

Explicação do Projeto de Classificação de Síndromes Genéticas

1. Entendimento do Desafio

O desafio proposto consiste em desenvolver um sistema completo de classificação de síndromes genéticas utilizando embeddings de imagens pré-processados. O objetivo principal é implementar um pipeline de machine learning que realize as seguintes tarefas:

1.1 Processamento de Dados

- Carregar e pré-processar dados de um arquivo pickle contendo embeddings hierárquicos
- Transformar a estrutura hierárquica (syndrome_id, subject_id, image_id) em formato adequado para análise
- Garantir a integridade dos dados e lidar com possíveis inconsistências

1.2 Análise Exploratória de Dados

- Fornecer estatísticas sobre o dataset (número de síndromes, imagens por síndrome)
- Identificar e discutir desbalanceamentos nos dados ou padrões observados

1.3 Visualização de Dados

- Utilizar t-SNE para reduzir a dimensionalidade dos embeddings para 2D
- Gerar gráficos que visualizem os embeddings coloridos por syndrome_id
- Interpretar clusters ou padrões na visualização e sua relação com a tarefa de classificação

1.4 Tarefa de Classificação

- Implementar o algoritmo K-Nearest Neighbors (KNN) para classificar os embeddings nas respectivas síndromes
- Comparar métricas de distância Cosine e Euclidean
- Realizar validação cruzada de 10 dobras para avaliar o desempenho do modelo
- Determinar o valor ótimo de k (de 1 a 15) usando validação cruzada
- Implementar cálculos de AUC, F1-Score e Top-k Accuracy
- Comparar resultados entre as duas métricas de distância

1.5 Métricas e Avaliação

- Gerar curvas ROC AUC para ambas as métricas de distância
- Criar tabelas resumindo métricas de performance para ambos os algoritmos
- Garantir que as tabelas sejam claras e geradas automaticamente pelo código

1.6 Relatório Técnico

- Escrever um relatório detalhado com metodologia, resultados, análise e recomendações

- Descrever desafios encontrados e soluções implementadas
- Propor melhorias e próximos passos

1.7 Perguntas de Interpretação

- Responder a perguntas teóricas sobre interpretação de modelos de machine learning
- Relacionar conceitos teóricos com os resultados práticos obtidos

2. Introdução

Este documento fornece uma explicação abrangente sobre como o sistema de classificação de síndromes genéticas foi desenvolvido, incluindo as bibliotecas utilizadas, o processo de desenvolvimento dos principais arquivos, os resultados obtidos e o treinamento realizado, com base no desafio proposto.

3. Bibliotecas e Tecnologias Utilizadas

3.1 Bibliotecas Importadas em ml_project.py

pickle

- Função: Utilizado para serializar e desserializar objetos Python
- Uso específico: Carregar o arquivo `mini_gm_public_v0.1.p` que contém os embeddings de síndromes genéticas em formato binário

numpy (importado como np)

- Função: Biblioteca fundamental para computação científica com arrays multidimensionais
- Uso específico: Manipulação de arrays de embeddings (320 dimensões), operações matemáticas, cálculos estatísticos e manipulação de dados numéricos

pandas (importado como pd)

- Função: Biblioteca para manipulação e análise de dados em formato tabular
- Uso específico: Criação de DataFrames para armazenar e exportar tabelas de desempenho (`performance_summary.csv` e `performance_detailed.csv`)

matplotlib e matplotlib.pyplot (importado como plt)

- Função: Biblioteca para criação de visualizações e gráficos
- Uso específico:
 - o `matplotlib.use('Agg')` configura o backend não interativo para geração de gráficos sem interface gráfica
 - o Criação de gráficos de distribuição, visualizações t-SNE, comparação de métricas e curvas ROC

seaborn (importado como sns)

- Função: Biblioteca para visualização estatística baseada no matplotlib
- Uso específico: Embora importado, não parece ser utilizado ativamente no código fornecido

scikit-learn (sklearn)

