UNIVERSIDADE FEDERAL DO PARANÁ

CAIO YUZO HIRAGA
FERNANDO GUILHERME GORSKI
GABRIEL KAUÊ CAITANO ALVES
LUCAS MASSAYUKI GOHARA

CLASSIFICAÇÕES TAXONÔMICAS USANDO INTELIGÊNCIA ARTIFICAL

CURITIBA
2024

CAIO YUZO HIRAGA
FERNANDO GUILHERME GORSKI
GABRIEL KAUÊ CAITANO ALVES
LUCAS MASSAYUKI GOHARA

CLASSIFICAÇÕES TAXONÔMICAS USANDO INTELIGÊNCIA ARTIFICAL

Trabalho apresentado ao curso de Tecnologia em Análise e Desenvolvimento de Sistemas, no Setor de Educação Profissional e Tecnológica, na Universidade Federal do Paraná, em cumprimento das exigências da disciplina Inteligência Artificial Aplicada 1 (DS803), como parte da nota parcial.

Orientador: Prof. Dr. Roberto Tadeu Raittz.

Coorientadora: Me. Camila Pereira Perico.

CURITIBA

2024

RESUMO

Este relatório descreve o desenvolvimento de um modelo de rede neural para a classificação de bactérias com base em suas respectivas classes taxonômicas. O modelo foi projetado utilizando a ferramenta Octave e um conjunto de dados estruturados, composto por vetores numéricos representando sequências biológicas. A solução foi construída com várias etapas essenciais, incluindo a visualização gráfica dos dados, normalização, oversampling para balanceamento de classes, redução de dimensionalidade através da Análise de Componentes Principais (PCA), e a aplicação de uma rede neural do tipo Perceptron Multicamadas (MLP). A validação cruzada k-fold foi utilizada para avaliar o desempenho do modelo. Os resultados demonstraram que o modelo teve um bom desempenho, com F1-scores médios variando entre 0.82 e 0.93, indicando que a classificação das amostras foi bem-sucedida. O trabalho propôs uma solução eficaz para o problema de classificação automatizada de bactérias, superando os desafios do desbalanceamento de classes e da alta dimensionalidade dos dados.

Palavras-chave: Rede-Neural, Classificação-de-Bactérias, PCA, Oversampling, Normalização, Validação-Cruzada, Taxonomia, F1-score, Octave, Inteligência-Artificial, Análise-de-Dados-Biológicos.

ABSTRACT

This report outlines the development of a neural network model for the classification of bacteria according to their taxonomic classes. The model was implemented using Octave and a dataset of numerical vectors representing biological sequences. The solution was built through several key steps, including graphical data visualization, normalization, oversampling for class balance, dimensionality reduction using Principal Component Analysis (PCA), and the application of a Multilayer Perceptron (MLP) neural network. k-fold cross-validation was employed to evaluate the model's performance. The results showed that the model performed well, with average F1-scores ranging from 0.82 to 0.93, indicating successful classification of the samples. This work proposed an effective solution to the automated classification of bacteria, addressing challenges such as class imbalance and high data dimensionality.

Keywords: Neural-Network, Bacteria-Classification, PCA, Oversampling, Normalization, Cross-Validation, Taxonomy, F1-score, Octave, Artificial-Intelligence, Biological-Data-Analysis.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Nesse relatório, está descrito o processo de implementação de uma rede neural para classificação de bactérias de acordo com suas respectivas classes taxonômicas. Para a realização do trabalho, a ferramenta de desenvolvimento escolhida foi o Octave, um *software* de aprendizado de máquina. O modelo foi projetado a partir de um conjunto de dados base e classes base, assim como novos dados fornecidos, para atividade de classificação se fez uso de técnicas como: normalização, oversampling, análise de componentes principais (PCA) e validação cruzada. É relatada, ainda neste documento, a execução das etapas e a descrição dos resultados obtidos, com detalhamentos em cada parte do processo.

2 PROBLEMA

Uma coleção de 2000 bactérias precisa ser classificada de acordo com suas respectivas classes taxonômicas. Devido ao alto custo e à demora do processo de classificação manual, é necessário desenvolver uma solução automatizada para essa tarefa.

