Documentação Técnica: Sistema de Diagnóstico Médico com Inteligência Artificial

Autores

João Victor Fernandes Souza

Vinicius Henrique de Oliveira Franzote

UNESP Bauru - Bacharelado em Sistemas de Informação

Demonstração Online: https://joao-vf-souza-projeto-final-ia-app-6ysln1.streamlit.app/

---

1. Introdução

Este documento apresenta o desenvolvimento completo de um sistema de diagnóstico médico automatizado baseado em Machine Learning, implementado como trabalho final do curso de Inteligência Artificial. O sistema utiliza algoritmos de aprendizado supervisionado para prever diagnósticos médicos a partir de sintomas reportados pelo usuário.

2. Objetivos do Projeto

2.1 Objetivo Geral

Desenvolver um sistema computacional capaz de realizar diagnósticos médicos preliminares a partir de sintomas informados, utilizando técnicas de Machine Learning para classificação multi-classe.

2.2 Objetivos Específicos

- Implementar um modelo de classificação com alta acurácia para diagnóstico de doenças
- Desenvolver interface web interativa para coleta de sintomas e apresentação de resultados
- Implementar sistema de classificação de níveis de emergência médica

- Avaliar e documentar métricas de desempenho do modelo
- Criar visualizações para análise de importância de features e probabilidades de diagnóstico

3. Fundamentação Teórica

3.1 Machine Learning em Diagnóstico Médico

O diagnóstico médico é um problema clássico de classificação onde, dado um conjunto de sintomas (features), deseja-se prever uma condição médica (classe). Algoritmos de Machine Learning são particularmente adequados para este tipo de problema devido à capacidade de identificar padrões complexos em grandes volumes de dados.

3.2 Random Forest Classifier

O Random Forest é um algoritmo de ensemble learning que constrói múltiplas árvores de decisão durante o treinamento e produz a classe que é moda das classes (classificação) das árvores individuais. As principais vantagens incluem:

- Robustez contra overfitting através de agregação de múltiplos modelos
- Capacidade de lidar com features não-lineares
- Fornecimento de métricas de importância de features
- Boa performance em datasets com alta dimensionalidade
- Não requer normalização de dados

4. Dataset

4.1 Origem e Características

O dataset utilizado é o **SymScan: Symptoms to Disease Dataset**, disponível na plataforma Kaggle (https://www.kaggle.com/datasets/behzadhassan/sympscan-symptomps-to-disease).

Características do dataset:

- Número de amostras: 96.088
- Número de features: 230 sintomas

- Número de classes: 100 diagnósticos diferentes
- Tipo de features: Binárias (0 = sintoma ausente, 1 = sintoma presente)
- Formato: CSV (Comma-Separated Values)

4.2 Estrutura dos Dados

O dataset está organizado em formato tabular onde:

- Primeira coluna: Nome da doença/diagnóstico (variável target)
- Colunas subsequentes: Sintomas binários (variáveis preditoras)

Exemplo da estrutura:

```
Disease | anxiety and nervousness | depression | shortness of breath | ...
--------|-------------------------|------------|---------------------|-----
Panic disorder | 1 | 0 | 1 | ...
Asthma | 0 | 0 | 1 | ...
```

4.3 Distribuição de Classes

O dataset apresenta classes relativamente balanceadas, com aproximadamente 960 amostras por doença. Esta distribuição equilibrada facilita o treinamento e evita viés do modelo em direção às classes majoritárias.

5. Metodologia

5.1 Pipeline de Desenvolvimento

O desenvolvimento seguiu as seguintes etapas:

1. Coleta e análise exploratória do dataset
2. Pré-processamento e codificação de labels
3. Divisão dos dados em conjuntos de treino e teste
4. Treinamento do modelo Random Forest
5. Avaliação de métricas de desempenho
6. Otimização de hiperparâmetros
7. Desenvolvimento da interface web
8. Implementação do sistema de níveis de emergência

