# 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

Repositório Github: https://github.com/joao-vf-souza/projeto-final-ia/

**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 (100 d
oenças mapeadas)
├── requirements.txt          # Dependências Python
├── README.md                 # Documentação de usuário
├── .gitignore                # Arquivos ignorados pelo Git
└── data/
    ├── Diseases_and_Symptoms_dataset.csv  # Dataset principal (96.088
amostras)
    ├── description.csv               # Descrições de 100 doenças
    └── model_real.pkl                # Modelo treinado (~50MB)
```

## 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 **mapeamento completo de 100 doenças**. Define quatro níveis:

**Verde (Emergência Baixa) - 44 doenças mapeadas:**

- Condições não urgentes
- Recomendação: Consulta em dias em posto de saúde
- Exemplos: common cold, eczema, seasonal allergies, acne, hemorrhoids

**Amarelo (Urgência) - 27 doenças mapeadas:**

- Condições que requerem atenção médica em horas
- Recomendação: Procurar UPA (Unidade de Pronto Atendimento)
- Exemplos: asthma, anxiety, gout, strep throat, injury to arm/leg

**Laranja (Emergência) - 20 doenças mapeadas:**

- Condições graves que requerem atendimento no mesmo dia
- Recomendação: Procurar pronto-socorro
- Exemplos: pneumonia, appendicitis, urinary tract infection, cholecystitis

**Vermelho (Crítica) - 9 doenças mapeadas:**

- Condições com risco de vida imediato
- Recomendação: Ligar 192 (SAMU) imediatamente
- Exemplos: heart attack, heart failure, sepsis, gastrointestinal hemorrhage, acute kidney injury

**Sistema de Avisos de Confiança:**

- Confiança < 25%: "Confiança muito baixa - CONSULTE UM MÉDICO"
- Confiança 25-40%: "Confiança moderada - Recomenda-se consulta médica"
- Confiança ≥ 40%: Sem aviso adicional

## 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 2.2.3:** Manipulação e análise de dados tabulares

## 7.3 Bibliotecas de Visualização

- **Streamlit 1.51.0:** 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:
    # Dicionário de mapeamento com 100 doenças classificadas
    DIAGNOSIS_MAPPING = {
```

```
        # VERMELHO - 9 doenças críticas
        'heart attack': 'VERMELHO',
        'heart failure': 'VERMELHO',
        'sepsis': 'VERMELHO',
        'acute pancreatitis': 'VERMELHO',
        # ... 5 mais

        # LARANJA - 20 doenças de emergência
        'appendicitis': 'LARANJA',
        'pneumonia': 'LARANJA',
        'cholecystitis': 'LARANJA',
        # ... 17 mais

        # AMARELO - 27 doenças de urgência moderada
        'asthma': 'AMARELO',
        'anxiety': 'AMARELO',
        # ... 25 mais

        # VERDE - 44 doenças de baixa urgência
        'common cold': 'VERDE',
        'eczema': 'VERDE',
        # ... 42 mais
    }

    @classmethod
    def get_level(cls, diagnosis, confidence=None):
        # Obtém nível do mapeamento (padrão: AMARELO)
        level_key = cls.DIAGNOSIS_MAPPING.get(diagnosis, 'AMARELO')
        level_info = cls.LEVELS[level_key].copy()
        level_info['level'] = level_key

        # Adiciona avisos baseados em confiança
        if confidence and confidence < 0.25:
            level_info['aviso'] = 'Confiança muito baixa - CONSULTE UM MÉDIC
O'
        elif confidence and confidence < 0.40:
            level_info['aviso'] = 'Confiança moderada - Consulta recomendada'
```

```
    return level_info
```

**Lógica de Classificação:**

1. **Mapeamento Direto**: Cada uma das 100 doenças é mapeada para um nível específico
2. **Fallback Padrão**: Doenças não mapeadas defaultam para AMARELO (precaução)
3. **Avisos de Confiança**: Ajustados para refletir realidade de classificação multi-classe
4. **Thresholds Realistas**: 25% e 40% (não 60% ou 80% que são inviáveis)