As sequências biológicas (genomas) dessas bactérias foram convertidas em vetores numéricos estruturados de tamanho fixo, com comprimento de 7500, utilizando a ferramenta rSWeeP. Esse formato permite que as amostras sejam comparadas a um banco de referência contendo 5115 vetores já classificados em classes taxonômicas bem definidas.

O objetivo é desenvolver um modelo de inteligência artificial (IA) capaz de classificar corretamente as bactérias do conjunto amostral, composto por 2000 vetores ainda sem classes definidas. A distribuição dessas amostras entre as classes esperadas é a seguinte:

- Alteromonadales: 128
- Aeromonadales: 137
- Moraxellales: 394
- Xanthomonadales: 393
- Pasteurellales: 180
- Enterobacterales: 2820
- Pseudomonadales: 628
- Vibrionales: 207
- Outros (grupos menores): 228

3 SOLUÇÃO PROPOSTA

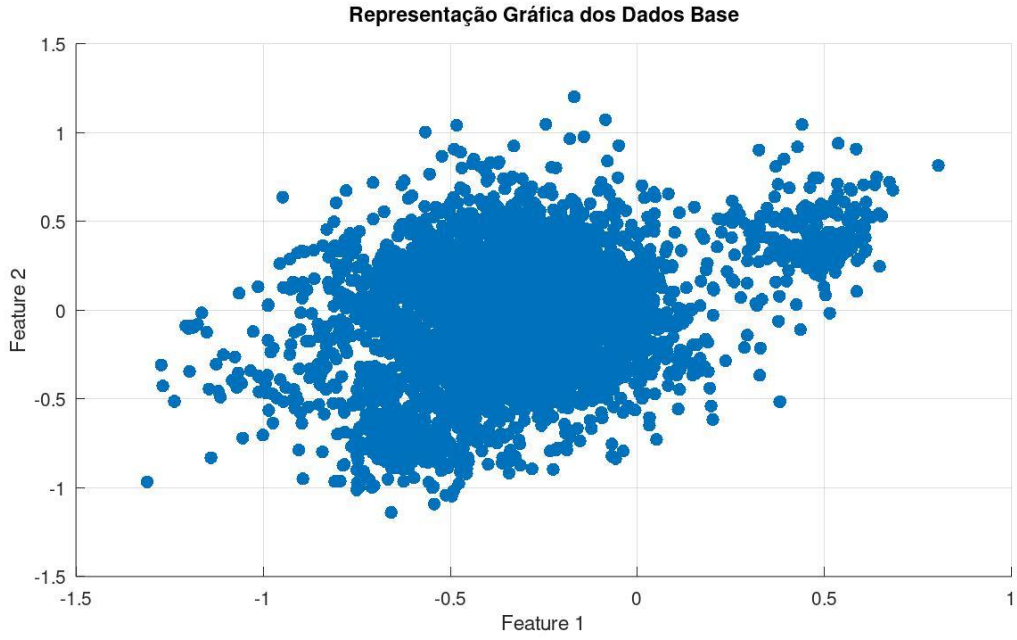
Este capítulo detalha as etapas realizadas para desenvolver a solução proposta, e inclui: visualização inicial dos dados, normalização, oversampling, redução de dimensionalidade, método de classificação, testes de performance e aplicação do modelo no conjunto de novas amostras.

Para implementar a solução, o software Octave foi utilizado, juntamente com o pacote *statistics*.

3.1 VISUALIZAÇÃO GRÁFICA DOS VETORES

O primeiro passo para entender a estrutura dos dados, foi realizar a visualização gráfica dos vetores. Nessa etapa, um gráfico 2D, que representa os dados projetos em dois eixos, foi gerado. É possível visualizar esse gráfico na FIGURA 1.

FIGURA 1: REPRESENTAÇÃO GRÁFICA DOS DADOS



FONTE: Os Autores (2024)

Essa análise teve os seguintes objetivos:

1º - Compreender a distribuição dos dados.

2º - Identificar a separação entre as classes.

A análise gráfica demostrou que há uma sobreposição entre algumas classes, tornando necessário o uso de técnicas para classificação.

3.2 NORMALIZAÇÃO

Os dados foram normalizados usando o método *z-score*, que transforma as variáveis para uma escala com média zero e desvio padrão unitário. Essa etapa é importante para evitar que variáveis com diferentes escalas influenciem desproporcionalmente o modelo. Sem essa normalização, variáveis com escalas maiores poderiam dominar o modelo, distorcendo os resultados e prejudicando a precisão das análises.