9. Testes e validação

5.2 Pré-processamento de Dados

5.2.1 Codificação de Labels

As classes de diagnóstico (strings) foram convertidas para valores numéricos usando `LabelEncoder` do scikit-learn. Este processo é essencial pois algoritmos de ML requerem entrada numérica.

```
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

5.2.2 Divisão Treino-Teste

O dataset foi dividido em 80% para treinamento e 20% para teste, utilizando estratificação para manter a proporção de classes em ambos os conjuntos.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

5.3 Treinamento do Modelo

5.3.1 Hiperparâmetros do Random Forest

Após análise e testes, os seguintes hiperparâmetros foram otimizados para melhor generalização:

- `n_estimators=300`: Número de árvores de decisão na floresta
- `max_depth=40`: Profundidade máxima de cada árvore
- `min_samples_split=5`: Número mínimo de amostras necessárias para dividir um nó
- `min_samples_leaf=2`: Número mínimo de amostras necessárias em um nó folha
- `max_features='log2'`: Número de features consideradas em cada divisão
- `max_samples=0.8`: Proporção de amostras usadas por árvore (bootstrap)
- `min_impurity_decrease=0.0001`: Penalidade mínima para realizar splits
- `ccp_alpha=0.001`: Parâmetro de poda (pruning) para reduzir overfitting

- `criterion='gini'`: Função para medir qualidade da divisão
- `class_weight='balanced'`: Ajuste automático de pesos para classes desbalanceadas
- `random_state=42`: Semente para reprodutibilidade

5.3.2 Justificativa dos Hiperparâmetros

n_estimators=300: Define o número de árvores de decisão na floresta. Mais árvores aumentam a estabilidade e precisão das previsões através de agregação, mas também aumentam o tempo computacional. 300 oferece bom equilíbrio entre performance e eficiência.

max_depth=40: Limita a profundidade máxima de cada árvore, controlando a complexidade do modelo. Valores muito altos podem causar overfitting (memorização dos dados de treino), enquanto valores muito baixos podem causar underfitting. 40 permite capturar padrões complexos sem sobreajuste.

min_samples_split=5 e min_samples_leaf=2: Estes parâmetros controlam quando uma divisão pode ocorrer na árvore. `min_samples_split` exige pelo menos 5 amostras para criar uma nova divisão, enquanto `min_samples_leaf` garante que cada folha tenha pelo menos 2 amostras. Isso previne a criação de regras muito específicas baseadas em poucos exemplos.

max_features='log2': Determina quantas features são consideradas aleatoriamente em cada divisão. Usar logaritmo base 2 do total de features (log2(230) ≈ 8) introduz diversidade entre as árvores, melhorando a capacidade de generalização do ensemble.

max_samples=0.8: Cada árvore é treinada com apenas 80% das amostras, selecionadas aleatoriamente. Esta técnica de bootstrap reduz a correlação entre árvores e aumenta a robustez do modelo contra outliers e ruído nos dados.

ccp_alpha=0.001: Parâmetro de poda (pruning) que remove galhos da árvore que contribuem minimamente para a redução de impureza. Valores pequenos como 0.001 fazem poda conservadora, removendo apenas divisões claramente desnecessárias.

class_weight='balanced': Ajusta automaticamente os pesos das classes inversamente proporcionais às suas frequências. Isso garante que classes menos representadas no dataset não sejam negligenciadas durante o treinamento.

5.4 Métricas de Avaliação

5.4.1 Métricas Utilizadas

Acurácia (Accuracy): Proporção de previsões corretas sobre o total de previsões.

```
Accuracy = (TP + TN) / (TP + TN + FP + FN)
```

Precisão (Precision): Proporção de previsões positivas corretas sobre todas as previsões positivas.

```
Precision = TP / (TP + FP)
```

Recall (Sensibilidade): Proporção de positivos reais identificados corretamente.

