

Gabriel Zuany Duarte Vargas

Projeto IA - IndustriALL

Vitória, ES

2023

1 Introdução

Este documento apresenta o *Projeto IA - IndustriALL*. O objetivo deste é explicar e exibir os métodos e ideias utilizadas para cumprir o desafio proposto pela equipe de IA (Inteligência Artificial) da IndustriALL. O projeto proposto foi construir um modelo que, a partir de dados coletados de sensores em um equipamento industrial, pudesse aferir e prever possíveis falhas e anormalidades na máquina.

Além desta introdução, este documento está organizado da seguinte forma: a Seção 2 apresenta os métodos e adotados na leitura dos dados fornecidos; a Seção 3 apresenta os principais processos na filtragem, limpeza e adequações dos valores contidos nos arquivos de leitura; a Seção 4 apresenta a etapa de análise exploratória, em que foram utilizadas técnicas de estatística descritiva para entender a relação e contexto dos dados. A Seção 5 abrange a explicação sobre a escolha do modelo de Machine Learning aplicado, um detalhamento teórico e de implementação sobre o mesmo, e, ainda, os resultados obtidos. E, por fim, a Seção 6 exibe uma implementação extra, a qual simula uma situação real de fluxo dos dados com Apache Kafka.

2 Leitura dos Dados

Este capítulo abrange a etapa de leitura dos dados e algumas técnicas que auxiliaram o desenvolvimento nesse quesito.

2.1 Origem dos Dados

Os dados foram fornecidos pelo time da IndustriALL no formato de csv (Comma Separated Values) para cada sensor que monitorava o equipamento.

Documentos > **Processo Seletivo - IndustriALL**




 Nome ▾
 dados.zip
 Desafio PS.pdf

Figura 1 – Fonte dos Dados

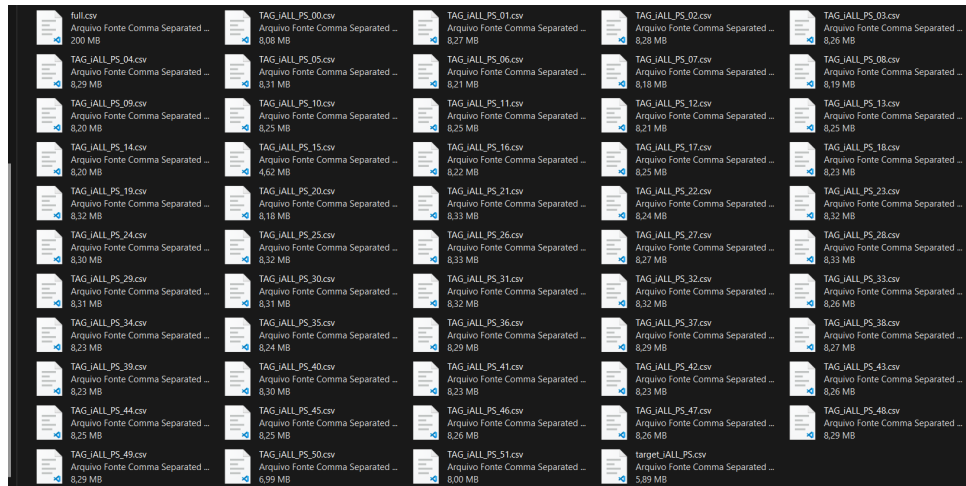


Figura 2 – Arquivos

2.2 Agregação dos Dados dos Sensores

A priori, havia duas alternativas para agregar os dados individuais em apenas um **Dataframe**: agregar dinamicamente, sempre lendo um a um e adicionando no Dataframe geral; ou construir previamente esse dataset completo e apenas ler. A primeira alternativa economiza armazenamento, mas por ser em tempo de execução, perde no desempenho geral, enquanto a segunda se comporta da forma contrária. Pensando nesses dois cenários, foi feito um script que constrói esse arquivo completo caso não o encontre.

```

rc > sensor_summarize_sequential.py > ...
1  import os
2  import time
3  import pandas as pd
4
5  DATAFOLDER = 'data/'
6  BASE_NAME = 'TAG_iALL_PS_'
7
8  start_time = time.time()
9
10 TAG_IALL_PS_00 = pd.read_csv(DATAFOLDER + 'TAG_iALL_PS_00.csv')
11 TAG_IALL_PS_00 = TAG_IALL_PS_00.set_index('timestamp')
12
13 full_df = pd.DataFrame()
14 full_df = pd.concat([full_df, TAG_IALL_PS_00], ignore_index=False)
15 for file in os.listdir(DATAFOLDER):
16     if file.endswith('00.csv'):
17         continue
18     print('Reading file: ' + file)
19     df = pd.read_csv(DATAFOLDER + file)
20     df = df.set_index('timestamp')
21     full_df = pd.concat([full_df, df[file.split('.')[0]]], ignore_index=False, axis=1)
22     if file == 'target_iALL_PS.csv':
23         full_df = full_df.rename(columns={'target_iALL_PS': 'status'})
24
25 full_df.to_csv(DATAFOLDER + 'full.csv')
26 # print(full_df.head())
27
28 end_time = time.time()
29 print('Elapsed time: ' + str(end_time - start_time) + ' seconds')

```

Figura 3 – Script (Sequencial)

```

src > sensor summarize parallel.py > ...
1  from multiprocessing import cpu_count
2  import os
3  import time
4  import pandas as pd
5  from concurrent.futures import ProcessPoolExecutor
6
7  DATAFOLDER = 'data/'
8  BASE_NAME = 'TAG_iALL_PS_'
9
10 start_time = time.time()
11
12 def read(file):
13     if file.endswith('00.csv'):
14         return None
15
16     print('Reading file: ' + file)
17     df = pd.read_csv(DATAFOLDER + file)
18     df = df.set_index('timestamp')
19
20     return df[file.split('.')[0]]
21
22 if __name__ == '__main__':
23     # Get the list of files to process
24     files_to_process = [file for file in os.listdir(DATAFOLDER) if not file.endswith('00.csv')]
25
26     # Use ProcessPoolExecutor for parallel processing
27     with ProcessPoolExecutor(cpu_count()) as executor:
28         # Map the function to process each file in parallel
29         dfs = list(executor.map(read, files_to_process))
30
31     # Create the full df by concatenating the DataFrames
32     full_df = pd.concat([pd.read_csv(DATAFOLDER + 'TAG_iALL_PS_00.csv').set_index('timestamp')] + dfs, ignore_index=False, axis=1)
33
34     # Rename the 'target_iALL_PS' column to 'status'
35     if 'target_iALL_PS.csv' in files_to_process:
36         full_df = full_df.rename(columns={'target_iALL_PS': 'status'})
37
38     # Save the result to a CSV file
39     full_df.to_csv(DATAFOLDER + 'full.csv')
40     end_time = time.time()
41     print('Elapsed time: ' + str(end_time - start_time) + ' seconds')

