 e 200 registros, respectivamente, cada um com 2206 características (após a codificação onde-hot)

```python
# Exibindo as dimensões dos conjuntos processados
X_train_processed.shape, X_test_processed.shape
```

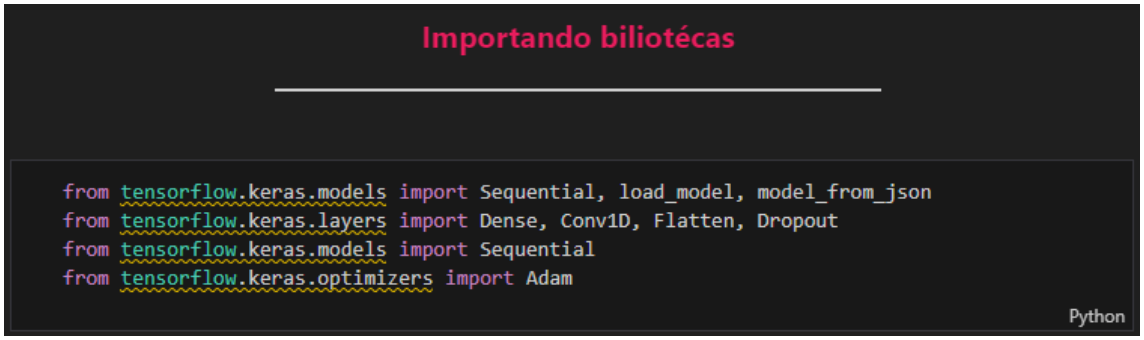
Python

```
((800, 2208), (200, 2208))
```

Modelo de Deep Learning (CNN)

3.1 – Importações das bibliotecas

Para realizar nosso pré-processamento, vamos usar as libs:
'Sequential', 'load_model', 'model_from_json', 'Dense', 'Conv1D', 'Flatten', 'Dropout', 'Sequential', 'Adam'
Todas essas libs fazem parte do pacote ***'tensorflow'***



3.2 – Construção do modelo com CNN

Para construir o modelo vamos converter e redimensionar nossos dados. Os mesmos devem ser redimensionados para o formato necessário para uma CNN. Como os dados são unidimensionais, vamos adicionar uma dimensão extra.

```python
# Convertendo as matrizes esparsas em arrays densos
X_train_dense = X_train_processed.toarray()
X_test_dense = X_test_processed.toarray()
```
Python

```python
# Redimensionamento dos dados para o formato necessário para uma CNN
X_train_reshaped = X_train_dense.reshape(X_train_dense.shape[0], X_train_dense.shape[1], 1)
X_test_reshaped = X_test_dense.reshape(X_test_dense.shape[0], X_test_dense.shape[1], 1)
```
Python

Depois de converter e redimensionar os dados, vamos dividir o conjunto entre treinamento e validação.

```python
X_train_final, X_val, y_train_final, y_val = train_test_split(X_train_reshaped, y_train,
test_size=0.2, random_state=42)
```
Python

Após isso, construímos o modelo CNN sequencial com os parâmetros:

```python
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train_reshaped.
shape[1], 1)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # Sigmoid para classificação binária
```

Python

1. ***model = Sequential()*:** Cria um modelo sequencial. Em Keras, um modelo sequencial é uma pilha linear de camadas. Esse tipo de modelo é muito usado quando há apenas uma entrada e uma saída, e as camadas são empilhadas de maneira sequencial.

2. ***model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train_reshaped.shape[1], 1))):*** Adiciona uma camada convolucional 1D ao modelo. Esta camada é frequentemente usada em dados sequenciais (como séries temporais ou dados de texto). Aqui, são utilizados 64 filtros com um tamanho de kernel de 3 e a função de ativação ReLU (Rectified Linear Unit). O ***input_shape*** é definido de acordo com o formato dos dados de treinamento, assumindo que ***X_train_reshaped*** é um array numpy que contém os dados de entrada já formatados adequadamente para a rede.

3. ***model.add(Dropout(0.5)):*** Adiciona uma camada de dropout com uma taxa de 0.5. O dropout é uma técnica de regularização usada para prevenir o overfitting. Durante o treinamento, aleatoriamente alguns nós da camada anterior são "desligados" (ou seja, seus pesos são ignorados) com a taxa especificada (neste caso, 50%).