3.3 BALANCEAMENTO DE CLASSES – OVERSAMPLING

O problema de classes desbalanceadas ocorre quando o número de amostras em cada classe de um conjunto de dados é muito desigual. Esse desequilíbrio pode comprometer o desempenho de algoritmos de classificação, já que eles tendem a priorizar a classe majoritária, resultando em previsões enviesadas.

No cenário da classificação de bactérias, pode-se observar um desbalanceamento de classes, por exemplo: a classe Enterobacterales possui 2820 amostras, enquanto a Alteromonadales possuí apenas 128, isso mostra como o desbalanceamento é grande.

Para resolver esse problema, a técnica de *oversampling* foi utilizada. A técnica consiste em aumentar a quantidade de amostras das classes minoritárias até que todas as classes tenham uma representação proporcional no conjunto de dados.

Os passos realizados foram:

1º - Contar o número de amostras de cada classe.

2º - Identificar as classes minoritárias.

3º - Reequilibrar o conjunto de dados, duplicando as amostras.

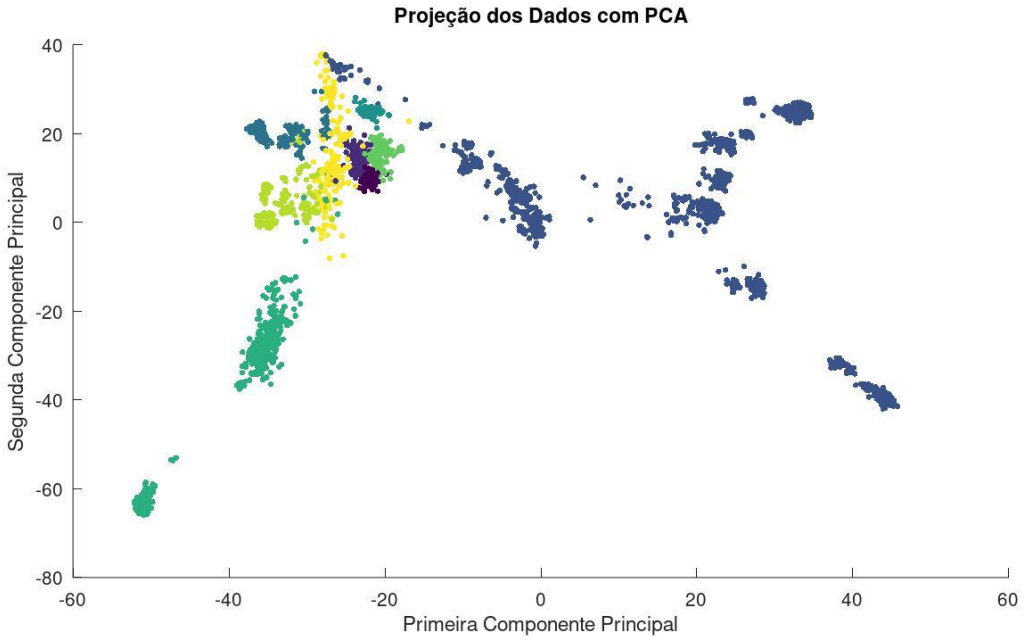
3.4 REDUÇÃO DE DIMENSIONALIDADE

Foi necessário realizar a redução de dimensionalidade devido à alta quantidade de variáveis presentes no conjunto, são 7500. A técnica aplicada para realizar essa redução chama-se Análise de Componentes Principais (PCA), está, é uma abordagem estatística que transforma os dados originais em um conjunto de variáveis não correlacionadas chamadas componentes principais.

Os passos realizados no PCA foram:

- **Centralização dos dados para média zero:** ajusta os valores subtraindo a média de cada variável, deixando os dados com média zero, para evitar distorções na análise das variações.
- **Cálculo da Matriz de Covariância:** Identificou como as variáveis se relacionam entre si.
- **Determinação dos Autovalores e Autovetores:** Identificou as direções principais que explicam a maior parte da variância nos dados.
- **Seleção das Principais Componentes:** Apenas os componentes com maior contribuição para a variância foram mantidos. O algoritmo foi ajustado para selecionar 2000 componentes.