```

Figura 4 – Script um pouco mais refinado (Processamento Paralelo)

- Layout do arquivo gerado:

```

timestamp : object | TAG_iALL_PS_00 : float64 | TAG_iALL_PS_01 : float64 | TAG_iALL_PS_02 : float64 | TAG_iALL_PS_03 : float64 |
TAG_iALL_PS_04 : float64 | TAG_iALL_PS_05 : float64 | TAG_iALL_PS_06 : float64 | TAG_iALL_PS_07 : float64 | TAG_iALL_PS_08 : float64 |
TAG_iALL_PS_09 : float64 | TAG_iALL_PS_10 : float64 | TAG_iALL_PS_11 : float64 | TAG_iALL_PS_12 : float64 | TAG_iALL_PS_13 : float64 |
TAG_iALL_PS_14 : float64 | TAG_iALL_PS_15 : float64 | TAG_iALL_PS_16 : float64 | TAG_iALL_PS_17 : float64 | TAG_iALL_PS_18 : float64 |
TAG_iALL_PS_19 : float64 | TAG_iALL_PS_20 : float64 | TAG_iALL_PS_21 : float64 | TAG_iALL_PS_22 : float64 | TAG_iALL_PS_23 : float64 |
TAG_iALL_PS_24 : float64 | TAG_iALL_PS_25 : float64 | TAG_iALL_PS_26 : float64 | TAG_iALL_PS_27 : float64 | TAG_iALL_PS_28 : float64 |
TAG_iALL_PS_29 : float64 | TAG_iALL_PS_30 : float64 | TAG_iALL_PS_31 : float64 | TAG_iALL_PS_32 : float64 | TAG_iALL_PS_33 : float64 |
TAG_iALL_PS_34 : float64 | TAG_iALL_PS_35 : float64 | TAG_iALL_PS_36 : float64 | TAG_iALL_PS_37 : float64 | TAG_iALL_PS_38 : float64 |
TAG_iALL_PS_39 : float64 | TAG_iALL_PS_40 : float64 | TAG_iALL_PS_41 : float64 | TAG_iALL_PS_42 : float64 | TAG_iALL_PS_43 : float64 |
TAG_iALL_PS_44 : float64 | TAG_iALL_PS_45 : float64 | TAG_iALL_PS_46 : float64 | TAG_iALL_PS_47 : float64 | TAG_iALL_PS_48 : float64 |
TAG_iALL_PS_49 : float64 | TAG_iALL_PS_50 : float64 | TAG_iALL_PS_51 : float64 | status : object |

```

Figura 5 – Layout do Arquivo

3 Pré-processamento, Filtragem e Limpeza dos Dados

Nessa etapa, foram utilizados métodos como `head()`, `tail()`, `summarize()`, etc, para entender um pouco sobre os dados. A partir disso, foi feita a retirada de linhas duplicadas para evitar a interferência nos demais cálculos que virão posteriormente; em seguida, ocorreu a cautelosa remoção de linhas cujos valores eram todos nulos (já que um valor nulo para um ou mais sensores pode indicar claramente uma falha, desde que o status *não seja nulo*). Também houve a reindexação após a remoção desses valores e a promoção da coluna de *timestamp* como coluna index, transformando, assim, nosso Dataframe em uma [série temporal](#).

Ainda, houve a ordenação do Dataframe com base no index. Embora conheçamos a natureza dos dados e esperamos que esteja ordenado, não é uma garantia; e reordenando temos uma segurança maior para trabalhar com os dados na próximas etapas.

```
# Drop duplicates and NaNs (when all rows/columns are NaN)
full_df = full_df.drop_duplicates()
full_df = full_df.dropna(axis=1, how='all')
full_df = full_df.dropna(axis=0, how='all')
full_df = full_df.reset_index()

# Convert the timestamp column to datetime and set it as the index
full_df['timestamp'] = pd.to_datetime(full_df['timestamp'])
full_df = full_df.sort_values(by='timestamp')
full_df = full_df.reset_index()
full_df = full_df.set_index('timestamp') if 'timestamp' in full_df.columns else full_df

# Create a column with the status as a boolean (might be useful for plotting later)
full_df['status_bool'] = np.where(full_df['status'] == 'NORMAL', 0, 1)

# drop the columns that are not useful
full_df = full_df.drop(columns=['index'])
full_df = full_df.drop(columns=['level_0'])
```

✓ 1.6s

Figura 6 – Processamento Inicial

4 Análise Exploratória

4.1 Verificações Iniciais

As verificações iniciais incluíram a proporção de estados normais e anormais do equipamento dentro dos valores registrados para entender qual o comportamento majoritário presente.

```
status
NORMAL    205836
ANORMAL    14484
Name: count, dtype: int64
status
NORMAL    0.934259
ANORMAL    0.065741
Name: proportion, dtype: float64
```

Figura 7 – Proporção entre Status

4.2 Identificação de Outliers

Tendo em vista a quantidade de dados à disposição, não era viável analisar (ou plotar) os boxplots de cada sensor, então a decisão tomada foi aplicar uma identificação de outliers de modo analítico. Nem sempre o método analítico identifica todos os outliers (podem existir cenários contrários), entretanto, no caso vigente, os resultados foram significativamente assertivos.

- $IQR = Q3 - Q1$
- $upperbound = Q3 + 1.5 * IQR$
- $lowerbound = Q1 - 1.5 * IQR$

```
Outliers: 84262
Non normal: 14484
Non normal and outlier: 6363
```

Figura 8 – Outliers



Figura 9 – Boxplot de 'TAG iALL PS 00' (exemplo)

4.3 Correlação entre as Variáveis (Sensores)

Aqui é uma etapa essencial para a decisão acerca do modelo de machine learning que será escolhido, uma vez que entenderemos a correlação entre as variáveis que temos e o target (status). A partir do heatmap abaixo, podemos perceber que a correlação entre os dados das variáveis é, em geral, baixa (apenas a porção do meio, entre o sensor 16 e 36, possui um pouco mais de relação entre si). Assumimos, então, que se tratam de sensores majoritariamente independentes.