- Função: Biblioteca principal para machine learning em Python
- Módulos específicos utilizados:
 - o `sklearn.manifold.TSNE`: Para redução de dimensionalidade e visualização dos embeddings em 2D
 - o `sklearn.model_selection.cross_val_score`, `StratifiedKFold`: Para validação cruzada estratificada
 - o `sklearn.neighbors.KNeighborsClassifier`: Implementação do algoritmo K-Nearest Neighbors
 - o `sklearn.metrics`: Módulos para cálculo de métricas (`roc_auc_score`, `f1_score`, `accuracy_score`, `roc_curve`, `auc`)

scipy

- Função: Biblioteca para computação científica e matemática
- Módulos específicos utilizados:
 - o `scipy.spatial.distance.cosine`, `euclidean`: Funções para cálculo de distâncias (embora não sejam usadas diretamente como funções de distância no KNN, estão disponíveis)
 - o `scipy.interpolate.interp1d`: Para interpolação de curvas ROC multiclasse

os

- Função: Módulo para interação com o sistema operacional
- Uso específico: Criação de diretórios (como o diretório 'results') e manipulação de caminhos

collections

- Função: Módulo que fornece tipos de dados especializados
- Uso específico: Importa `defaultdict` (embora não seja utilizado ativamente no código fornecido)

warnings

- Função: Módulo para lidar com avisos do Python
- Uso específico: `warnings.filterwarnings('ignore')` para suprimir avisos durante a execução

sklearn.preprocessing.LabelEncoder

- Função: Utilizado para codificar rótulos categóricos em valores numéricos
- Uso específico: Converter rótulos de síndromes em formatos numéricos para cálculo de curvas ROC multiclasse

2.2 Bibliotecas Importadas em create_report.py e interpretation_report_pdf.py

fpdf

- Função: Biblioteca para criação de documentos PDF
- Uso específico: Geração dos relatórios em formato PDF (genetic_syndrome_classification_report.pdf e interpretation_analysis_report.pdf)

3. Desenvolvimento do ml_project.py

O arquivo ml_project.py é o núcleo do projeto de classificação de síndromes genéticas. Seu processo de desenvolvimento envolveu as seguintes etapas principais:

3.1 Funções Principais

load_data(filepath)

- Utiliza `pickle` para carregar o dataset serializado
- Retorna a estrutura hierárquica de síndromes, sujeitos e embeddings

flatten_data(data)

- Transforma a estrutura hierárquica em arrays planos
- Retorna embeddings (n_samples, 320), labels e metadados
- Utiliza `numpy` para manipulação eficiente dos arrays

exploratory_data_analysis(labels)

- Realiza análise estatística dos dados
- Utiliza `numpy` para calcular estatísticas descritivas
- Gera gráfico de distribuição usando `matplotlib`
- Salva o gráfico como `distribution_syndromes.png`

visualize_embeddings_tsne(embeddings, labels)

- Aplica t-SNE para redução de dimensionalidade
- Usa `sklearn.manifold.TSNE` para transformação não linear
- Cria visualização 2D para entender agrupamentos de síndromes
- Salva o resultado como `tsne_visualization.png`

`knn_classification(embeddings, labels, distance_metric)`

- Implementação do algoritmo K-Nearest Neighbors
- Utiliza `sklearn.neighbors.KNeighborsClassifier` com diferentes métricas
- Implementa validação cruzada com `sklearn.model_selection.StratifiedKFold`
- Calcula múltiplas métricas de performance:
 - o AUC (Area Under the ROC Curve) com `sklearn.metrics.roc_auc_score`
 - o F1-Score com `sklearn.metrics.f1_score`
 - o Accuracy com `sklearn.metrics.accuracy_score`
 - o Top-k Accuracy com implementação customizada

`plot_metrics_comparison(euclidean_results, cosine_results)`

- Cria gráficos comparativos de desempenho
- Visualiza métricas AUC, F1-Score, Accuracy e Top-k Accuracy
- Utiliza `matplotlib` para gráficos de linha
- Salva o gráfico como `metric_comparison.png`

`plot_roc_curves(embeddings, labels, euclidean_k, cosine_k)`

- Gera curvas ROC para ambos os métodos
- Calcula curvas ROC multiclasse
- Usa `sklearn.metrics.roc_curve` e `auc` para cálculo das curvas
- Utiliza `scipy.interpolate.interp1d` para interpolação de curvas
- Salva como `roc_curves_comparison.png`

4. Desenvolvimento do `create_report.py`

O arquivo `create_report.py` gera um relatório PDF (`genetic_syndrome_classification_report.pdf`) contendo:

- Uso de `fpdf` para estruturação do documento
- Criação de cabeçalhos e rodapés personalizados
- Seções com introdução, metodologia, resultados, análise e recomendações
- Formatação de texto em diferentes estilos (negrito, itálico)
- Exportação em formato PDF

