

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 foi utilizado técnicas de estatística descritiva para entender a relação e contexto dos dados. E, por fim, 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.

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

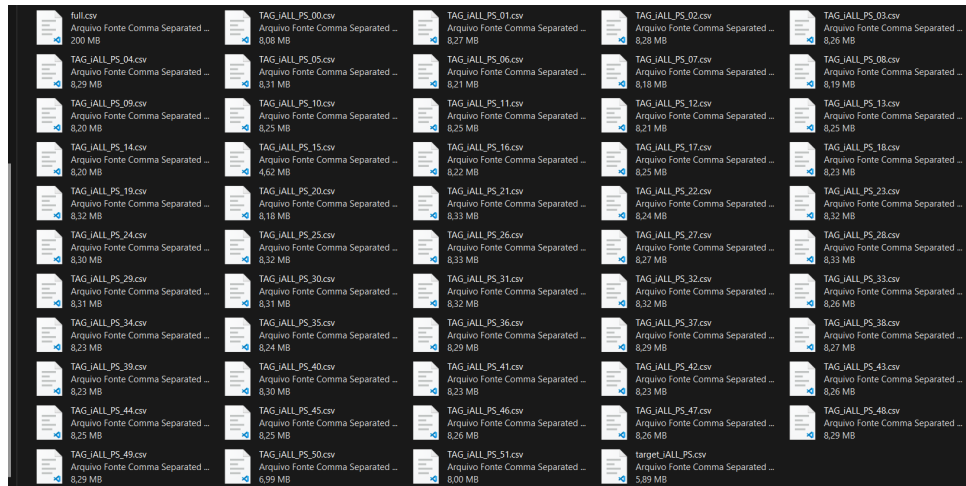


Figura 2 – Arquivos

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 parallelpy > ...
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 (e até mesmo 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.

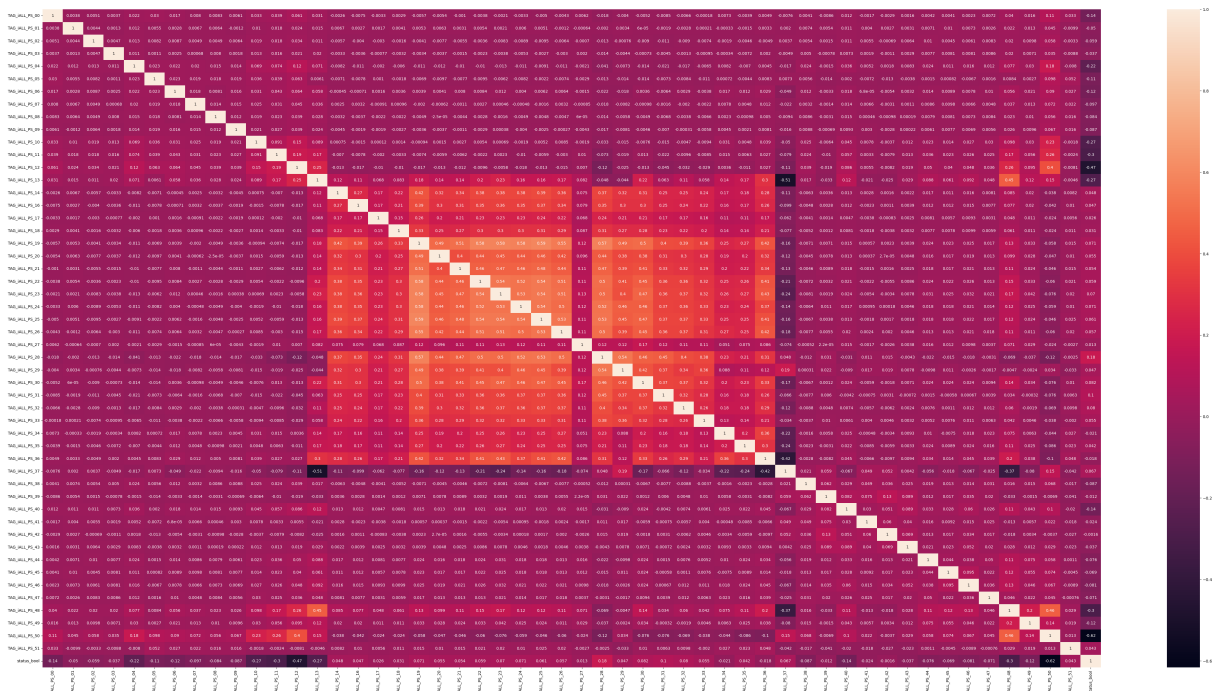


Figura 10 – Correlação entre Sensores

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.