```
Recall = TP / (TP + FN)
```

5.4.2 Resultados Obtidos

Métrica	Treino	Teste
Acurácia	88.90%	89.22%
Precisão	-	91.30%
Recall	-	89.22%

Análise dos Resultados:

A acurácia de teste (89.22%) **superior** à acurácia de treino (88.90%) indica **excelente capacidade de generalização** do modelo. O gap negativo de -0.32% demonstra que o modelo não está sofrendo overfitting - na verdade, está performando ligeiramente melhor em dados não vistos.

A precisão de 91.30% no conjunto de teste indica alta confiabilidade nas previsões positivas do modelo, ou seja, quando o modelo diagnostica uma doença específica, há 91.3% de probabilidade de estar correto.

O recall de 89.22% (idêntico à acurácia em problemas multi-classe balanceados) demonstra que o modelo identifica corretamente 89.22% dos casos reais de cada doença, indicando boa sensibilidade diagnóstica.

Este resultado foi alcançado através de:

- Hiperparâmetros conservadores que priorizam generalização
- Uso de técnicas de regularização (pruning com ccp_alpha)
- Bagging agressivo (max_samples=0.8)
- Limitação da profundidade e complexidade das árvores

O modelo demonstra robustez adequada para aplicação em cenário real de triagem médica preliminar.

5.5 Análise de Importância de Features

O Random Forest fornece métricas de importância de features através do cálculo de diminuição média de impureza (Mean Decrease in Impurity). Os 10 sintomas mais importantes identificados foram:

1. hot flashes (ondas de calor) - 1.3%
2. symptoms of the scrotum and testes (sintomas escrotais e testiculares) - 1.2%
3. symptoms of the face (sintomas faciais) - 1.2%
4. itchy ear(s) (coceira no ouvido) - 1.1%
5. pus draining from ear (pus drenando do ouvido) - 1.1%
6. back cramps or spasms (cãibras ou espasmos nas costas) - 1.0%
7. vomiting blood (vômito com sangue) - 1.0%
8. pain during intercourse (dor durante relação sexual) - 1.0%
9. mouth ulcer (úlcera na boca) - 0.9%
10. coughing up sputum (tosse com expectoração) - 0.9%

Esta análise demonstra que sintomas específicos e distintivos possuem maior poder discriminativo no modelo otimizado. A distribuição mais uniforme de importância (variando de 0.9% a 1.3%) indica que o modelo considera múltiplos sintomas de forma equilibrada, reduzindo dependência de features individuais.

6. Arquitetura do Sistema

6.1 Estrutura de Arquivos

```
projeto-final-ia/
├── app.py                    # Interface Streamlit
├── train_model_real.py       # Script de treinamento
├── emergency_level.py        # Sistema de níveis de emergência
├── classifier.py             # Classificador sintético (não utilizado)
├── requirements.txt          # Dependências Python
├── README.md                 # Documentação de usuário
├── DOCUMENTACAO.md           # Documentação técnica
├── .gitignore                # Arquivos ignorados pelo Git
└── data/
    ├── Diseases_and_Symptoms_dataset.csv  # Dataset principal
    ├── description.csv                    # Descrições de doenças
    └── model_real.pkl                     # Modelo treinado serializado
```

6.2 Componentes do Sistema

6.2.1 train_model_real.py

Módulo responsável pelo treinamento do modelo. Contém a classe `DiagnosticClassifierReal` que encapsula toda a lógica de:

- Carregamento e processamento do dataset
- Treinamento do Random Forest
- Cálculo de métricas
- Serialização do modelo treinado

Principais métodos:

- `load_real_dataset(csv_path)`: Carrega e processa o dataset
- `train(df)`: Executa o pipeline de treinamento
- `predict(symptoms_dict)`: Realiza predições
- `save(path)`: Serializa o modelo usando joblib

6.2.2 app.py

Aplicação web desenvolvida com Streamlit. Estruturada em quatro abas principais:

Aba Diagnóstico:

- Interface de seleção de sintomas (checkboxes)
- Botão para realizar diagnóstico
- Exibição de resultado com nível de confiança
- Descrição da condição diagnosticada
- Top 3 diagnósticos alternativos
- Classificação de nível de emergência
- Gráfico de probabilidades por diagnóstico

Aba Métricas:

- Informações do modelo (número de sintomas, doenças, tipo)
- Métricas de desempenho (acurácia, precisão, recall)
- Gráfico de importância de features (top 20)
- Distribuição de diagnósticos no dataset

Aba Informações:

- Descrição do modelo e metodologia
- Informações sobre o dataset
- Aviso de uso educacional
- Stack tecnológica utilizada

Aba Dados:

- Visualização do dataset completo
- Filtros por diagnóstico
- Estatísticas do dataset
- Botão para download em CSV

6.2.3 emergency_level.py

Módulo que implementa sistema de classificação de níveis de emergência médica baseado em diagnóstico e confiança. Define quatro níveis:

Verde (Emergência Baixa):

- Condições não urgentes
- Recomendação: Consulta em dias em posto de saúde

- Exemplos: Acne, alergias leves, resfriado comum

Amarelo (Urgência):

- Condições que requerem atenção médica em horas
- Recomendação: Procurar UPA (Unidade de Pronto Atendimento)
- Exemplos: Febre persistente, dor moderada, infecções

Laranja (Emergência):

- Condições graves que requerem atendimento no mesmo dia
- Recomendação: Procurar pronto-socorro
- Exemplos: Pneumonia, apendicite, fraturas

Vermelho (Crítica):

- Condições com risco de vida
- Recomendação: Ligar 192 (SAMU) imediatamente
- Exemplos: Infarto, AVC, trauma grave

6.3 Fluxo de Dados