FIGURA 2: ANÁLISE DE COMPONENENTES PRINCIAIS - PCA



FONTE: Os Autores (2024)

3.5 MÉTODO DE CLASSIFICAÇÃO

A classificação foi realizada utilizando uma rede neural Perceptron Multicamadas (MLP), que é um modelo de máquina capaz de aprender padrões complexos de dados. A MLP é composta por múltiplas camadas de neurônios interconectados, incluindo uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Cada neurônio nas camadas ocultas aplica uma função de ativação não linear, permitindo que a rede capture relações não lineares entre as variáveis dos dados. Esse modelo foi treinado para mapear as entradas para suas respectivas classes, ajustando seus pesos de forma a minimizar o erro de classificação

O algoritmo em questão foi configuração da seguinte forma:

Camadas:

- Uma camada de entrada, correspondente ao número de dimensões após o PCA.
- Uma camada oculta com 10 neurônios, configurada para capturar padrões não lineares.
- Uma camada de saída com o número de neurônios igual às classes, utilizando a função *softmax* para previsão de probabilidades.

Função de Ativação: A função *sigmoidal* foi usada na camada oculta, pois ela permite capturar relações não lineares.

Otimização: O treinamento foi feito usando o algoritmo de retropropagação, com uma taxa de aprendizado de 0,01 e um total de 1000 iterações (épocas).

3.6 TESTES DE PERFORMANCE

Foi empregado a validação cruzada k-Fold para avaliar o desempenho do modelo. A validação cruzada é uma técnica de avaliação de modelos de aprendizado de máquina usada para medir a performance de um modelo de forma mais robusta e confiável. Em vez de dividir o conjunto de dados apenas uma vez em treino e teste, essa técnica divide os dados em k subconjuntos (ou "folds") de forma aleatória. O processo envolve 3 partes:

1. **Divisão dos dados**: O conjunto de dados é dividido em k partes aproximadamente iguais. Cada uma dessas partes é chamada de fold.
2. **Treinamento e teste**: O modelo é treinado em k-1 folds, e o restante (o fold que não foi usado para o treinamento) é utilizado como conjunto de teste. Isso é repetido k vezes, com cada fold sendo utilizado uma vez como teste.
3. **Média dos resultados**: O desempenho do modelo é avaliado em cada um dos k testes, e o resultado final é a média dos scores obtidos nas k iterações. Isso ajuda a garantir que o modelo não esteja sobretreinado (overfitting) a um único conjunto de dados.

Para realizar os passos descritos, o algoritmo foi ajustado da seguinte forma: o conjunto de dados foi dividido em 3 subconjuntos (k=3), alternando entre dados de treinamento e teste. O F1-Score médio foi calculado para quantificar a eficácia do modelo.

3.7 APLICAÇÃO DO MODELO NO GRUPO DE TESTES

Os dados novos foram primeiramente normalizados utilizando o mesmo método (*z-score*) aplicado aos dados de treinamento. Em seguida, o modelo treinado foi utilizado para prever as classes das novas amostras. As classes previstas foram geradas com base na saída da rede neural, mapeando as probabilidades previstas para cada classe na classe mais provável.

4 ANÁLISE DOS RESULTADOS

1. Discuta a importância da divisão dos dados em conjuntos de treino, teste e validação neste contexto. Como podemos identificar eventos de overfitting e underfitting através deles?

A divisão dos dados em conjuntos de treino, teste e validação é crucial para garantir que o modelo aprenda de forma generalizável. Para este trabalho, a utilização da validação cruzada k-fold permite que o modelo seja treinado e testado em diferentes subconjuntos dos dados, garantindo uma avaliação mais robusta e imparcial. O conjunto de treino é utilizado para ajustar os parâmetros do modelo, enquanto o conjunto de teste serve para avaliar como o modelo generaliza para dados desconhecidos.

Overfitting: Quando o modelo tem um desempenho excelente no conjunto de treino, mas apresenta um desempenho ruim no conjunto de teste, isso é sinal de overfitting. O modelo "memoriza" os dados de treino, ao invés de aprender os padrões gerais. Esse fenômeno pode ser detectado quando a diferença entre a performance no treino e no teste é muito grande.