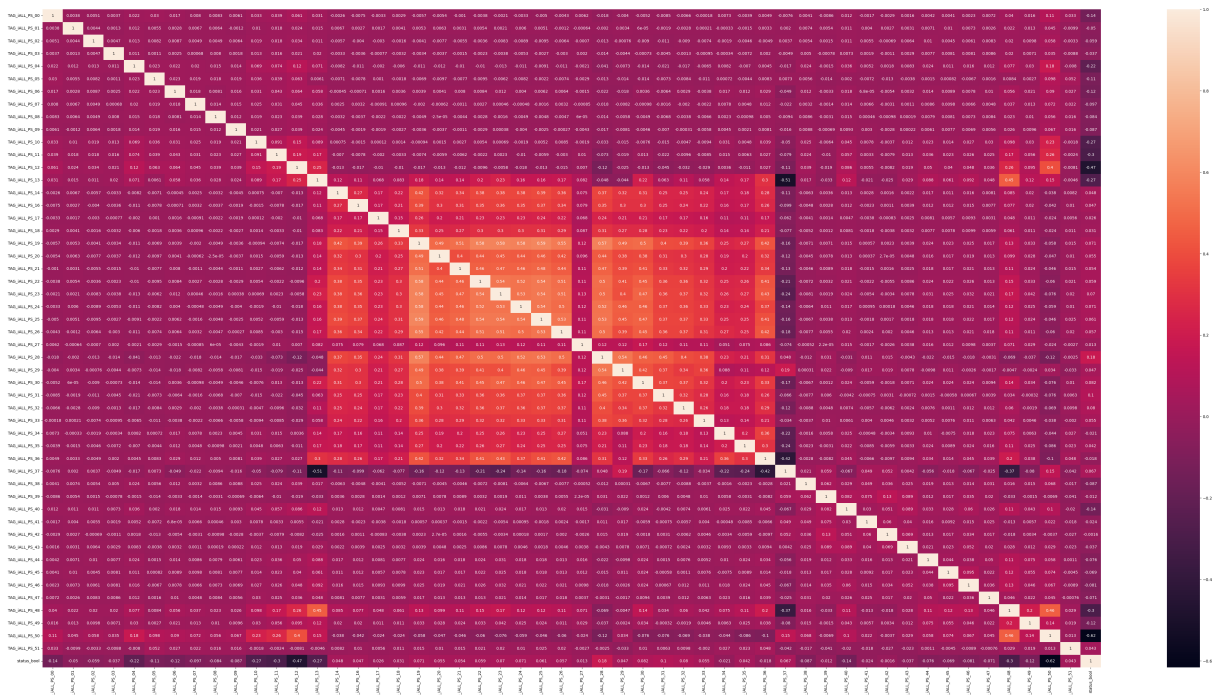


Figura 10 – Correlação entre Sensores

Analisando um a um, por meio do heatmap abaixo, percebemos que os sensores individualmente afetam pouco o status do equipamento. Logo, é um conjunto de estados de sensores que interferem no status.

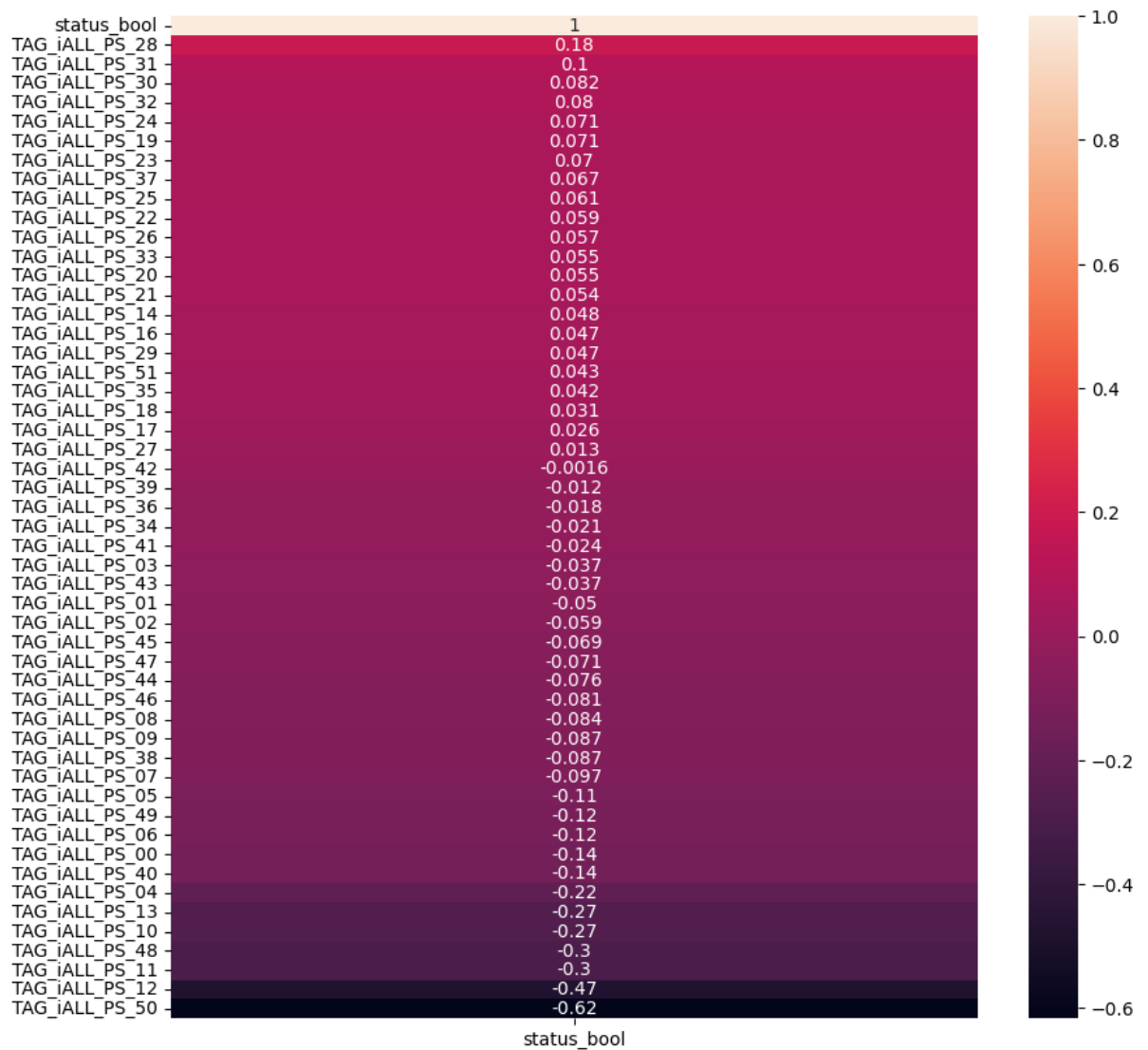


Figura 11 – Correlação entre Sensores e Status

Com isso, concluímos que temos uma [análise multivariada](#) com [features](#) independentes.

5 Modelo de Machine Learning

5.1 Naive Bayes

Dado que temos variáveis independentes, o algoritmo de Naive Bayes é uma boa escolha para o nosso problema.