```
[Usuário]
   ↓ (Seleciona sintomas)
[Interface Web - Streamlit]
   ↓ (symptoms_dict: {sintoma: 0/1})
[Modelo Random Forest]
   ↓ (Vetorização e predição)
[Resultado]
   ├─ Diagnóstico principal
   ├─ Nível de confiança
   ├─ Probabilidades de todos diagnósticos
   └─ Nível de emergência
   ↓
[Visualização]
   ├─ Descrição da condição
   ├─ Top 3 diagnósticos
   ├─ Gráfico de probabilidades
   └─ Recomendações de ação
```

7. Tecnologias Utilizadas

7.1 Linguagem e Ambiente

- **Python 3.11:** Linguagem de programação principal
- **pip:** Gerenciador de pacotes Python

7.2 Bibliotecas de Machine Learning

- **scikit-learn 1.7.2:** Framework principal de ML
  - RandomForestClassifier: Algoritmo de classificação
  - train_test_split: Divisão de dados
  - LabelEncoder: Codificação de labels
  - Métricas: accuracy_score, precision_score, recall_score
- **NumPy 1.26.4:** Computação numérica e operações matriciais
- **Pandas 2.1.1:** Manipulação e análise de dados tabulares

7.3 Bibliotecas de Visualização

- **Streamlit 1.28.1:** Framework para criação de aplicações web interativas
- **Plotly 5.17.0:** Biblioteca de visualizações interativas
- **Matplotlib 3.8.1:** Biblioteca de visualizações estáticas

7.4 Bibliotecas Auxiliares

- **Joblib 1.3.2:** Serialização eficiente de modelos scikit-learn
- **Pillow 10.0.1:** Manipulação de imagens

7.5 Justificativa das Escolhas Tecnológicas

scikit-learn: Biblioteca madura e amplamente utilizada, com implementações otimizadas de algoritmos de ML e excelente documentação.

Streamlit: Permite criação rápida de interfaces web interativas com código Python puro, sem necessidade de conhecimento em HTML/CSS/JavaScript.

Plotly: Gráficos interativos que melhoram a experiência do usuário na exploração de dados e resultados.

8. Implementação Detalhada

8.1 Classe DiagnosticClassifierReal

```
class DiagnosticClassifierReal:
    def __init__(self):
        self.model = None
        self.label_encoder = None
        self.symptoms_list = None
        self.diagnoses = None
        self.feature_importance = None
        self.metrics = None
```

Atributos:

- `model`: Instância do RandomForestClassifier treinado
- `label_encoder`: Codificador de diagnósticos (string → int)
- `symptoms_list`: Lista ordenada de todos os sintomas (features)
- `diagnoses`: Lista de todos os diagnósticos possíveis (classes)
- `feature_importance`: Dicionário {sintoma: importância}
- `metrics`: Dicionário contendo métricas de desempenho

8.2 Método de Predição

```
def predict(self, symptoms_dict):
    # Criar vetor de features na ordem correta
    X = np.array([[symptoms_dict.get(s, 0) for s in self.symptoms_list]])

    # Predição de classe e probabilidades
    y_pred = self.model.predict(X)[0]
    y_proba = self.model.predict_proba(X)[0]

    # Decodificar diagnóstico
    diagnosis = self.label_encoder.inverse_transform([y_pred])[0]
    confidence = y_proba[y_pred]