4. ***model.add(Flatten()):*** Adiciona uma camada que "achata" os dados de entrada. É usado para transformar os dados multidimensionais em um vetor unidimensional, que pode ser usado em camadas densas (fully connected).

5. ***model.add(Dense(50, activation='relu')):*** Adiciona uma camada densa (ou totalmente conectada) com 50 neurônios e a função de ativação ReLU.

6. ***model.add(Dense(1, activation='sigmoid')):*** Adiciona outra camada densa, mas desta vez com apenas um neurônio e a função de ativação sigmoid. Esta configuração é típica para a camada de saída em tarefas de classificação binária, onde o objetivo é produzir uma única saída representando a probabilidade de a entrada pertencer a uma das duas classes.

Em resumo, este modelo é uma combinação de camadas convolucionais e densas, com regularização via dropout, projetado para tarefas de classificação binária.

3.3 – Treinamento do modelo

Treinamos nosso modelo usando o conjunto de treinamento com os ajustes de hiperparâmetros, usando o conjunto de validação.

```
history = model.fit(X_train_final, y_train_final, epochs=5, batch_size=32, validation_data=
(X_val, y_val), verbose=1)
```

Python

```
Epoch 1/5
20/20 [==============================] - 2s 77ms/step - loss: 0.5345 - accuracy: 0.9203 - val_loss:
Epoch 2/5
20/20 [==============================] - 1s 71ms/step - loss: 0.2040 - accuracy: 0.9547 - val_loss:
Epoch 3/5
20/20 [==============================] - 1s 71ms/step - loss: 0.1882 - accuracy: 0.9547 - val_loss:
Epoch 4/5
20/20 [==============================] - 1s 70ms/step - loss: 0.1796 - accuracy: 0.9547 - val_loss:
Epoch 5/5
20/20 [==============================] - 1s 71ms/step - loss: 0.1791 - accuracy: 0.9547 - val_loss:
```

Epoch 1/5 até Epoch 5/5: Cada "Epoch" representa uma passagem completa através do conjunto de dados de treinamento. Este modelo foi configurado para ser treinado por 5 épocas, então você vê o progresso desde a primeira até a quinta época.

20/20 [==]: Indica o número de lotes (batches) processados durante a época. Neste caso, o conjunto de dados de treinamento foi dividido em 20 lotes, e todos os 20 lotes foram processados. O progresso é mostrado visualmente pela barra.

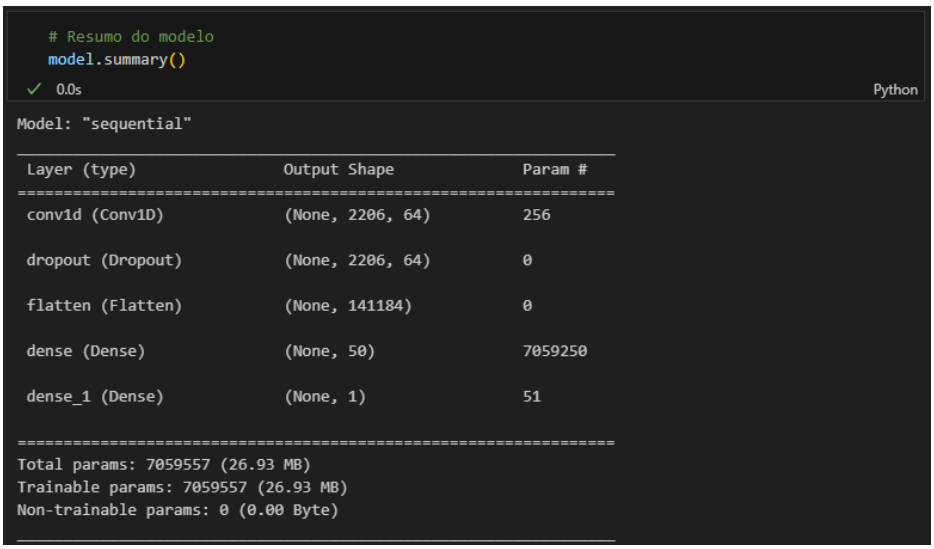
- 2s 75ms/step - 1s 69ms/step: Mostra o tempo total levado para completar a época e o tempo médio por passo (ou lote). Por exemplo, a primeira época levou 2 segundos no total, com uma média de 75 milissegundos por lote.

loss: 0.5050 - accuracy: 0.9203: Estes são os valores da função de perda (loss) e da acurácia no conjunto de treinamento. A função de perda quantifica o quão bem o modelo está performando (quanto menor, melhor), e a acurácia é a proporção de previsões corretas. Aqui, a perda inicial é 0.5050 e a acurácia é 92.03% na primeira época.

val_loss: 0.2232 - val_accuracy: 0.9688: São os valores da função de perda e da acurácia no conjunto de validação. O conjunto de validação é um conjunto de dados separado, não utilizado no treinamento, que serve para testar o desempenho do modelo. A perda de validação inicial é 0.2232 e a acurácia é 96.88%.