5.1.1 Formulação Geral

A probabilidade condicional $P(y|X)$, representando a probabilidade da classe (evento) y dado o vetor de features X , segue o Teorema de Bayes:

$$P(y|X) = \frac{P(X|y) \cdot P(y)}{P(X)}$$

5.1.2 Independência

A característica distintiva do Naive Bayes é a suposição de independência entre as features, dada a classe. Assim, a fórmula se simplifica para:

$$P(y|X) = \frac{P(x_1|y) \cdot P(x_2|y) \cdot \dots \cdot P(x_n|y) \cdot P(y)}{P(X)}$$

5.1.3 Seleção da Classe

Para determinar a classe mais provável, simplificamos ainda mais, eliminando $P(X)$ já que não depende da classe:

$$y = \operatorname{argmax}_y (P(x_1|y) \cdot P(x_2|y) \cdot \dots \cdot P(x_n|y) \cdot P(y))$$

5.1.4 Transformação Logarítmica

Dado que as probabilidades são valores entre 0 e 1, a expressão pode ser reescrita como uma soma de logaritmos para facilitar os cálculos:

$$y = \operatorname{argmax}_y (\log(P(x_1|y)) + \log(P(x_2|y)) + \dots + \log(P(x_n|y)) + \log(P(y)))$$

5.1.5 Priori e Condicional de Classe com Distribuição Gaussiana

$P(y)$ representa a probabilidade a priori (probabilidade de um evento ocorrer com base em conhecimento ou informação prévia), refletindo a frequência de cada classe. $P(x_i|y)$

representa a probabilidade condicional de classe, muitas vezes modelada com a fórmula da distribuição Gaussiana:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

5.1.6 Passos Principais

- Treinamento: Calcule a média, variância e a frequência de cada classe.
- Previsões: Calcule a probabilidade a futura para cada classe usando a fórmula simplificada.
- Escolha da Classe: Selecciona a classe com a maior probabilidade a priori como a previsão final.

5.2 Aplicação do Modelo

A implementação do modelo se deu por meio do módulo [scikit-learn](#). É importante ressaltar que existem algumas implementações diferentes da função distribuição de probabilidade do Naive Bayes, e nesse caso foram utilizadas duas formas distintas:

- [Gaussiana](#)
- [Bernoulli](#)

```
from sklearn.naive_bayes import GaussianNB, BernoulliNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# set nan values to 0. The missing values might not affect the model
full_df = full_df.fillna(0)

# create a list of features
features = full_df.columns.tolist()
features.remove('status')
features.remove('status_bool')

# create a list of target
target = 'status'

# split the data into train and test
X_train, X_test, y_train, y_test = train_test_split(full_df[features], full_df[target], test_size=0.2, random_state=42)
```

Figura 12 – Split das Amostras

```
# create a Gaussian and Bernoulli Naive Bayes classifier
gnb = GaussianNB()
bnb = BernoulliNB()

# train the model using the training sets
gnb.fit(X_train, y_train)
bnb.fit(X_train, y_train)

# predict the response for test dataset
y_pred_gnb = gnb.predict(X_test)
y_pred_bnb = bnb.predict(X_test)
```

Figura 13 – Treino e Previsão

```
# print the first 10 anormal predictions
def print_result_sample(y_pred, y_test):
    first_10_anormal_predictions = []
    first_10_anormal_actual = []
    for i in range(len(y_pred)):
        if y_pred[i] == 'ANORMAL':
            first_10_anormal_predictions.append(y_pred[i])
            first_10_anormal_actual.append(y_test[i])
            if len(first_10_anormal_predictions) == 10:
                break
    print(first_10_anormal_predictions)
    print(first_10_anormal_actual)

print(f"\nGNB accuracy: {accuracy_score(y_test, y_pred_gnb)}")
print_result_sample(y_pred_gnb, y_test)
print(f"\nBNB accuracy: {accuracy_score(y_test, y_pred_bnb)}")
print_result_sample(y_pred_bnb, y_test)

# check how many predictions match between the two models
print(f"\nNumber of matching predictions: {np.sum(y_pred_gnb == y_pred_bnb)} / {len(y_pred_gnb)}")
```

✓ 12s

```
GNB accuracy: 0.9864969135802469
['ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL']
['NORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL']

BNB accuracy: 0.9746505083514887
['ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL']
['ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL', 'NORMAL', 'ANORMAL', 'ANORMAL', 'ANORMAL']

Number of matching predictions: 42604 / 44064
```

Figura 14 – ML Resultado

6 Simulando cenário real com Apache Kafka

Apache Kafka é uma plataforma open-source de processamento de streams desenvolvida pela Apache Software Foundation, escrita em Scala e Java. O projeto tem como objetivo fornecer uma plataforma unificada, de alta capacidade e baixa latência para tratamento de dados em tempo real. Sua camada de armazenamento é uma "fila de mensagens de publishers/subscribers escalável projetada como um log de transações distribuído", tornando-o altamente valioso para infra-estruturas corporativas que processam transmissão de dados. Pensando nisso, a ideia extra para esse projeto foi simular um ambiente em que os dados dos sensores não estão em um arquivo estático previamente fornecido, mas sim sendo gerados e consumidos em tempo real. Desse modo, o modelo de Naive Bayes é executado cada vez que novos dados são gerados, aumentando, assim, a sua precisão.

6.1 Arquitetura do Pipeline

A ideia principal é que a cada marcação de tempo (segundo, no caso) os dados do conjunto de sensores são recebidos, formatados e escritos na fila do Kafka (*producer*). Então, o *consumer* fica ouvindo essa fila e a cada nova entrada, ele lê, processa e escreve no banco de dados os novos registros. O banco ([PostgreSQL](#)) está em execução num container [Docker](#). Após os dados gravados no banco, o script com o modelo de Machine Learning é disparado e começa a consumir da tabela, gerando as análises atualizadas em tempo real.

6.2 Visão Geral do Kafka

6.2.1 Kafka Topic

Os "tópicos" do Kafka são mecanismos de armazenam a sequência de eventos recebidos. Esses tópicos podem ser replicados em diferentes partições do Broker e até mesmo serem replicados para outros nós (sistemas distribuídos) para serem processados/consumidos em paralelo. Para esse projeto foi criado apenas um tópico (*SensorDataStream*) com duas partições.

```
from kafka.admin import KafkaAdminClient, NewTopic

TOPIC="SensorsDataStream"

try:
    admin_client = KafkaAdminClient(bootstrap_servers="localhost:9092", client_id='IndustriALL')
except Exception as e:
    print("Exception while connecting Kafka")
    print(str(e))
    exit(1)

if TOPIC in admin_client.list_topics():
    print("Topic already exists")
    exit(0)

try:
    topic_list = []
    new_topic = NewTopic(name=TOPIC, num_partitions= 2, replication_factor=1)
    topic_list.append(new_topic)
    admin_client.create_topics(new_topics=topic_list)
    print("Topic created successfully")
except Exception as e:
    print("Exception while creating topic")
    print(str(e))
    exit(1)
```