Underfitting: Se o modelo tem um desempenho ruim tanto no treino quanto no teste, isso indica underfitting. O modelo não consegue capturar os padrões subjacentes dos dados, e isso é visível quando a performance é baixa em ambos os conjuntos. A validação cruzada ajuda a identificar essas situações, pois avalia o modelo em diferentes divisões dos dados.

2. As classes dos dados (vetores) são separáveis entre si? Isso é importante? Utilize o gráfico para responder essa questão;

A separabilidade das classes é uma característica fundamental para o desempenho do modelo. Se as classes dos dados são separáveis no espaço de características, significa que existe uma clara distinção entre elas, o que facilita a classificação. No caso das amostras taxonômicas, o gráfico gerado pela projeção dos dados utilizando PCA mostra como as amostras são distribuídas no espaço das componentes principais.

O gráfico (FIGURA 2) apresenta a projeção dos dados em duas dimensões utilizando PCA, e é possível observar que as classes, representadas por diferentes cores, não são completamente separáveis entre si. Embora existam agrupamentos distintos para algumas classes, há uma significativa sobreposição entre elas, especialmente na região central.

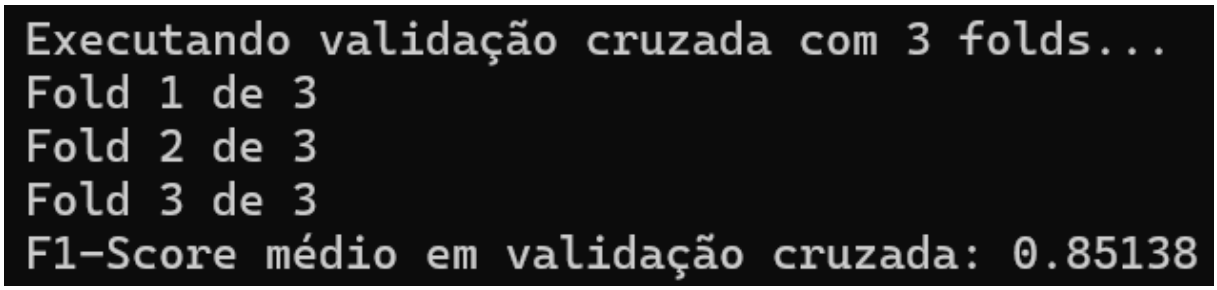
3. Pode-se afirmar que a classificação dos dados da amostra foi bem sucedida?

A avaliação do sucesso da classificação dos dados depende diretamente das métricas de desempenho utilizadas. No caso, o F1-score médio obtido pela validação cruzada é uma medida crucial para determinar a eficácia do modelo. O F1-score combina precisão, que é a capacidade de identificar corretamente as amostras positivas, e recall, que é a capacidade de detectar todas as amostras positivas, em uma única métrica.

Portanto, a análise do F1-score médio nos permite concluir se o modelo teve um desempenho satisfatório na classificação dos dados da amostra. Quanto mais próximo de 1 for o valor do F1-score, melhor foi o desempenho do modelo na identificação correta das classes.

Com o F1-score médio variando entre 0.82 e 0.93, pode-se dizer que o modelo apresentou um desempenho satisfatório na classificação dos dados da amostra, indicando que o modelo foi eficaz na identificação correta das classes.

FIGURA 3: EXEMPLO F1-SCORE



FONTE: Os Autores (2024)

REFERÊNCIAS

GUIA DE HOSPEDAGEM. O que é F1-Score. Disponível em: https://guiadehospedagem.com.br/glossario/o-que-e-f1-score/.

DATAGEEKS. Overfitting. Disponível em: https://www.datageeks.com.br/overfitting/.

DATAGEEKS. Análise de Componentes Principais. Disponível em: https://www.datageeks.com.br/analise-de-componentes-principais/.

DATACAMP. Multilayer Perceptrons in Machine Learning. Disponível em: https://www.datacamp.com/pt/tutorial/multilayer-perceptrons-in-machine-learning.

ESTATÍSTICA FÁCIL. O que é normalização Z-Score. Disponível em: https://estatisticafacil.org/glossario/o-que-e-normalizacao-z-score/.

MEDIUM. Aprenda a balancear seus dados com undersampling e oversampling em Python. Disponível em: https://medium.com/@daniele.santiago/aprenda-a-balancear-seus-dados-com-undersampling-e-oversampling-em-python-6fd87095d717.