5. Desenvolvimento do `interpretation_report_pdf.py`

O arquivo `interpretation_report_pdf.py` cria um relatório de interpretação (`interpretation_analysis_report.pdf`) que:

- Utiliza a mesma estrutura de `create_report.py` com `fpdf`
- Inclui análise teórica de conceitos de machine learning
- Incorpora imagens dos resultados no documento PDF
- Relaciona conceitos teóricos com resultados práticos
- Explica tradeoffs de bias-variance e interpretação de modelos

6. Estrutura do Dataset (mini_gm_public_v0.1.p)

O arquivo `mini_gm_public_v0.1.p` contém dados serializados com a seguinte estrutura:

- Formato: Arquivo pickle com estrutura hierárquica `{syndrome_id: {subject_id: {image_id: embedding_vector}}}`
- Tipo de dados: Vetores de embeddings de 320 dimensões
- Quantidade: 1,116 amostras de 10 diferentes síndromes genéticas
- Origem: Vetores de características extraídos de um modelo de classificação pré-treinado
- Dimensão dos embeddings: 320 dimensões cada

7. Processo de Treinamento e Validação

7.1 Estratégia de Validação

- Validação Cruzada: 10-fold StratifiedKFold para manter a distribuição proporcional das classes
- Random State: Utilizado para reprodutibilidade dos resultados (`random_state=42`)

7.2 Métricas de Avaliação

- AUC (Area Under the ROC Curve): Medida da capacidade de discriminação do modelo
- F1-Score: Média harmônica entre precisão e recall (média ponderada para multiclasse)
- Accuracy: Proporção de previsões corretas
- Top-k Accuracy: Porcentagem de amostras cuja classe correta está entre as k previsões mais prováveis

7.3 Algoritmo e Configurações

- Algoritmo: K-Nearest Neighbors (KNN)
- Métricas de Distância Testadas: Euclidean e Cosine
- Valores de k Testados: 1 a 15
- Implementação: `sklearn.neighbors.KNeighborsClassifier`

8. Resultados Obtidos

8.1 Métricas de Desempenho

- Melhor Configuração: Distância Cosine com $k=15$
- AUC Máxima: 0.9630 (Cosine) vs 0.9504 (Euclidean)
- F1-Score Máximo: 0.7874 (Cosine) vs 0.7547 (Euclidean)
- Accuracy Máxima: 0.7948 (Cosine) vs 0.7634 (Euclidean)

8.2 Top-k Accuracy

- Top-1 Accuracy: 0.7948 (Cosine) vs 0.7634 (Euclidean)
- Top-3 Accuracy: 0.9418 (Cosine) vs 0.9247 (Euclidean)
- Top-5 Accuracy: 0.9749 (Cosine) vs 0.9659 (Euclidean)

9. Arquivos de Resultados Gerados

9.1 Arquivos CSV

- performance_summary.csv: Tabela resumida com métricas para os melhores valores de k
- performance_detailed.csv: Tabela detalhada com métricas para todos os valores de k (1 a 15)

9.2 Arquivos de Imagem

- distribution_syndromes.png: Gráfico de barras mostrando distribuição de amostras por síndrome
- tsne_visualization.png: Visualização 2D dos embeddings após aplicação de t-SNE
- metric_comparison.png: Gráficos comparativos de todas as métricas para diferentes valores de k
- roc_curves_comparison.png: Curvas ROC comparando os dois métodos de distância

9.3 Arquivos PDF

- genetic_syndrome_classification_report.pdf: Relatório completo com metodologia, resultados e análise
- interpretation_analysis_report.pdf: Relatório com interpretação teórica dos resultados e conceitos de ML

10. Considerações Técnicas Adicionais

10.1 Configurações de Ambiente

- Backend Matplotlib: Configurado como 'Agg' para permitir execução em ambientes sem interface gráfica

- Supressão de Avisos: Utilização de `warnings.filterwarnings('ignore')` para evitar saída excessiva

10.2 Manipulação de Dados

- Codificação de Rótulos: Utilização de `Label Encoder` para converter rótulos categóricos em numéricos
- Interpolação de Curvas ROC: Uso de `scipy.interpolate.interp1d` para interpolação de curvas em espaço de FPR comum