Figura 15 – Creating Topic

6.2.2 Enfileirando os Dados

Os dados dos sensores são lidos do arquivo local *full.csv* criado pelo *sensor-summirize.py* referenciado nas seções anteriores. A partir do Dataframe, ele insere linha por linha na fila com o atraso de 1s, a fim de simular a condição real do problema.

```
from time import sleep
from kafka import KafkaProducer
import json
import pandas as pd

TOPIC = "SensorsDataStream"
INPUT_FILE = "data/full.csv"

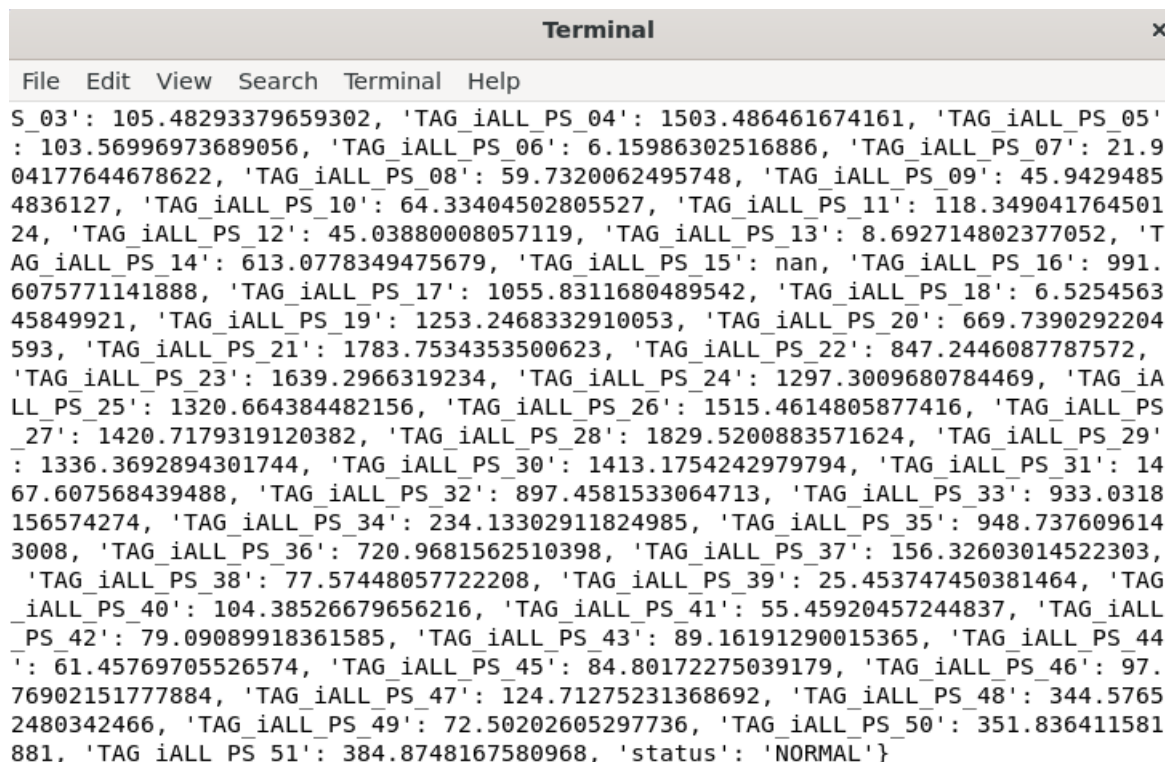
df = pd.read_csv(INPUT_FILE)

producer = KafkaProducer(
    bootstrap_servers=['localhost:9092'],
    value_serializer=lambda v: json.dumps(v).encode('utf-8'))

for idx in range(len(df)):
    print(df.iloc[idx].to_dict())
    producer.send(TOPIC, df.iloc[idx].to_dict())
    producer.flush()
    sleep(1)

producer.close()
```

Figura 16 – Producer Script



```

Terminal
File Edit View Search Terminal Help
S_03': 105.48293379659302, 'TAG_iALL_PS_04': 1503.486461674161, 'TAG_iALL_PS_05':
: 103.56996973689056, 'TAG_iALL_PS_06': 6.15986302516886, 'TAG_iALL_PS_07': 21.9
04177644678622, 'TAG_iALL_PS_08': 59.7320062495748, 'TAG_iALL_PS_09': 45.9429485
4836127, 'TAG_iALL_PS_10': 64.33404502805527, 'TAG_iALL_PS_11': 118.349041764501
24, 'TAG_iALL_PS_12': 45.03880008057119, 'TAG_iALL_PS_13': 8.692714802377052, 'T
AG_iALL_PS_14': 613.0778349475679, 'TAG_iALL_PS_15': nan, 'TAG_iALL_PS_16': 991.
6075771141888, 'TAG_iALL_PS_17': 1055.8311680489542, 'TAG_iALL_PS_18': 6.5254563
45849921, 'TAG_iALL_PS_19': 1253.2468332910053, 'TAG_iALL_PS_20': 669.7390292204
593, 'TAG_iALL_PS_21': 1783.7534353500623, 'TAG_iALL_PS_22': 847.2446087787572,
'TAG_iALL_PS_23': 1639.2966319234, 'TAG_iALL_PS_24': 1297.3009680784469, 'TAG_iA
LL_PS_25': 1320.664384482156, 'TAG_iALL_PS_26': 1515.4614805877416, 'TAG_iALL_PS
_27': 1420.7179319120382, 'TAG_iALL_PS_28': 1829.5200883571624, 'TAG_iALL_PS_29'
: 1336.3692894301744, 'TAG_iALL_PS_30': 1413.1754242979794, 'TAG_iALL_PS_31': 14
67.607568439488, 'TAG_iALL_PS_32': 897.4581533064713, 'TAG_iALL_PS_33': 933.0318
156574274, 'TAG_iALL_PS_34': 234.13302911824985, 'TAG_iALL_PS_35': 948.737609614
3008, 'TAG_iALL_PS_36': 720.9681562510398, 'TAG_iALL_PS_37': 156.32603014522303,
'TAG_iALL_PS_38': 77.57448057722208, 'TAG_iALL_PS_39': 25.453747450381464, 'TAG
_iALL_PS_40': 104.38526679656216, 'TAG_iALL_PS_41': 55.45920457244837, 'TAG_iALL
_PS_42': 79.09089918361585, 'TAG_iALL_PS_43': 89.16191290015365, 'TAG_iALL_PS_44
': 61.45769705526574, 'TAG_iALL_PS_45': 84.80172275039179, 'TAG_iALL_PS_46': 97.
76902151777884, 'TAG_iALL_PS_47': 124.71275231368692, 'TAG_iALL_PS_48': 344.5765
2480342466, 'TAG_iALL_PS_49': 72.50202605297736, 'TAG_iALL_PS_50': 351.836411581
881, 'TAG_iALL_PS_51': 384.8748167580968, 'status': 'NORMAL'}

```

Figura 17 – Producer Terminal

6.2.3 Processando a Fila

Enquanto as mensagens são enfileiradas pelo *producer*, o nosso *consumer* processa uma a uma e insere no banco de dados na medida em que lê, trata e formata as entradas.

```

from time import sleep
from kafka import KafkaConsumer
import json
import pandas as pd
import psycopg2 as pg

TOPIC="SensorsDataStream"
TABLE_NAME=TOPIC

try:
    conn = pg.connect("dbname='postgres' user='postgres' host='localhost' password='example'")
    print("Connected to database")
    cursor = conn.cursor()
except Exception as e:
    print("Exception while connecting to database")
    print(str(e))
    exit(1)

try:
    consumer = KafkaConsumer(TOPIC,
                              group_id=None,
                              bootstrap_servers=['localhost:9092'],
                              auto_offset_reset = 'earliest')
except Exception as e:
    print("Exception while connecting Kafka")
    print(str(e))
    exit(1)

print("Consumer started")
print(consumer.subscription())
check = True
cursor.execute(f"ROLLBACK;")