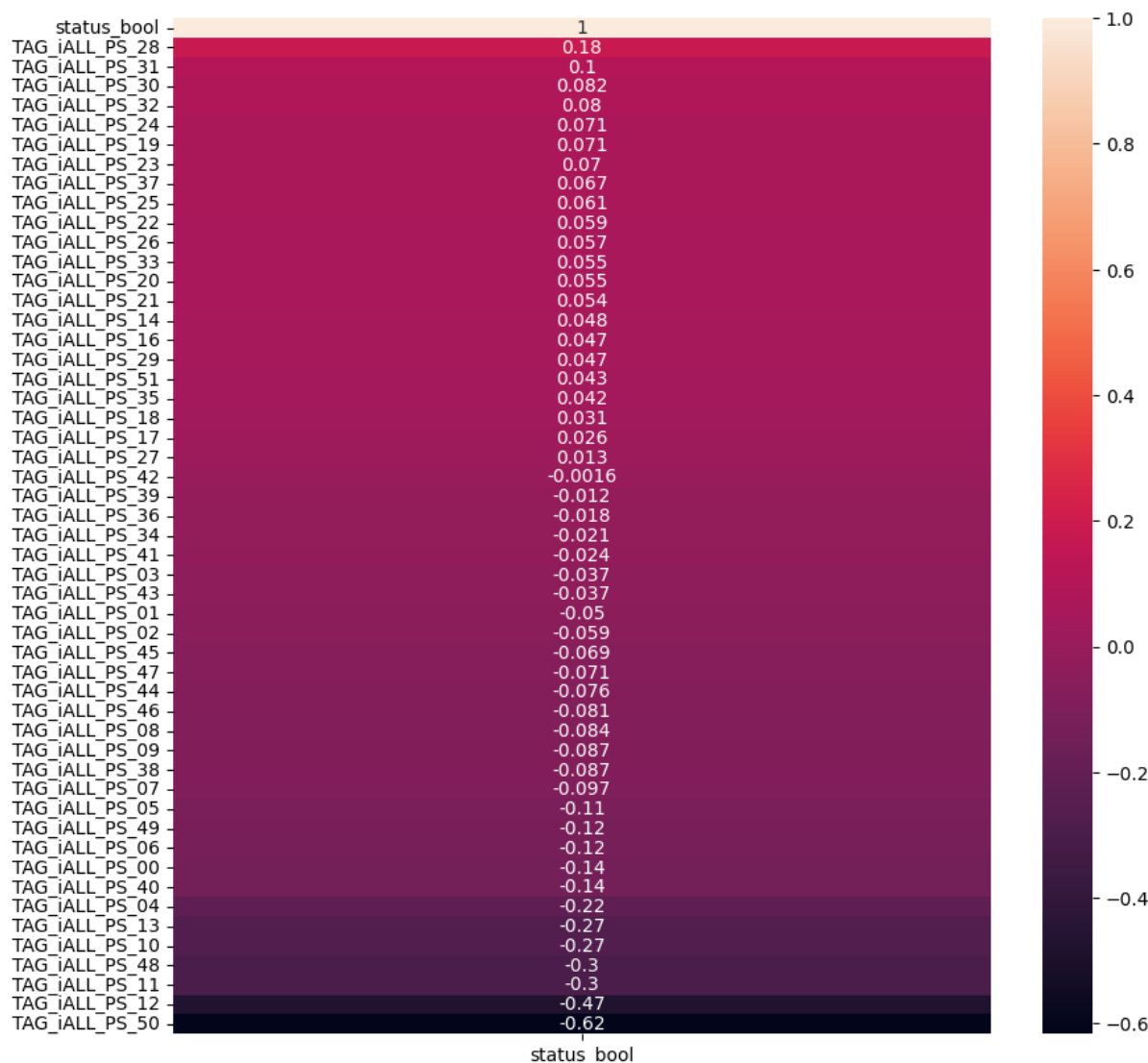


Figura 11 – Correlação entre Sensores e Status

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

```

1  import numpy as np
2
3  class NaiveBayes:
4      def fit(self, X, y):
5          """Training the model
6
7          Args:
8              X: training samples
9              y: training labels
10         """
11         n_samples, n_features = X.shape
12         self._classes = np.unique(y)
13         n_classes = len(self._classes)
14
15         # calculate mean, var and prior for each class
16         self._mean = np.zeros((n_classes, n_features), dtype=np.float64)
17         self._var = np.zeros((n_classes, n_features), dtype=np.float64)
18         self._priors = np.zeros(n_classes, dtype=np.float64) # frequency of each class
19
20         for idx, c in enumerate(self._classes):
21             X_c = X[y == c]
22             self._mean[idx, :] = X_c.mean(axis=0)
23             self._var[idx, :] = X_c.var(axis=0)
24             self._priors[idx] = X_c.shape[0] / float(n_samples)
25
26     def predict(self, X):
27         """_summary_
28
29         Args:
30             X: samples
31         """
32         y_pred = [self._predict(x) for x in X]
33         return np.array(y_pred)
34
35     def _predict(self, x):
36         posteriors = []
37
38         # calculate posterior probability for each class
39         for idx, c in enumerate(self._classes):
40             prior = np.log(self._priors[idx])
41             posterior = np.sum(np.log(self._probability_density_function(idx, x)))
42             posterior = posterior + prior
43             posteriors.append(posterior)
44
45         return self._classes[np.argmax(posteriors)]
46
47     def _probability_density_function(self, class_idx, x):
48         mean = self._mean[class_idx]
49         var = self._var[class_idx]
50         numerator = np.exp(-((x - mean) ** 2)/(2*var))
51         denominator = np.sqrt(2 * np.pi * var)
52         return numerator / denominator

```

Figura 15 – Implementação Naive Bayes