# 9. Análise de Confiança em Classificação Multi-Classe

## 9.1 Comportamento Esperado

Em problemas de classificação com **100 classes**, é matematicamente **normal e esperado** que os níveis de confiança sejam relativamente baixos:

**Estatísticas Observadas (teste em 100 amostras):**

- **Confiança Máxima**: ~75%
- **Confiança Média**: ~28%
- **Confiança Mediana**: ~24%
- **Confiança Mínima**: ~2%

**Distribuição Típica:**

- 63% das predições: confiança < 30%
- 22% das predições: confiança 30-50%
- 15% das predições: confiança > 50%
- 0% das predições: confiança > 90%

## 9.2 Por Que a Confiança é Baixa?

**Razão 1 - Diluição de Probabilidade:**
Com 100 classes, mesmo uma distribuição uniforme resultaria em 1% por

classe. O modelo precisa concentrar muito mais "votos" em uma classe para atingir alta confiança.

**Razão 2 - Sintomas Compartilhados:**
Muitas doenças compartilham sintomas similares (ex: febre, dor de cabeça, fadiga), dispersando os votos das 300 árvores entre múltiplas classes relacionadas.

**Razão 3 - Voting Mechanism:**
Cada árvore do Random Forest vota em uma classe. Com 300 árvores e 100 opções, mesmo a classe vencedora raramente recebe mais de 50% dos votos.

## 9.3 Impacto na Acurácia

**Importante**: Confiança baixa ≠ Acurácia baixa!

- O modelo tem **89.22% de acurácia** (prediz corretamente 9 em 10 casos)
- Mas confiança **média de apenas 28%**
- Isto significa: o modelo acerta frequentemente, mas com incerteza distribuída

**Exemplo Real:**

- Diagnóstico: "panic disorder"
- Confiança: 48.39%
- Top 2: "anxiety" com 6.21%
- **Predição: CORRETA**

O modelo identificou corretamente, mas a confiança ficou abaixo de 50% porque sintomas de pânico são similares aos de ansiedade e outras condições.

## 9.4 Thresholds Ajustados

Dados os níveis observados, o sistema foi ajustado com thresholds realistas:

- **< 25%**: Confiança muito baixa (abaixo da mediana)
- **25-40%**: Confiança moderada (em torno da média)
- **≥ 40%**: Confiança adequada (acima da média)

Thresholds anteriores de 60% ou 80% eram **inviáveis** e resultavam em alertas constantes mesmo para predições corretas.

# 10. Serialização e Persistência

## 10.1 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

## 10.2 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.

# 11. Interface do Usuário

## 11.1 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

### 11.2 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)
```

## 12. Validação e Testes

### 12.1 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!")
```

## 12.2 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

## 12.3 Testes de Consistência

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

# 13. Limitações e Trabalhos Futuros

## 13.1 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 - Confiança Baixa em Classificação Multi-Classe:**
Com 100 classes, o modelo apresenta confiança típica de 20-40%, com máxima observada de ~75%. Isto é normal e esperado em problemas com muitas classes, não indicando problema no modelo.

## 13.2 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.

# 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 89.22% no conjunto de teste, **superior** à acurácia de treino (88.90%), demonstrando excelente capacidade de generalização sem overfitting.

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. Hassan, B. (2023). SymScan: Symptoms to Disease Dataset. Kaggle. https://www.kaggle.com/datasets/behzadhassan/sympscan-symptomps-to-disease
2. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer.
3. Streamlit Documentation. (2024). https://docs.streamlit.io/

4. Scikit-learn Documentation. (2024). https://scikit-learn.org/stable/

# Apêndices

## Apêndice A - Requisitos de Sistema

```
streamlit==1.51.0
scikit-learn==1.7.2
pandas==2.2.3
numpy==1.26.4
plotly==5.17.0
matplotlib==3.8.1
pillow==10.4.0
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