    # Gerar dicionário de probabilidades
```

```
    all_probabilities = dict(zip(
        self.label_encoder.classes_,
        y_proba
    ))

    return diagnosis, confidence, all_probabilities
```

Funcionamento:

1. Converte dicionário de sintomas em vetor NumPy mantendo ordem das features
2. Aplica modelo para obter classe predita e probabilidades
3. Decodifica classe numérica de volta para nome do diagnóstico
4. Extrai confiança da predição
5. Cria dicionário com probabilidades de todas as classes
6. Retorna tupla (diagnóstico, confiança, probabilidades)

8.3 Sistema de Níveis de Emergência

```
class EmergencyLevel:
    VERDE = {
        'level': 'Verde',
        'color': '🟢',
        'descricao': 'Emergência Baixa',
        'acao': 'Consultar em dias',
        'recomendacao': 'Agende consulta em posto de saúde'
    }

    @staticmethod
    def get_level(diagnosis, confidence):
        diagnosis_lower = diagnosis.lower()

        # Classificação crítica
        if any(keyword in diagnosis_lower for keyword in
               ['heart attack', 'stroke', 'hemorrhage']):
            return EmergencyLevel.VERMELHO
```

```
        # Classificação por confiança
        if confidence >= 0.8:
            return EmergencyLevel.AMARELO

        return EmergencyLevel.VERDE
```

Lógica de Classificação:

1. Verifica palavras-chave críticas no diagnóstico
2. Avalia nível de confiança da predição
3. Retorna dicionário com informações do nível de emergência
4. Combina análise semântica com confiança estatística

9. Serialização e Persistência

9.1 Salvamento do Modelo

O modelo treinado é serializado usando Joblib, que oferece compressão eficiente para objetos NumPy:

```
def save(self, path='data/model_real.pkl'):
    os.makedirs(os.path.dirname(path) or '.', exist_ok=True)
    joblib.dump(self, path)
```

Vantagens do Joblib:

- Compressão eficiente de arrays NumPy
- Preservação de estrutura de objetos complexos
- Carregamento rápido
- Compatibilidade com scikit-learn

9.2 Carregamento do Modelo

Streamlit utiliza cache para evitar recarregamento desnecessário:

```
@st.cache_resource
def load_model():
    model_path = 'data/model_real.pkl'
    if not os.path.exists(model_path):
```

```
        st.error("Modelo não encontrado!")
        st.stop()

    classifier = joblib.load(model_path)
    return classifier
```

O decorador `@st.cache_resource` mantém o modelo em cache durante a sessão, melhorando performance.

10. Interface do Usuário

10.1 Design da Interface

A interface foi desenvolvida seguindo princípios de usabilidade:

- Layout responsivo em colunas
- Organização clara por abas funcionais
- Feedback visual imediato
- Cores semânticas para níveis de emergência
- Gráficos interativos para exploração de dados

10.2 Componentes Customizados

CSS customizado foi aplicado para melhorar apresentação:

```
st.markdown("""
    <style>
    .emergency-box-verde {
        background-color: #d4edda;
        padding: 20px;
        border-radius: 10px;
        border-left: 5px solid #28a745;
    }
    .emergency-box-vermelho {
        background-color: #f8d7da;
        padding: 20px;
        border-radius: 10px;
        border-left: 5px solid #dc3545;
    }
```

```
    </style>
""", unsafe_allow_html=True)
```

11. Validação e Testes

11.1 Validação de Entrada

O sistema valida que pelo menos um sintoma foi selecionado antes de realizar predição:

```
if not any(symptoms_selected.values()):
    st.error("Selecione pelo menos um sintoma!")
```

11.2 Tratamento de Erros

Implementação de blocos try-except para tratamento gracioso de erros:

- Arquivo de modelo não encontrado
- Erro ao carregar dataset
- Erro ao realizar predição

11.3 Testes de Consistência

Verificação de que métricas salvas correspondem ao modelo treinado através de armazenamento persistente no objeto do modelo.

12. Limitações e Trabalhos Futuros

12.1 Limitações Identificadas

Limitação 1 - Natureza Educacional:
O sistema foi desenvolvido para fins educacionais e não deve substituir consulta médica profissional.

Limitação 2 - Dataset Único:
Treinamento baseado em um único dataset pode limitar generalização.

Limitação 3 - Sintomas Binários:
Não captura intensidade ou duração dos sintomas.

Limitação 4 - Idioma:
Dataset e sintomas em inglês, limitando uso direto por usuários brasileiros.

Limitação 5 - Overfitting:
Diferença de 7.5 pontos percentuais entre acurácia de treino e teste indica leve overfitting.

12.2 Melhorias Propostas

Melhoria 1 - Multilíngue:
Implementar sistema de tradução para português e outros idiomas.

Melhoria 2 - Deep Learning:
Explorar redes neurais profundas para potencialmente melhorar acurácia.

Melhoria 3 - Features Adicionais:
Incorporar intensidade de sintomas, duração, idade, sexo, histórico médico.

Melhoria 4 - Ensemble Avançado:
Combinar múltiplos algoritmos (Random Forest, Gradient Boosting, SVM) através de voting ou stacking.

Melhoria 5 - API REST:
Desenvolver API para integração com outros sistemas de saúde.

Melhoria 6 - Aplicativo Mobile:
Versão mobile usando React Native ou Flutter.

Melhoria 7 - Explicabilidade:
Implementar SHAP ou LIME para explicar decisões do modelo ao usuário.

Melhoria 8 - Histórico de Usuário:
Armazenar histórico de consultas para análise temporal.

13. Aspectos Éticos e Legais

13.1 Considerações Éticas

O sistema apresenta avisos claros sobre sua natureza educacional e limitações. Enfatiza que não substitui atendimento médico profissional e encoraja busca de ajuda qualificada.

13.2 Privacidade de Dados

A aplicação atual não armazena dados pessoais de usuários. Todas as operações são realizadas em sessão temporária sem persistência de informações sensíveis.

13.3 Responsabilidade

O sistema está claramente marcado como projeto educacional e não deve ser utilizado para tomadas de decisão médica real.

14. Conclusão

Este projeto demonstrou a aplicabilidade de técnicas de Machine Learning no domínio de diagnóstico médico. O modelo Random Forest desenvolvido alcançou acurácia de 87.23% no conjunto de teste, demonstrando capacidade de generalização adequada para um sistema educacional.

A interface web desenvolvida em Streamlit fornece experiência de usuário intuitiva, permitindo fácil seleção de sintomas e visualização de resultados. O sistema de níveis de emergência adiciona camada importante de triagem, orientando usuários sobre urgência da condição.

Os resultados obtidos validam a hipótese de que algoritmos de Machine Learning podem auxiliar no processo de diagnóstico médico preliminar, desde que utilizados como ferramenta de apoio e não substituição de profissionais qualificados.

15. Referências

1. Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5-32.
2. Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.
3. Hassan, B. (2023). SymScan: Symptoms to Disease Dataset. Kaggle. https://www.kaggle.com/datasets/behzadhassan/sympscan-symptomps-to-disease
4. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer.
5. Streamlit Documentation. (2024). https://docs.streamlit.io/
6. Scikit-learn Documentation. (2024). https://scikit-learn.org/stable/

Apêndices

Apêndice A - Requisitos de Sistema

```
streamlit==1.28.1
scikit-learn==1.7.2
pandas==2.1.1
numpy==1.26.4
plotly==5.17.0
matplotlib==3.8.1
pillow==10.0.1
joblib==1.3.2
```

Apêndice B - Comandos de Execução

Treinamento do Modelo:

```
python train_model_real.py
```

Execução da Aplicação:

```
python -m streamlit run app.py
```

Apêndice C - Estrutura do Dataset

O arquivo CSV possui 231 colunas:

- Coluna 0: Disease (diagnóstico)
- Colunas 1-230: Sintomas binários (0/1)

Exemplo de sintomas:

- anxiety and nervousness
- depression
- shortness of breath
- sharp chest pain
- dizziness
- palpitations
- (... 224 sintomas adicionais)

Apêndice D - Matriz de Confusão

Devido ao grande número de classes (100), a matriz de confusão completa possui dimensão 100×100. As principais observações:

- Diagonal principal concentra maioria das predições (acertos)
- Confusões mais comuns ocorrem entre doenças com sintomas similares
- Classes bem separadas apresentam zero confusões

Documento elaborado em: Novembro de 2025

Versão: 1.0

Autores: Projeto Final - Inteligência Artificial