Ao longo das épocas, você pode observar como a função de perda e a acurácia mudam. Idealmente, a função de perda deveria diminuir e a acurácia aumentar com o tempo, tanto no conjunto de treinamento quanto no de validação. Aqui, a acurácia no conjunto de treinamento começa em 92.03% e fica estável em torno de 95.47%, enquanto a acurácia de validação se mantém constante em 96.88%. A função de perda diminui gradualmente tanto no treinamento quanto na validação, indicando que o modelo está aprendendo e melhorando seu desempenho ao longo do tempo.

Aqui temos um resumo do nosso modelo.



```
# Resumo do modelo
model.summary()
```

✓ 0.0s Python

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 2206, 64)          256

 dropout (Dropout)           (None, 2206, 64)          0

 flatten (Flatten)           (None, 141184)            0

 dense (Dense)               (None, 50)                7059250

 dense_1 (Dense)             (None, 1)                 51

=================================================================
Total params: 7059557 (26.93 MB)
Trainable params: 7059557 (26.93 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

1. **Model:**
    a. "sequential": Indica que o modelo é do tipo "Sequential", que é uma pilha linear de camadas.

2. **Camadas do Modelo:**
    a. conv1d (Conv1D): É a primeira camada do modelo, uma camada convolucional 1D. A saída desta camada é de dimensão (None, 2206, 64). "None" refere-se ao tamanho do lote (batch size), que pode variar. 2206 é o tamanho da dimensão de saída após a aplicação da convolução, e 64 é o número de filtros utilizados. Esta camada tem 256 parâmetros treináveis.
    b. dropout (Dropout): Uma camada de dropout que ajuda a prevenir o overfitting, mantendo a mesma forma de saída da camada anterior e não adicionando novos parâmetros.
    c. flatten (Flatten): Esta camada achata a saída da camada anterior para um vetor unidimensional de tamanho 141184.
    d. dense (Dense): Uma camada densa com 50 neurônios. A saída desta camada é um vetor de dimensão 50. Esta camada tem 7,059,250 parâmetros treináveis.
    e. dense_1 (Dense): Outra camada densa, mas com apenas 1 neurônio, típica para a saída em tarefas de classificação binária. Tem 51 parâmetros treináveis.

3. **Resumo dos Parâmetros:**
    a. Total params: 7059557: O modelo tem um total de 7,059,557 parâmetros treináveis.
    b. Trainable params: 7059557: Todos os parâmetros são treináveis.
    c. Non-trainable params: 0: Não há parâmetros não treináveis.

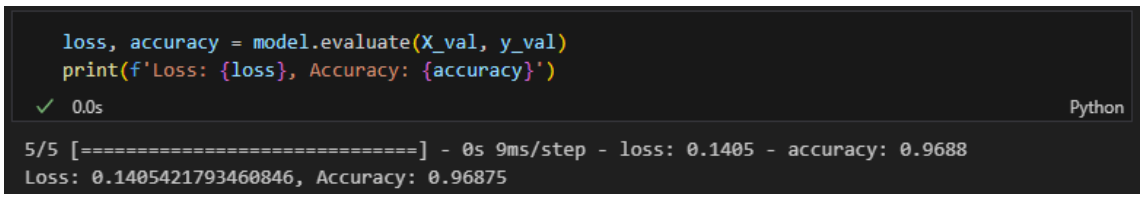
4. **Tamanho do Modelo:**
    a. O tamanho do modelo é aproximadamente 26.93 MB, indicando o espaço necessário para armazenar os pesos do modelo.

Em resumo, parece que o modelo está treinado de forma eficiente para a tarefa de classificação binária, mas é sempre importante realizar avaliações adicionais e considerar ajustes para garantir sua eficácia e robustez.

