Análise Preditiva da Saúde Cardíaca de Cidadãos Brasileiros

Bernardo Vieira
Instituto de Ciências Exatas e Informática
Sabará, Minas Gerais, Brasil
baavieira@sga.pucminas.br

Gabriel Jota Lizardo
Instituto de Ciências Exatas e Informática
Nova Lima, Minas Gerais, Brasil
gabriel.jota@sga.pucminas.br

Guilherme Oliveira de Rodrigues
Instituto de Ciências Exatas e Informática
Nova Lima, Minas Gerais, Brasil
guilherme.rodrigues.1449815@sga.pucminas.br

Henrique Oliveira da C. Franco
Instituto de Ciências Exatas e Informática
Belo Horizonte, Minas Gerais, Brasil
henrique.franco@sga.pucminas.br

Larissa Mariella
Instituto de Ciências Exatas e Informática
Matozinhos, Minas Gerais, Brasil
larissa.mariella@sga.pucminas.br

Victor Hugo
Instituto de Ciências Exatas e Informática
Belo Horizonte, Minas Gerais, Brasil
cpalmer@sga.pucminas.br

ABSTRACT

Esse projeto visa alcançar um entendimento melhor acerca de atividades, hábitos ou características que podem influenciar na probabilidade de uma pessoa ter (ou não) doenças cardíacas. Inicialmente foi aplicado um pré-processamento na base de dados, como seleção de colunas relevantes, balanceamento de dados e tratamento de dados ausentes. A partir disso foi aplicado uma árvore de decisão como modelo de previsão. Os resultados geraram um relatório que busca entender melhor quais circunstâncias estão mais relacionadas com problemas cardíacos.

KEYWORDS

complicações cardíacas, saúde, análise de dados, inteligência artificial, árvore de decisão, pesquisa nacional de saúde, medicina preventiva, predição

ACM Reference Format:
Bernardo Vieira, Gabriel Jota Lizardo, Guilherme Oliveira de Rodrigues, Henrique Oliveira da C. Franco, Larissa Mariella, and Victor Hugo. 2024. Análise Preditiva da Saúde Cardíaca de Cidadãos Brasileiros. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/1122445.1122456

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUÇÃO

A saúde do coração é um dos pilares fundamentais para o bem-estar e a longevidade humana. Doenças cardiovasculares são uma das principais causas de morte no mundo, representando um grave problema de saúde pública. Assim, prever condições cardíacas antes que se tornem críticas é uma abordagem preventiva de grande valor.

Neste contexto, a utilização de inteligência artificial (IA) surge como uma ferramenta promissora na área médica. Com o objetivo de contribuir para a detecção precoce de problemas cardíacos, este projeto explora a criação de um modelo preditivo utilizando dados da Pesquisa Nacional de Saúde (PNS). A PNS é uma base de dados abrangente que coleta informações sobre diversos aspectos da saúde dos brasileiros, incluindo fatores de risco para doenças cardíacas, como hábitos de vida, condições de saúde preexistentes e características demográficas [1].

O projeto busca desenvolver uma IA capaz de analisar esses dados e identificar padrões associados à presença de problemas cardíacos. Utilizando técnicas de aprendizado de máquina, a IA será treinada para reconhecer sinais sutis que podem escapar à observação humana, oferecendo uma ferramenta potencialmente poderosa para apoiar diagnósticos clínicos.

A importância deste trabalho reside não apenas no avanço tecnológico, mas também na possibilidade de oferecer uma solução preventiva de baixo custo e alta precisão. A previsão de condições cardíacas com antecedência pode reduzir significativamente a mortalidade, melhorar a qualidade de vida dos pacientes e aliviar a pressão sobre o sistema de saúde. Ao final, este estudo propõe uma aplicação prática de IA que pode revolucionar o campo da medicina preventiva, colocando em evidência o papel da ciência de dados na saúde pública.

2 MATERIAIS E MÉTODOS

2.1 Descrição da base de dados

A imagem a seguir ilustra os grupos de atributos que são relevantes e consequentemente foram utilizados para a classificação das entidades com base na classe escolhida.