10.3 Visualização Estatística

- Paletas de Cores: Utilização de `plt.cm.tab10` para diferenciar classes nos gráficos
- Layout de Gráficos: Configuração de layout adequado com `plt.tight_layout()`

11. Conclusão

O projeto demonstra uma implementação completa de classificação de síndromes genéticas utilizando embeddings pré-treinados e algoritmos de machine learning. A combinação de várias bibliotecas especializadas do ecossistema Python possibilita uma pipeline robusta de análise, treinamento e avaliação, com resultados que demonstram alta capacidade discriminativa (AUC de 0.9630) usando uma abordagem baseada em similaridade angular (distância cosseno) entre vetores de características.

com base na análise do desafio proposto e do sistema desenvolvido, posso afirmar que consegui entregar tudo o que foi colocado como desafio. Veja como cada requisito foi atendido:

1. Processamento de Dados 
 - Carregamento e pré-processamento: O arquivo `ml_project.py` carrega os dados do arquivo `pickle mini_gm_public_v0.1.p`
 - Transformação hierárquica: A função `flatten_data()` transforma a estrutura `{syndrome_id: {subject_id: {image_id: embedding_vector}}}` em arrays planos de embeddings e rótulos
 - Integridade dos dados: O sistema processa corretamente os 1,116 embeddings de 320 dimensões
2. Análise Exploratória de Dados 
 - Estatísticas do dataset: A função `exploratory_data_analysis()` fornece contagem de amostras por síndrome, número total de amostras (1,116), número de síndromes diferentes (10), média, desvio padrão, mínimo e máximo de amostras por síndrome
 - Análise de desbalanceamento: O sistema identifica e mostra o desbalanceamento entre classes (64 a 210 amostras por síndrome)
3. Visualização de Dados 
 - t-SNE 2D: A função `visualize_embeddings_tsne()` reduz os embeddings de 320 para 2 dimensões
 - Visualização colorida: Os embeddings são visualizados com cores diferentes por `syndrome_id`
 - Análise de clusters: A visualização permite identificar agrupamentos e padrões

4. Tarefa de Classificação

- KNN implementado: O algoritmo K-Nearest Neighbors está implementado com `sklearn.neighbors.KNeighborsClassifier`
- Métricas de distância: Ambas as distâncias (Cosine e Euclidean) são testadas e comparadas
- Validação cruzada: 10-fold cross-validation com `StratifiedKFold` implementada
- Otimização de k: Valores de k de 1 a 15 são testados e o ótimo determinado
- Métricas implementadas: AUC, F1-Score e Top-k Accuracy calculados e comparados

5. Métricas e Avaliação

- Curvas ROC AUC: Geradas para ambas as métricas de distância e plotadas no mesmo gráfico
- Tabelas de performance: Arquivos CSV gerados automaticamente (`performance_summary.csv` e `performance_detailed.csv`)
- Comparação: Resultados comparados entre as duas métricas de distância

6. Relatório Técnico

- Metodologia: Documentada em `create_report.py` e no arquivo `explicacao.md`
- Resultados: Apresentados com gráficos e tabelas
- Análise: Com interpretação dos resultados e comparação entre métricas
- Desafios e soluções: Identificados e documentados
- Recomendações: Propostas para melhorias futuras

7. Perguntas de Interpretação

- Relatório de interpretação: Gerado em `interpretation_analysis_report.pdf` respondendo às questões teóricas
- Conexão teoria-prática: Relaciona conceitos teóricos com os resultados obtidos

8. Deliverables

- Scripts Python: `ml_project.py`, `create_report.py`, `interpretation_report_pdf.py`
- Arquivo de requisitos: `requirements.txt`
- README: Presente
- PDF de relatório: `genetic_syndrome_classification_report.pdf`
- PDF de interpretação: `interpretation_analysis_report.pdf`

Resultados Obtidos:

- AUC Máxima: 0.9630 (Cosine) vs 0.9504 (Euclidean)
- F1-Score Máximo: 0.7874 (Cosine) vs 0.7547 (Euclidean)

- Accuracy Máxima: 0.7948 (Cosine) vs 0.7634 (Euclidean)
- Top-k Accuracy: Completamente implementado e documentado

CONCLUSÃO: O projeto atende integralmente a todos os requisitos do desafio proposto, com implementação completa, análise de dados, visualização, classificação, métricas de avaliação e relatórios técnicos.