```

Figura 18 – Conectando à Fila e ao Banco

```

for msg in consumer:
    dictionary = json.loads(msg.value.decode('utf-8'))
    df = pd.DataFrame(dictionary, index=[0])
    print(df)

    columns = ",".join(df.columns)

    # create table if not exists
    if check:
        try:
            cursor.execute(f"CREATE TABLE {TABLE_NAME} (id SERIAL PRIMARY KEY);")
            for column in df.columns:
                if column == "timestamp":
                    cursor.execute(f"ALTER TABLE {TABLE_NAME} ADD COLUMN {column} TIMESTAMP;")
                elif column == "status":
                    cursor.execute(f"ALTER TABLE {TABLE_NAME} ADD COLUMN {column} VARCHAR;")
                else:
                    cursor.execute(f"ALTER TABLE {TABLE_NAME} ADD COLUMN {column} FLOAT;")
            conn.commit()
            check = False
        except pg.errors.DuplicateTable:
            print("Table already exists")
            check = False

    # insert data
    values = df.values.tolist()[0]
    values_str = ""
    is_timestamp = True
    for value in values:
        if is_timestamp:
            values_str+=f'"{value}",'
            is_timestamp = False
            continue
        if value == "NORMAL" or value == "ANORMAL":
            values_str+=f'"{value}",'
            break
        values_str+=str(value)+","
    values_str = values_str[:-1]
    values_str = values_str.replace("nan", "NULL")

    cursor.execute(f"INSERT INTO {TABLE_NAME}({columns}) VALUES ({values_str});", values)
    conn.commit()
    sleep(1)

cursor.close()
conn.close()

```

Figura 19 – Processando a Fila e escrevendo no Banco

Terminal					
File	Edit	View	Search	Terminal	Help
0	2018-04-01 00:20:00	3.050104	...	400.710023	NORMAL
[1 rows x 54 columns]					
	timestamp	TAG_iALL_PS_00	...	TAG_iALL_PS_51	status
0	2018-04-01 00:25:00	6.009467	...	383.953027	NORMAL
[1 rows x 54 columns]					
	timestamp	TAG_iALL_PS_00	...	TAG_iALL_PS_51	status
0	2018-04-01 00:27:00	2.013368	...	666.257182	NORMAL
[1 rows x 54 columns]					
	timestamp	TAG_iALL_PS_00	...	TAG_iALL_PS_51	status
0	2018-04-01 00:28:00	-0.87427	...	594.525997	NORMAL
[1 rows x 54 columns]					
	timestamp	TAG_iALL_PS_00	...	TAG_iALL_PS_51	status
0	2018-04-01 00:29:00	3.461133	...	657.742314	NORMAL
[1 rows x 54 columns]					
	timestamp	TAG_iALL_PS_00	...	TAG_iALL_PS_51	status
0	2018-04-01 00:30:00	6.91672	...	229.23811	NORMAL
[1 rows x 54 columns]					

Figura 20 – Consumer Terminal

6.3 Banco de Dados

O banco escolhido foi o PostgreSQL rodando no Docker. É uma implementação simples, já que estamos interessados apenas nos valores emitidos pelos sensores. Se fosse fornecido o tipo de sensor e informações extras, modelariamos o banco com as PK e FK para permitir joins e outros filtros, mas não era pertinente no contexto.

```

services:
  db:
    image: postgres
    restart: always
    environment:
      POSTGRES_PASSWORD: example
    volumes:
      - pgdata:/var/lib/postgresql/data
    ports:
      - 5432:5432
    volumes:
      pgdata:
# user: postgres
# host: localhost
# port: 5432
# pwd: example
# db: postgres