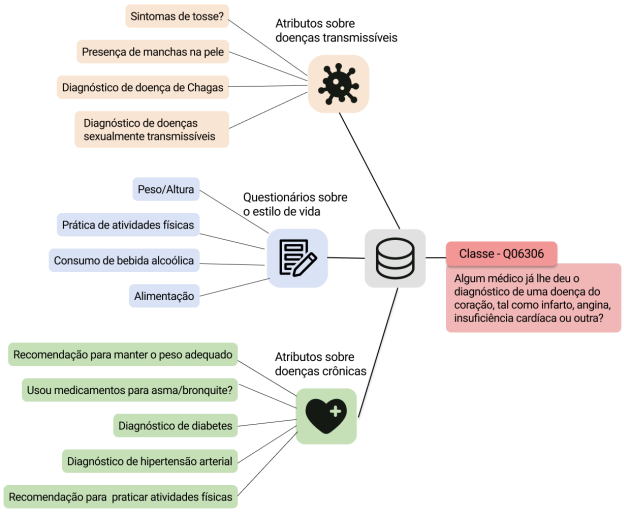

Figure 1: Fluxo de Pré-processamento dos Dados

2.2 Pré-processamento

2.2.1 Filtro de instâncias. Primeiramente foi aplicado um filtro na base de dados, mantendo somente as instâncias onde a variável alvo, chamada na base de "Q06306", é diferente de nula. Já diminuindo o tamanho da base em cerca de 61%, partindo de 293727 para 90847 instâncias. Após isso, foram retiradas as colunas que possuíam mais de 70% de valores nulos, sendo estas inúteis para o trabalho, como segue o código:

```
initial_rows, initial_columns = df.shape
print(f"Numero Inicial de Instancias: {initial_rows}")
print(f"Numero Inicial de Colunas {initial_columns}")

missing_percentage = df.isnull().mean() * 100

columns_to_drop =
missing_percentage[missing_percentage > 70].index
df_cleaned = df.drop(columns=columns_to_drop)

after_column_removal_rows,
after_column_removal_columns
= df_cleaned.shape
print(f"Numero de Colunas Restantes
apos remover colunas com >70% de Null:
{after_column_removal_columns}")

threshold = len(df_cleaned.columns) * 0.5
df_cleaned = df_cleaned.dropna(thresh=threshold)

final_rows, final_columns = df_cleaned.shape
print(f"Numero de Instancias apos
retirar instancias com mais
de 50% de campos nulos: {final_rows}")

df = df_cleaned
```

2.2.2 Codificação numérica-simbólica. Como a maior parte das colunas já estava em uma codificação ideal, foi necessário o uso do "Label Encoder" somente para a variável alvo, transformando os valores de 2.0 = não e 1.0 = sim, para 0 = não e 1 = sim. Como demonstrado no código abaixo:

```
from sklearn.preprocessing import LabelEncoder

# Instanciando o LabelEncoder
le = LabelEncoder()

# Aplicar a transformação na coluna 'Q06306'
df['Q06306'] = le.fit_transform(df['Q06306'])

print(df['Q06306'].value_counts())
```

2.2.3 Balanceamento de dados. Após todo esse processo, foi percebido que havia um grande desbalanceamento entre a classe, sendo 86114 "nãos" e 4732 "sims", sendo necessário um balanceamento. Foi utilizada a tecnologia de balanceamento do "Undersampling" onde são retirados dados da classe majoritária para balancear com a classe minoritária. Foi utilizado um modelo para "Undersampling", o "RandomUnderSampler", mantendo o conjunto majoritário 20% maior que o conjunto majoritário. Sendo necessário lembrar também que as instâncias retiradas foram introduzidas no conjunto de teste. Código abaixo:

```
from sklearn.model_selection import train_test_split

# Dividir os dados em conjunto de treino e teste

X_train, X_test, y_train, y_test =
train_test_split(df.drop(columns='Q06306'),
df['Q06306'], test_size=0.3, random_state=42)

from imblearn.under_sampling import RandomUnderSampler

import numpy as np

rus = RandomUnderSampler(sampling_strategy=0.8,
random_state=42)
X_resampled, y_resampled =
```

```
rus.fit_resample(X_train, y_train)
print(y_resampled.value_counts())

#Obter os índices que foram mantidos e removidos
mask_kept = rus.sample_indices_

# Índices das instâncias mantidas no conjunto de treino
mask_removed = np.setdiff1d(np.arange(len(X_train)),
mask_kept)

# Índices das instâncias removidas

# Capturar as instâncias removidas
X_removed = X_train.iloc[mask_removed]
y_removed = y_train.iloc[mask_removed]

# Verificar quantas instâncias foram removidas
print(f"Número de instâncias removidas:
{len(mask_removed)}")

X_test = pd.concat([X_test, X_removed],
ignore_index=True)
y_test = pd.concat([y_test, y_removed],
ignore_index=True)
```

Após o "Undersampling", foi feito um "Oversampling" para aumentar um pouco o conjunto minoritário. Sendo considerado um balanceamento misto. O código do "Oversampling" está abaixo:

```
from imblearn.over_sampling import SMOTE

 #Aplicar SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled =
smote.fit_resample(X_resampled, y_resampled)

# Exibir o número de ocorrências após o balanceamento
print(y_resampled.value_counts())
```