Avaliação de Desempenho do Modelo

4.1 – Avaliação de desempenho

Após treinar o modelo, vamos utilizar o conjunto de validação para avaliar seu desempenho. Isso geralmente é feito usando métricas como precisão, recall, F1-score e AUC-ROC. No TensorFlow/Keras, vamos fazer isso usando o método evaluate do modelo.



1. ***loss, accuracy = model.evaluate(X_val, y_val):*** Esta linha de código está usando o método evaluate do modelo para calcular a função de perda (loss) e a acurácia (accuracy) no conjunto de validação. '***X_val'*** e '***y_val***' são, respectivamente, os dados de entrada e os rótulos correspondentes do conjunto de validação.

2. ***print(f'Loss: {loss}, Accuracy: {accuracy}'):*** Após a avaliação, os valores de perda e acurácia são impressos. Esses valores são um indicador do desempenho do modelo no conjunto de validação.

3. ***5/5 [==] - 0s 9ms/step - loss: 0.1405 - accuracy: 0.9688:***
    a. **5/5:** Indica que o conjunto de validação foi dividido em 5 lotes (batches) e todos foram processados.

    b. **0s 9ms/step:** Mostra o tempo total levado para a avaliação e o tempo médio por lote.

    c. **loss: 0.1405:** O valor da função de perda no conjunto de validação. Uma função de perda baixa indica que as previsões do modelo estão, em média, próximas dos valores reais.

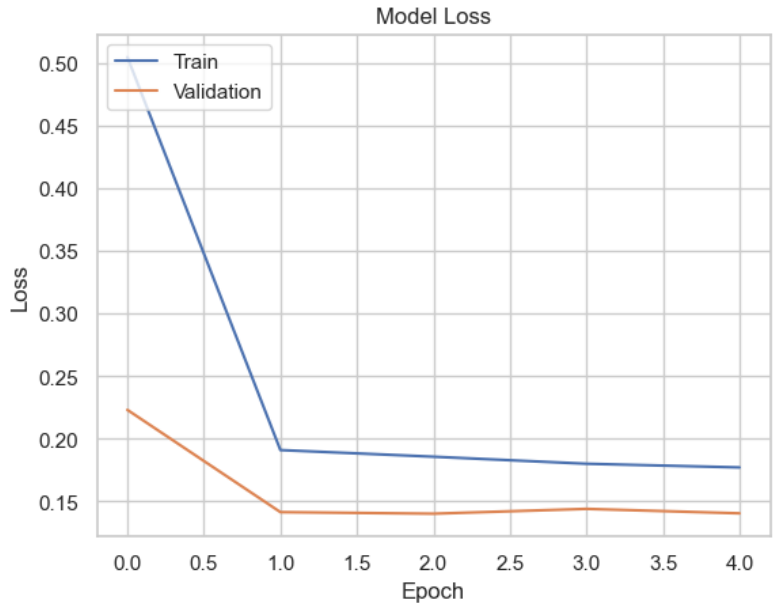
    d. **accuracy: 0.9688:** A acurácia do modelo no conjunto de validação, expressa como uma fração de 1. Neste caso, 0.9688 ou 96.88%, indicando que o modelo fez previsões corretas em 96.88% dos casos no conjunto de validação.

4. ***Loss: 0.1405421793460846, Accuracy: 0.96875:*** Estes são os valores numéricos exatos para a função de perda e acurácia que foram impressos pelo comando print. Eles confirmam que a função de perda é de aproximadamente 0.1405 e a acurácia é de cerca de 96.875%.

Esses resultados sugerem que o modelo tem um bom desempenho no conjunto de validação, com alta acurácia e uma função de perda relativamente baixa. Isso geralmente é um indicativo de que o modelo aprendeu padrões significativos dos dados e está fazendo previsões precisas.