ESCOLA DNC. Entendendo oversampling: técnicas e métodos para balanceamento de dados. Disponível em: https://www.escoladnc.com.br/blog/entendendo-oversampling-tecnicas-e-metodos-para-balanceamento-de-dados/.

OCTAVE. Disponível em: https://octave.org.

GNU OCTAVE. Pacote de Estatísticas. Disponível em: https://gnu-octave.github.io/packages/statistics/.

GNU OCTAVE. Estatísticas no Octave. Disponível em: https://gnu-octave.github.io/statistics/.

ANEXO 1 – CÓDIGO FONTE

```
pkg load statistics;

% ========================= Carregar Dados =========================
disp("Carregando os dados...");
% Alterar para o local onde estão salvos os dados
disp("Carregando os dados_base...");
dados_base = dlmread('C:/Users/ferna/Desktop/Inteligência Artificial 1/IA-trabalho/Dados_Base.csv', ',');
disp("Carregando as classes_base...");
classes_base = textread('C:/Users/ferna/Desktop/Inteligência Artificial 1/IA-trabalho/Classes_Base.csv', '%s', 'delimiter', ',');
disp("Carregando os dados_novos...");
dados_novos = dlmread('C:/Users/ferna/Desktop/Inteligência Artificial 1/IA-trabalho/Dados_Novos.csv', ',');
disp("Concluído!");

% ========================= Representação Gráfica Dados-Base =========================
disp("Plotando os dados base...");
figure;
scatter(dados_base(:, 1), dados_base(:, 2), 50, 'filled');
title("Representação Gráfica dos Dados Base");
xlabel("Feature 1");
ylabel("Feature 2");
grid on;



% ========================= Normalização =========================
disp("Normalizando os dados_base...");
dados_base_norm = zscore(dados_base);
```

```
disp("Concluído!");

disp("Normalizando os dados_novos...");
dados_novos_norm = zscore(dados_novos);
disp("Concluído!");



% ======================== Transformação das Classes para Formato Binário ========================
[classes_unicas, ~, classes_numericas] = unique(classes_base);
num_classes = length(classes_unicas);
num_amostras = length(classes_base);
classes_binarias = eye(num_classes)(classes_numericas, :);

% ======================== Oversampling ========================
disp("Oversampling Iniciado..");
contagem_classes = hist(classes_numericas, num_classes);
min_class_count = min(contagem_classes);
min_classes_idx = find(contagem_classes == min_class_count);
dados_base_balanciado = dados_base_norm;
classes_base_balanciado = classes_numericas;

for i = 1:length(min_classes_idx)
   min_class_idx = min_classes_idx(i);

   amostras_minoritaria = dados_base_norm(classes_numericas == min_class_idx, :);

   num_amostras_faltando = max(contagem_classes) - min_class_count;

   num_amostras_faltando = min(num_amostras_faltando, size(amostras_minoritaria, 1));
```

```
    amostras_duplicadas = amostras_minoritaria(randperm(size(amostras_minoritaria, 1), num_amostras_faltando), :);

    dados_base_balanciado = [dados_base_balanciado; amostras_duplicadas];
    classes_base_balanciado = [classes_base_balanciado; repmat(min_class_idx, num_amostras_faltando, 1)];
end

disp("Oversampling concluído para todas as classes minoritárias!");


% ========================= PCA - Dados-Base =========================
disp("Realizando PCA...");
% Centralizar os dados
dados_base_centralizados = dados_base_norm - mean(dados_base_norm);
% Calcular a matriz de covariância
covariancia = cov(dados_base_centralizados);
% Calcular os autovalores e autovetores
[autovetores, autovalores] = eig(covariancia);
% Ordenar os autovalores e selecionar as componentes principais
[autovalores, indice] = sort(diag(autovalores), 'descend');
autovetores = autovetores(:, indice);
pca = 2000;  % número de componentes principais desejadas
dados_reduzidos = dados_base_centralizados * autovetores(:, 1:pca);
disp("PCA concluído!");


% ========================= Visualizar os dados projetados =========================