2.2.4 Tratamento de dados ausentes. Por conta do tamanho da base e o grande número de instâncias foi utilizado um "Imputer", mais especificamente o "KNNImputer"[2] ou seja, um modelo para imputar valores nos dados ausentes. Foi percebido uma demora muito grande para imputar todos os valores, então foi utilizada uma função que separa os conjuntos em blocos, imputa cada bloco, e os concatena. A função está mais detalhada na seção 4. O chamar da função está detalhado no código abaixo:

```
# Aplicando o KNNImputer com barra de progresso
X_resampled = impute_with_progress(X_resampled, imputer)

# Aplicando o KNNImputer com barra de progresso
X_test = impute_with_progress(X_test, imputer)
```

2.3 Descrição dos Métodos Utilizados

2.3.1 Algoritmos de Aprendizado de Máquina. Para a etapa de aprendizado, utilizamos uma Árvore de Decisão (*Decision Tree Classifier*), implementada por meio da biblioteca `scikit-learn`. O algoritmo foi configurado com os seguintes hiperparâmetros:

- **random_state**: 42
- **class_weight**: `balanced`, para lidar com o desbalanceamento inicial de classes.
- **max_depth**: 3, limitando a profundidade máxima da árvore para evitar overfitting.

Os hiperparâmetros foram ajustados com base na performance do modelo sobre os dados de treino, priorizando a melhor acurácia possível sem perder a interpretabilidade da árvore. A profundidade máxima de 3 foi escolhida para manter a simplicidade do modelo, facilitando a análise visual e evitando overfitting nos dados.

Além disso, utilizamos técnicas de balanceamento de classes com `SMOTE` (*Synthetic Minority Over-sampling Technique*) para criar instâncias sintéticas da classe minoritária. Isso garantiu um balanceamento adequado após a subamostragem (`RandomUnderSampler` com *sampling_strategy*=0.8) para evitar vieses no modelo.

2.3.2 Ambiente de Desenvolvimento. Todo o código foi desenvolvido na versão 3.9 do Python, utilizando o ambiente `Jupyter Notebook` para facilitar a execução e visualização dos resultados. As principais bibliotecas utilizadas foram:

- `pandas` para manipulação de dados.
- `scikit-learn` para a construção e avaliação do modelo.
- `imblearn` para o balanceamento das classes.
- `matplotlib` e `seaborn` para a visualização de gráficos e resultados.
- `tqdm` para exibir o progresso da imputação de valores nulos.

2.3.3 Etapas de pré-processamento. As etapas de pré-processamento foram essenciais para a preparação da base de dados para o algoritmo de aprendizado de máquina. A Figura 2 apresenta o fluxograma com as principais etapas executadas:

- **Remoção de colunas redundantes**:
  - `P00102` - O(A) Sr(a) sabe seu peso?
  - `P00103` - Peso - Informado (em kg)
  - `P00402` - O(A) Sr(a) sabe sua altura? (mesmo que seja valor aproximado)
  - `P00403` - Altura - Informada (em cm)
  - `UPA_PNS` - UPA

  - VDDATA - Data de geração do arquivo de microdados. Data ordenada na forma: ano (4 algarismos), mês (2) e dia (2) - AAAAMMDD
  - V0006_PNS - Número de ordem do domicílio na PNS foram removidas
- **Remoção de colunas com muitos valores nulos**: Foram removidas colunas com mais de 70% de valores nulos.
- **Imputação de valores nulos**: Usamos o algoritmo KNNImputer com n_neighbors=6 para preencher os valores ausentes.
- **Balanceamento de classes**: Inicialmente aplicamos RandomUnderSampler para reduzir o número de instâncias da classe majoritária e, em seguida, utilizamos o SMOTE para balancear a base de dados.

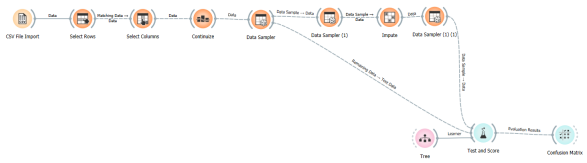

Figure 2: Fluxo de Pré-processamento dos Dados

2.4 Métricas de avaliação de qualidade

Para avaliar os resultados foram utilizadas como métricas de avaliação a precisão, o "recall", ou sensibilidade, e o "f1-score".

3 RESULTADOS E DISCUSSÕES

Os resultados não foram os ideais, o modelo apresentou uma grande dificuldade em identificar instâncias com o valor "sim", tendo uma precisão muito baixa nessa classe, enquanto isso, o modelo apresentou uma precisão altíssima na classe "não", demonstrando talvez um "Overfitting". Abaixo se encontra a matriz de confusão, na imagem 3 e a árvore de decisão, na imagem 5

O nó raiz se refere ao atributo "Q00201", que diz se: "Algum médico já lhe deu o diagnóstico de hipertensão arterial (pressão alta)?"