```

Figura 21 – Docker Compose Pgsqsl

#	id	timestamp	tag_ail_ps_00	tag_ail_ps_01	tag_ail_ps_02	tag_ail_ps_03	tag_ail_ps_04	tag_ail_ps_05	tag_ail_ps_06	tag_ail_ps_07	tag_ail_ps_08	tag_ail_ps_09
93	93	2018-04-01 01:31:00.000	3.0252057121	56.1409784023	108.6835224763	93.7372616011	1.295.6620761135	75.2307978293	26.4968999056	36.8189435555	38.356635131	17.5297496588
94	94	2018-04-01 01:33:00.000	5.0208757939	59.5634779848	75.3622583963	32.337486835	842.6515194271	133.6080388679	31.1880917758	30.3699468933	54.7586004925	41.0185634083
95	95	2018-04-01 01:35:00.000	4.2171450275	5.8194111799	83.941072112	85.8390998634	1.136.1624560074	149.7886200006	34.7688630467	31.0466407104	34.2811268165	21.8190974891
96	96	2018-04-01 01:37:00.000	6.5756074188	111.9914626976	99.3230684024	90.5433683004	1.081.5248876355	114.2816341298	45.4758761086	64.202993596	21.9170128389	15.8343607281
97	97	2018-04-01 01:39:00.000	1.8570044879	3.7822791008	110.4372120137	123.694423227	593.968806803	287.1981179429	23.794715362	12.5169777844	10.214626689	46.7861228048
98	98	2018-04-01 01:36:00.000	2.7127214443	88.732385002	189.3710625858	60.992672489	1.308.9691608862	121.1789755618	39.2812625895	20.897165587	28.9433141365	48.1426535792
99	99	2018-04-01 01:38:00.000	6.614705977	147.0936164358	71.785063745	64.1066135114	470.4902208419	188.9519298712	23.3145404974	52.114773761	57.8703337823	24.9152014406
100	100	2018-04-01 01:40:00.000	4.2124190349	13.834036736	153.9285204469	73.3211458919	1.520.6847813992	236.5882190947	14.6727044868	21.6971886104	2.1957581746	26.0715189119
101	101	2018-04-01 01:41:00.000	3.3655528953	193.2138795652	62.8719436781	101.9763806184	1.005.1448700225	160.0694008479	25.179551763	47.7667440312	55.8749170623	13.7710786048
102	102	2018-04-01 01:39:00.000	2.4136414042	64.2622986908	158.3612928758	85.3192014136	737.466785573	221.9977359401	16.1982904566	43.99174233	21.9735580056	25.1159920774
103	103	2018-04-01 01:44:00.000	7.7493374016	5.7690958504	109.6741773815	133.968146867	1.505.0457651357	214.3366297745	27.757524941	34.5394671033	49.2282118435	48.4031888552
104	104	2018-04-01 01:42:00.000	3.8040125738	134.3833520083	115.9327238826	58.6275997857	1.965.8761852588	140.7520771892	21.533767686	6.304380779	33.1107655562	22.1213457011
105	105	2018-04-01 01:43:00.000	1.4563793568	133.6007544352	-4.8408971776	97.7505631694	589.4179634592	206.1191858926	29.497336445	27.295086139	22.104965747	15.3603935261
106	106	2018-04-01 01:45:00.000	10.579961044	28.7474493399	116.4269452797	104.844090305	592.737907047	166.0121437768	28.7953004489	26.546994237	37.3586474941	12.9140546434
107	107	2018-04-01 01:47:00.000	5.749633574	83.660775984	175.4314774732	89.2782653222	835.1093606233	104.422070101	25.8623482881	46.2879661882	-1.9766939731	10.6437142668
108	108	2018-04-01 01:49:00.000	2.9805601543	95.0489893555	81.0441927511	117.5459286153	1.233.2270969195	152.6047823169	53.0904464355	28.033143728	40.9120684996	32.203220792
109	109	2018-04-01 01:46:00.000	5.3201159463	151.0492128127	64.5035264644	111.7292500626	1.109.0151109819	97.7126889757	-1.8860818136	23.586538753	25.6853752886	46.3154057225
110	110	2018-04-01 01:48:00.000	4.4527760091	33.0422493976	101.2603931632	141.5026530891	1.380.9196813976	85.0321795125	18.1262909614	11.1773578154	33.3824777835	12.9424837022
111	111	2018-04-01 01:50:00.000	6.3342167527	83.0986427181	111.5888492878	68.8582690926	735.7672975556	257.5089178418	30.3633398142	26.667444213	33.234675508	10.4917428783
112	112	2018-04-01 01:51:00.000	2.0962776062	43.2378685985	71.5940993764	90.3939990309	668.39902682	45.8761176226	26.2521550652	36.7756144434	22.8039581929	17.5744397277
113	113	2018-04-01 01:52:00.000	2.6316512216	85.0110539187	134.333987347	62.8700621327	537.4567626277	315.8116091077	16.6816853405	33.2192429516	43.0414136169	64.6720409289
114	114	2018-04-01 01:53:00.000	7.6971711468	161.9832809811	192.032673737	45.9019218172	975.4732114603	176.326602216	30.7682088356	64.5193469852	20.6545004236	2.1077786872
115	115	2018-04-01 01:57:00.000	3.5729115308	97.643431261	145.7570698519	92.5274723913	303.3207685853	166.5186354761	44.8256158652	30.2802364083	36.9756223356	52.042658735
116	116	2018-04-01 01:54:00.000	4.7523561965	42.6012830298	157.8416391899	73.7814984086	762.4870154217	127.4959736155	37.1627213083	15.0555834934	44.8210268062	16.173228723
117	117	2018-04-01 01:55:00.000	7.137111335	200.461816649	106.0553666758	113.9834830486	936.4031735155	150.2688786135	37.6308713928	35.3569890067	35.002704258	3.6203688616
118	118	2018-04-01 01:56:00.000	0.75843127442	109.1142326807	181.9147803054	59.4792524317	1.374.1707588742	334.4779628068	39.6377532175	9.0218023927	59.899696217	29.776666246
119	119	2018-04-01 01:58:00.000	1.5807190175	205.4908064148	102.7406355166	74.6113154523	2.018.4008932442	148.2853780498	42.5759542555	-5.372543076	68.830417069	33.382664625
120	120	2018-04-01 02:01:00.000	0.9906876667	184.7040868965	-73.5796245803	37.1424134263	2.447.2107488634	111.4929232037	21.9573759148	22.4185825563	33.1529914799	20.7289141054
121	121	2018-04-01 01:59:00.000	5.3729405933	128.088048529	120.9134766449	74.7322929886	1.533.4857296684	162.6362846706	39.4944138133	34.0445778521	36.2682946985	17.2068040096
122	122	2018-04-01 02:00:00.000	6.8991116084	-35.0155383147	89.9106978899	35.3203210934	749.5018587391	101.7783249266	28.4119902178	20.9315254397	61.4845489928	28.0949206235
123	123	2018-04-01 02:05:00.000	4.0412581009	80.6814925077	79.5509465399	49.2348630326	17.1504431965	71.8414002669	-5.4831670219	46.960910379	12.7945076627	42.3761386215
124	124	2018-04-01 02:02:00.000	4.6607391717	49.3096718864	64.854987629	76.565545861	2.016.3586817412	151.889453991	45.7840468071	68.6556488684	43.744255233	24.4903673666
125	125	2018-04-01 02:03:00.000	4.6740815699	78.0053574713	45.8411482917	89.8380662941	1.329.8873454329	159.5700096528	15.225235257	25.83953246	51.111048612	26.0281331084

Figura 22 – Registros sendo Escritos

6.4 O Algoritmo em Tempo Real

Já que os valores estão sendo gerados em tempo real, o modelo de Naive Bayes é executado cada vez que novos dados são inseridos no banco, aumentando, assim, a sua precisão.

OBS: as imagens do resultado do algoritmo estão com valores de 1.0 de precisão e apenas com status 'NORMAL' pois representam os primeiros registros, apenas mais para o final (linha aprox. 170k) que começam a aparecer o status 'ANORMAL'

Timestamp	ID	Status	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8	Col 9	Col 10	Col 11	Col 12	Col 13	Col 14
2018-04-01 00:00:00	1	NORMAL	4.548754	90.886874	58.698896	89.301134	...	119.246973	60.729332	310.022461	124.735196	426.651658	410.82048	
2018-04-01 00:01:00	2	NORMAL	7.887998	56.555373	80.802810	120.898222	...	63.457947	62.910653	306.084796	158.822485	375.316113	143.62072	
2018-04-01 00:02:00	3	NORMAL	4.975919	182.086958	87.273632	9.914782	...	60.560834	81.332910	353.863854	88.772027	444.809188	618.89800	
2018-04-01 00:03:00	4	NORMAL	6.304142	58.417235	75.059520	64.167463	...	24.141894	36.219671	301.563110	53.387484	414.052496	427.32337	
2018-04-01 00:04:00	5	NORMAL	1.671733	108.946809	94.910470	14.551922	...	53.913605	116.770552	298.957820	168.746952	431.548430	514.65988	
...
2018-04-01 01:20:00	80	NORMAL	4.407174	177.266716	6.622025	127.420173	...	33.617811	51.190581	291.098700	216.650123	424.343677	247.03801	
2018-04-01 01:21:00	82	NORMAL	9.207485	127.108040	135.255643	101.934049	...	71.230622	110.163526	224.818371	39.509751	356.072623	583.95355	
2018-04-01 01:23:00	83	NORMAL	0.027884	52.325026	144.217406	-0.638060	...	67.568270	37.575692	404.095635	145.426873	414.846741	455.32829	
2018-04-01 01:24:00	84	NORMAL	5.365662	48.190821	97.781580	155.578305	...	5.077884	180.074651	357.636277	138.126290	388.515858	540.58459	
2018-04-01 01:25:00	86	NORMAL	3.698614	29.892224	98.595536	110.169836	...	115.970997	86.703624	378.201432	94.764088	509.148873	418.61916	

[69 rows x 53 columns]

GNB accuracy: 1.0
BNB accuracy: 1.0
Number of matching predictions: 14 / 14
Next prediction status GNB: ['NORMAL']
Next prediction status BNB: ['NORMAL']
Number of ANORMAL status find until now: 0

Figura 23 – Result

6.5 Executando em sua Máquina

Todos os resultados acima foram utilizando o SO Windows11 com WSL. Apenas o Docker estava em execução no próprio Windows, os demais serviços rodaram no WSL. Entre na pasta do projeto (IndustriALL-AIChallenge) e siga as próximas instruções.