4.2 – Considerações do modelo

Model Loss



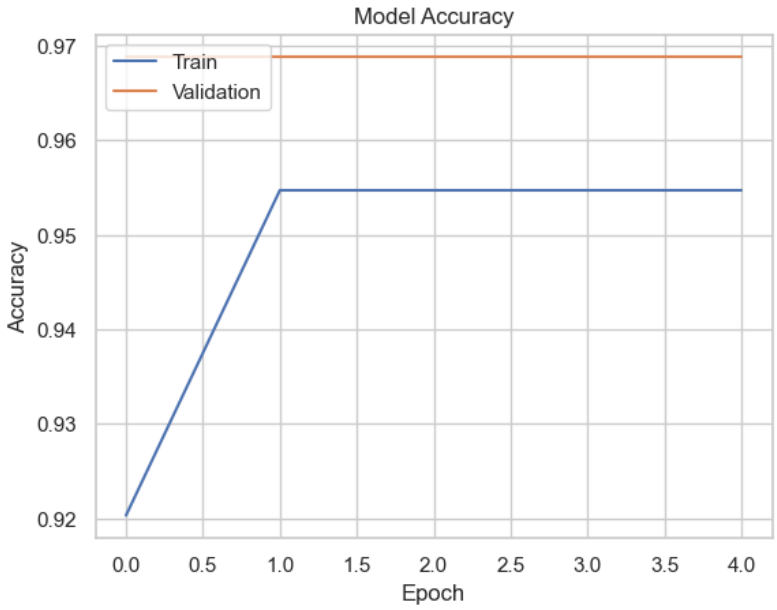
Linha Azul (Train Loss): Esta linha representa a perda do modelo no conjunto de dados de treinamento. Podemos ver que a perda diminui de maneira constante à medida que o número de épocas aumenta, o que indica que o modelo está aprendendo e melhorando seu desempenho no conjunto de treinamento ao longo do tempo.

Linha Laranja (Validation Loss): Esta linha representa a perda do modelo no conjunto de dados de validação. Observamos que a perda diminui inicialmente, mas depois começa a se estabilizar até quinta época.

Overfitting: O modelo está se ajustando bem aos dados de treinamento, sem indicativo de overfitting, que acontece quando um modelo aprende padrões específicos do conjunto de treinamento que não se generalizam bem para novos dados.

Conclusão: O gráfico da esquerda mostra que a perda de treinamento (azul) e a perda de validação (laranja) diminuem à medida que o número de épocas aumenta. A perda de validação inicialmente diminui a um ritmo mais lento em comparação com a perda de treinamento, mas eventualmente, ambas parecem se estabilizar, com a perda de validação sendo ligeiramente maior do que a perda de treinamento. Isso é um bom sinal de que o modelo não está sofrendo de overfitting severo, já que a perda de validação segue a de treinamento de perto e não há um aumento significativo ou uma divergência grande entre as duas.

Model Accuracy



Linha Azul (Train Loss): Mostra um aumento na precisão ao longo do tempo no conjunto de treinamento, o que indica que o modelo está se ajustando bem aos dados de treinamento e aprendendo a classificar corretamente as amostras de treino.

Linha Laranja (Validation Loss): A precisão no conjunto de validação permanece relativamente constante e não acompanha os aumentos observados na precisão de treinamento. A precisão de validação é consistentemente mais baixa do que a precisão de treinamento.

Avaliação de Desempenho: A precisão de validação parece ter atingido um platô, o que pode sugerir que ajustes adicionais no modelo ou nos dados são necessários para obter melhorias no desempenho.

Conclusão: O gráfico da direita mostra que a acurácia de treinamento (azul) e a acurácia de validação (laranja) aumentam com o tempo. A acurácia de validação parece começar um pouco abaixo da acurácia de treinamento, mas rapidamente converge para um valor semelhante. Ambas as curvas parecem se estabilizar rapidamente, o que pode indicar que o modelo rapidamente alcançou uma capacidade de generalização boa e que mais épocas de treinamento podem não ser necessárias. A acurácia parece ser muito alta (próxima de 97%), o que sugere que o modelo tem um desempenho muito bom na tarefa para a qual foi treinado.

4.3 – Métricas de desempenho

Previsões e Classes Previstas, Métricas e Matriz de Confusão

```
# Fazendo previsões no conjunto de teste
predictions = model.predict(X_test_reshaped)
predicted_classes = np.round(predictions).flatten()  # Arredondando as previsões para obter
a classe

# Calculando e imprimindo as métricas de desempenho
print(classification_report(y_test, predicted_classes))
print("Matriz de Confusão:\n", confusion_matrix(y_test, predicted_classes))
print("AUC-ROC:", roc_auc_score(y_test, predicted_classes))
print()
```

✓ 0.1s Python

```
7/7 [==============================] - 0s 8ms/step
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       192
           1       0.00      0.00      0.00         8

    accuracy                           0.96       200
   macro avg       0.48      0.50      0.49       200
weighted avg       0.92      0.96      0.94       200

Matriz de Confusão:
 [[192   0]
 [  8   0]]
AUC-ROC: 0.5
```

1. **Previsões e Classes Previstas:**
    a. predictions = model.predict(X_test_reshaped): O modelo faz previsões no conjunto de teste X_test_reshaped.

    b. predicted_classes = np.round(predictions).flatten(): As previsões, que provavelmente são probabilidades, são arredondadas para 0 ou 1 para representar as classes previstas. O método flatten é usado para transformar a saída em um vetor unidimensional.