4 UTILIZAÇÃO DO CHATGPT

A ferramenta ChatGPT foi utilizada no contexto de geração do código referente ao algoritmo do KNNImputer da biblioteca scikit-learn.

```
from sklearn.impute import KNNImputer
from tqdm import tqdm
import numpy as np

# Definindo o KNNImputer
```

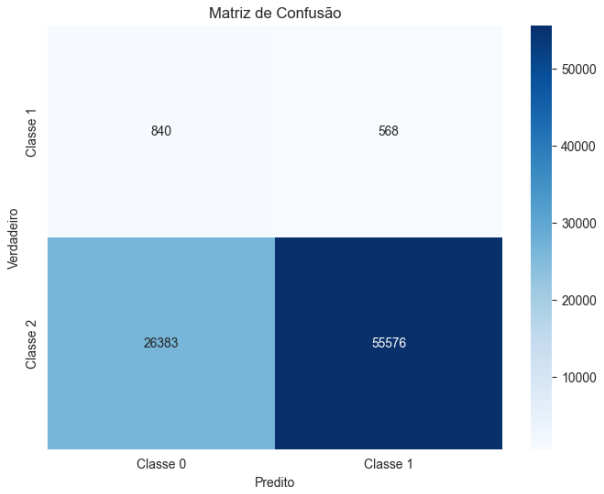

Figure 3: Matriz de confusão

```
Acurácia: 0.7179939304520974

Relatório de classificação:
              precision    recall  f1-score   support

           0       0.03      0.56      0.06      1408
           1       0.99      0.72      0.83     81959

    accuracy                           0.72     83367
   macro avg       0.51      0.64      0.45     83367
weighted avg       0.97      0.72      0.82     83367
```

Figure 4: Relatório de classificação

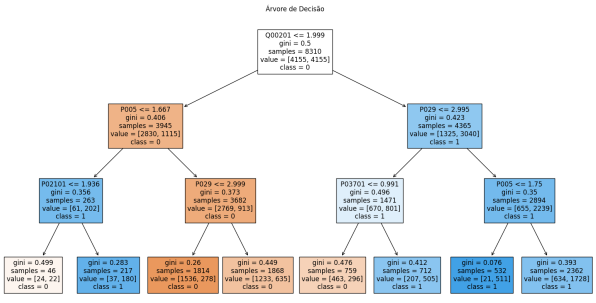

Figure 5: Árvore de decisão formada

```
imputer = KNNImputer(n_neighbors=6)
# Função personalizada para fazer a imputação
em blocos e acompanhar o progresso
def impute_with_progress(df, imputer,
    chunk_size=1000):
    # Calcula o número de blocos
```

```
num_chunks = int(np.ceil(
(df.shape[0] / chunk_size)
)

# Lista para armazenar blocos processados
imputed_chunks = []
num_cols = df.shape[1]
# Número de colunas esperado

# Barra de progresso usando tqdm
for i in tqdm(range(num_chunks),
      desc="Imputando valores"):
    # Determina o índice de início
      e fim do bloco
    start = i * chunk_size
  end = min((i + 1) * chunk_size, df.shape[0])

    # Seleciona o bloco
    df_chunk = df.iloc[start:end]

    # Aplica o imputador no bloco
    imputed_chunk =
    imputer.fit_transform(df_chunk)

 # Verifica se o número de colunas está correto
    if imputed_chunk.shape[1] == num_cols:
        imputed_chunks.append(imputed_chunk)
    else:
      print(f"Erro: O bloco {i} tem um número
        diferente de colunas
        ({imputed_chunk.shape[1]})
        do que o esperado ({num_cols})")
        return None

# Junta todos os blocos imputados
return np.vstack(imputed_chunks)
```

5 CÓDIGO DESENVOLVIDO

Inicialmente, foi desenvolvido um algoritmo de árvore de decisão utilizando uma estrutura existente no Python. Esse algoritmo constrói uma árvore onde cada nó interno representa uma decisão baseada em atributos dos dados, e as folhas indicam a classificação ou previsão final, permitindo a análise de novos dados com base nessa estrutura hierárquica. O código desenvolvido se encontra no seguinte link do notebook do Deepnote: Código_desenvolvido

REFERENCES

[1] PNS. 2019. *BASES DE DADOS*. https://www.pns.icict.fiocruz.br/bases-de-dados/ Acessado em: 19 de setembro de 2024.

[2] scikit-learn developers. 2007 - 2024. *KNNImputer*. https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html Acessado em: 24 de setembro de 2024.