OBS: é necessário criar uma pasta 'data' contendo o full.csv ou ter, nessa pasta, os arquivos fonte para rodar o summarize.py e construir o full.csv, ver Figura 2

6.5.1 Criando o Virtual Environment e Instalando Dependências

- > wsl
- > python3 -m venv venv
- > source venv/bin/activate
- > pip install -r requirements.txt

6.5.2 Instalando Dependências no WSL

- > sudo apt update
- > sudo apt upgrade
- > sudo apt install dos2unix
- > sudo apt install gnome-terminal

6.5.3 Executando o setup.sh

- > `chmod +x kafka/setup.sh`
- > `chmod +x setup.sh`
- > `./setup.sh`
- > *(caso precise converter algum script use: > dos2unix <script.sh>)*

Você deverá ver 4 terminais extras para o Kafka:

1. Zookeeper
2. Server
3. Producer
4. Consumer

E no próprio terminal a execução do Naive Bayes

```

Server
File Edit View Search Terminal Help
message.downconversion.enable -> true, segment.jitter.ms -> 0,
cleanup.policy -> [delete], flush.ms -> 9223372036854775807, re
tention.ms -> 60480000, segment.bytes -> 1073741824, flush.mes
sages -> 9223372036854775807, message.format.version -> 2.8-IV1
, max.compaction.lag.ms -> 9223372036854775807, file.delete.del
ay.ms -> 60000, max.message.bytes -> 1048500, min.compaction.la
g.ms -> 0, message.timestamp.type -> CreateTime, preallocate ->
false, index.interval.bytes -> 4096, min.cleanable.dirty.ratio
-> 0.5, unclean.leader.election.enable -> false, retention.by
tes -> -1, delete.retention.ms -> 86400000, segment.ms -> 604800
000, message.timestamp.difference.max.ms -> 9223372036854775807
, segment.index.bytes -> 10485760, (kafka.log.LogManager)
[2023-11-29 18:40:48,249] INFO [Partition SensorsDataStream-0 b
roker=0] No checkpointed highwatermark is found for partition S
ensorsDataStream-0 (kafka.cluster.Partition)
[2023-11-29 18:40:48,250] INFO [Partition SensorsDataStream-0 b
roker=0] Log loaded for partition SensorsDataStream-0 with init
ial high watermark 0 (kafka.cluster.Partition)

Zookeeper
File Edit View Search Terminal Help
[2023-11-29 18:40:27,820] INFO zookeeper.snapshotSizeFacto
r = 0.33 (org.apache.zookeeper.server.ZKDatabase)
[2023-11-29 18:40:27,828] INFO Snapshotting: 0x0 to /tmp/z
ookeeper/version-2/snapshot.0 (org.apache.zookeeper.server
.persistence.FileTxnSnapLog)
[2023-11-29 18:40:27,831] INFO Snapshotting: 0x0 to /tmp/z
ookeeper/version-2/snapshot.0 (org.apache.zookeeper.server
.persistence.FileTxnSnapLog)
[2023-11-29 18:40:27,853] INFO PrepRequestProcessor (sid:0
) started, reconfigEnabled=false (org.apache.zookeeper.ser
ver.PreRequestProcessor)
[2023-11-29 18:40:27,861] INFO Using checkIntervalMs=60000
maxPerMinute=10000 (org.apache.zookeeper.server.Container
Manager)
[2023-11-29 18:40:38,471] INFO Creating new log file: log.
1 (org.apache.zookeeper.server.persistence.FileTxnLog)

Consumer
File Edit View Search Terminal Help
1 rows x 24 columns] ... status
timestamp ... NORMAL
2018-04-01 03:00:00 ... NORMAL

1 rows x 54 columns] ... status
timestamp ... NORMAL
2018-04-01 03:01:00 ... NORMAL

1 rows x 54 columns] ... status
timestamp ... NORMAL
2018-04-01 03:03:00 ... NORMAL

1 rows x 54 columns] ... status
timestamp ... NORMAL
2018-04-01 03:05:00 ... NORMAL

1 rows x 54 columns]

Producer
File Edit View Search Terminal Help
[2023-11-29 18:40:27,820] INFO zookeeper.snapshotSizeFacto
r = 0.33 (org.apache.zookeeper.server.ZKDatabase)
[2023-11-29 18:40:27,828] INFO Snapshotting: 0x0 to /tmp/z
ookeeper/version-2/snapshot.0 (org.apache.zookeeper.server
.persistence.FileTxnSnapLog)
[2023-11-29 18:40:27,831] INFO Snapshotting: 0x0 to /tmp/z
ookeeper/version-2/snapshot.0 (org.apache.zookeeper.server
.persistence.FileTxnSnapLog)
[2023-11-29 18:40:27,853] INFO PrepRequestProcessor (sid:0
) started, reconfigEnabled=false (org.apache.zookeeper.ser
ver.PreRequestProcessor)
[2023-11-29 18:40:27,861] INFO Using checkIntervalMs=60000
maxPerMinute=10000 (org.apache.zookeeper.server.Container
Manager)
[2023-11-29 18:40:38,471] INFO Creating new log file: log.
1 (org.apache.zookeeper.server.persistence.FileTxnLog)

timestamp ... 410.820484
2018-04-01 00:00:00 1 ... 410.820484
2018-04-01 00:00:00 126 ... 410.820484
2018-04-01 00:01:00 127 ... 143.620728
2018-04-01 00:01:00 2 ... 143.620728
2018-04-01 00:02:00 3 ... 618.898000
...
2018-04-01 02:57:00 302 ... 641.466579
2018-04-01 02:58:00 303 ... 657.250467
2018-04-01 02:59:00 305 ... 639.824957
2018-04-01 03:02:00 306 ... 206.853022
2018-04-01 03:04:00 307 ... 406.205254

[163 rows x 53 columns]

GNB accuracy: 1.0
BNB accuracy: 1.0

Number of matching predictions: 33 / 33

Next prediction status GNB: ['NORMAL']

Next prediction status BNB: ['NORMAL']
Number of ANORMAL status find until now: 0
=====
No new data
No new data
No new data

```

Figura 24 – Terminals