figure;
scatter(dados_reduzidos(:, 1), dados_reduzidos(:, 2), 10, classes_numericas,
```

```
'filled');
title('Projeção dos Dados com PCA');
xlabel('Primeira Componente Principal');
ylabel('Segunda Componente Principal');



% ========================= Validação Cruzada k-Fold =========================
k = 3;  % Número de folds
disp(['Executando validação cruzada com ', num2str(k), ' folds...']);

% Gerar índices manualmente
indices = repmat(1:k, 1, ceil(num_amostras / k));
indices = indices(1:num_amostras);
indices = indices(randperm(num_amostras));  % Embaralhar os índices

f1_scores_fold = zeros(1, k);

for fold = 1:k
    disp(['Fold ', num2str(fold), ' de ', num2str(k)]);

    % Divisão dos dados em treinamento e teste
    teste_idx = (indices == fold);
    treino_idx = ~teste_idx;

    dados_treinamento = dados_base_balanciado(treino_idx, :);
    classes_treinamento = classes_base_balanciado(treino_idx);

    dados_teste = dados_base_balanciado(teste_idx, :);
    classes_teste = classes_base_balanciado(teste_idx);

    % Inicializar rede neural (MLP)
    num_neuronios_ocultos = 10;
```

```
num_epocas = 1000;
taxa_aprendizado = 0.01;

tamanho_entrada = size(dados_treinamento, 2);
tamanho_saida = num_classes;
pesos_entrada_oculta = rand(tamanho_entrada, num_neuronios_ocultos) * 0.01;
bias_oculta = zeros(1, num_neuronios_ocultos);
pesos_oculta_saida = rand(num_neuronios_ocultos, tamanho_saida) * 0.01;
bias_saida = zeros(1, tamanho_saida);

sigmoid = @(x) 1 ./ (1 + exp(-x));
sigmoid_derivada = @(x) sigmoid(x) .* (1 - sigmoid(x));

% Treinamento
for epoca = 1:num_epocas
    camada_oculta = sigmoid(dados_treinamento * pesos_entrada_oculta + bias_oculta);
    camada_saida = sigmoid(camada_oculta * pesos_oculta_saida + bias_saida);

    erro = camada_saida - eye(num_classes)(classes_treinamento, :);

    delta_saida = erro .* sigmoid_derivada(camada_saida);
    delta_oculta = (delta_saida * pesos_oculta_saida') .* sigmoid_derivada(camada_oculta);

    pesos_oculta_saida -= taxa_aprendizado * (camada_oculta' * delta_saida);
    bias_saida -= taxa_aprendizado * sum(delta_saida, 1);

    pesos_entrada_oculta -= taxa_aprendizado * (dados_treinamento' * delta_oculta);
    bias_oculta -= taxa_aprendizado * sum(delta_oculta, 1);
end
```

```
    % Validação
    camada_oculta_teste = sigmoid(dados_teste * pesos_entrada_oculta + bias_oculta);
    camada_saida_teste = sigmoid(camada_oculta_teste * pesos_oculta_saida + bias_saida);

    [~, classe_prevista] = max(camada_saida_teste, [], 2);

    % Cálculo do F1-Score
    f1_scores = zeros(1, num_classes);
    for i = 1:num_classes
        classes_binarias_reais = (classes_teste == i);
        classes_binarias_previstas = (classe_prevista == i);

        precisao = sum(classes_binarias_reais & classes_binarias_previstas) / (sum(classes_binarias_previstas) + eps);
        recall = sum(classes_binarias_reais & classes_binarias_previstas) / (sum(classes_binarias_reais) + eps);

        f1_scores(i) = 2 * (precisao * recall) / (precisao + recall + eps);
    end

    f1_scores_fold(fold) = mean(f1_scores);

end

% Resultado final
f1_score_medio = mean(f1_scores_fold);
disp(['F1-Score médio em validação cruzada: ', num2str(f1_score_medio)]);



% ========================= Classificação dos Dados Novos =========================
```

```
disp("Classificando os dados novos...");

% Forward pass para os dados novos
camada_oculta_novos = sigmoid(dados_novos_norm * pesos_entrada_oculta + bias_oculta);
camada_saida_novos = sigmoid(camada_oculta_novos * pesos_oculta_saida + bias_saida);

[~, classes_previstas_novos] = max(camada_saida_novos, [], 2);

disp("Classificação concluída!");
disp("Classes previstas para os dados novos:");
disp(classes_previstas_novos);
```