2. **Métricas de Desempenho:**
    a. Classification Report: Fornece uma visão detalhada do desempenho do modelo para cada classe (0 e 1 neste caso).

b. Para a classe 0 (provavelmente a classe majoritária), o modelo tem alta precisão (96%), recall (100%), e F1-score (98%).

c. Para a classe 1 (classe minoritária), o modelo tem 0% em todas as métricas, indicando que ele não conseguiu identificar corretamente nenhum exemplo desta classe.

d. A acurácia geral é de 96%, mas isso é enganoso devido ao desequilíbrio de classes.

e. As médias macro e ponderada mostram uma diferença significativa, refletindo o desempenho desigual entre as classes.

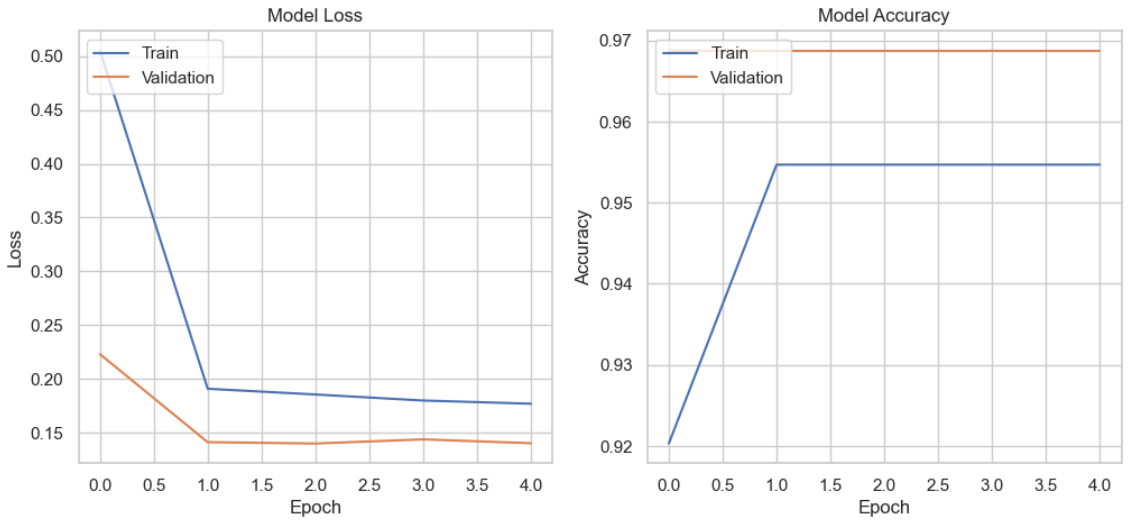
3. **Matriz de Confusão:**
   a. A matriz mostra 192 verdadeiros negativos (TN) e 0 verdadeiros positivos (TP).

   b. Existem 8 falsos negativos (FN), significando que 8 instâncias da classe 1 foram incorretamente classificadas como classe 0.

   c. Não há falsos positivos (FP), pois nenhuma instância da classe 0 foi classificada erroneamente como classe 1.

   d. AUC-ROC: O valor de 0.5 para a área sob a curva do operador de característica do receptor (ROC AUC) indica um desempenho não melhor do que um classificador aleatório. Isso é consistente com a incapacidade do modelo de identificar corretamente a classe 1.

Visualização do model LOSS e ACCURACY:



Conclusões:

- O modelo é altamente eficaz em prever a classe 0, mas pouco eficiente para a classe 1.

- A alta acurácia é enganosa devido ao desequilíbrio de classes (muitos exemplos da classe 0 e poucos da classe 1).

- O modelo aparenta estar enviesado para a classe majoritária (0), possivelmente devido a um desequilíbrio de classes no conjunto de treinamento.

- As métricas macro e ponderada mostram um desempenho muito mais fraco do que a acurácia geral sugere.

- Pode ser necessário reavaliar a abordagem de modelagem, incluindo o balanceamento de classes, ajuste de parâmetros, ou revisão da arquitetura do modelo, para melhorar o reconhecimento da classe minoritária.