Relatório - Atividade Prática 3
PDI - SVM

João Belfort[1], Miguel Ribeiro[1]
Instituto de Ciências Exatas e Tecnológicas
Universidade Federal de Viçosa- Campus Florestal (UFV-Florestal)
Rodovia LMG 818- km 6- Florestal- MG - Brasil
CEP: 35690-000

{joao.andrade1, miguel.a.silva}@ufv.br

1. Introdução

Este artigo aborda a terceira atividade de relatório, concentrando-se nas técnicas de **Support Vector Machine** (SVM) e **Multi-Layer Perceptron** (MLP). Inicialmente, a imagem é processada para transformá-la em um conjunto de dados que representa a distribuição dos pixels em um espaço tridimensional. Em seguida, uma visualização 3D é realizada para examinar a distribuição desse conjunto de dados. Utilizando o método do cotovelo (Elbow Method), é determinado o número ótimo de classes para o conjunto de dados, utilizando a biblioteca kneed e aplicando o algoritmo de agrupamento K-means.

Posteriormente, são realizadas as etapas 4 e 5, que envolvem a criação de um novo conjunto de dados com pelo menos 30 amostras de cada uma das sete classes (seis halteres e um fundo). Após isso, uma visualização 3D é realizada no novo conjunto de dados para observar sua separabilidade.

Para a classificação da imagem, são empregados os algoritmos de máquina de vetores de suporte (SVM) e uma rede neural MLP (Multi-Layer Perceptron). O desempenho desses algoritmos é avaliado em termos de precisão de treinamento e as matrizes de confusão resultantes são reportadas.

Além disso, é realizado um ajuste de hiperparâmetros utilizando a técnica de grid search, considerando diferentes configurações de camadas e taxas de aprendizado. Os parâmetros encontrados, juntamente com as respectivas precisões de treinamento e matrizes de confusão, são apresentados. Por fim, o uso do SVM com o conjunto de dados CIFAR-10 é explorado para fins de comparação e análise adicionais.

2. Distribuição dos pixels, método Kneed e K Means

2.1. Distribuição dos pixels

Para iniciar a análise, a imagem fornecida ("**halteres.jpg**") (Figura 01) foi carregada e convertida em uma matriz utilizando a biblioteca Python, especificamente a função Image.open do módulo **PIL**, seguida pela conversão dessa imagem em um array **NumPy**. A forma (shape) dessa matriz foi então impressa para verificar as dimensões do dataset resultante. (Figura 02)

```
imagem = Image.open("halteres.jpg")
imagem_array = np.array(imagem)
print("Shape da imagem array:", imagem_array.shape)
```

Figura 01 - Conversão da imagem

Figura 02 - 'halteres.py'

O resultado dessa etapa mostrou que a imagem foi corretamente convertida em um dataset representado por um array tridimensional, onde as dimensões correspondem à altura, largura e canais de cor da imagem (Shape da imagem array: (225, 300, 3).

A fim de visualizar a distribuição dos pixels da imagem no espaço tridimensional RGB (Red, Green, Blue), uma plotagem tridimensional foi criada utilizando a biblioteca **matplotlib**. Para isso, os valores de intensidade de cada canal de cor (R, G, B) foram extraídos do array da imagem e achatados (flatten) para formar listas de coordenadas x, y e z (Figura 03)

.

```
fig = plt.figure(figsize=(5,5))
ax = fig.add_subplot(111, projection='3d')

r = imagem_array[:, :, 0].flatten()
g = imagem_array[:, :, 1].flatten()
b = imagem_array[:, :, 2].flatten()

ax.scatter(r, g, b, c='b', marker='.')
ax.set_xlabel('Red')
ax.set_ylabel('Green')
ax.set_zlabel('Blue')
plt.show()
```

Figura 03 - Código para visualizar a distribuição dos Pixels

O gráfico resultante mostra uma dispersão tridimensional dos pixels da imagem, onde cada ponto representa um pixel com suas coordenadas de intensidade de vermelho, verde e azul (x,y,z) (Figura 04).

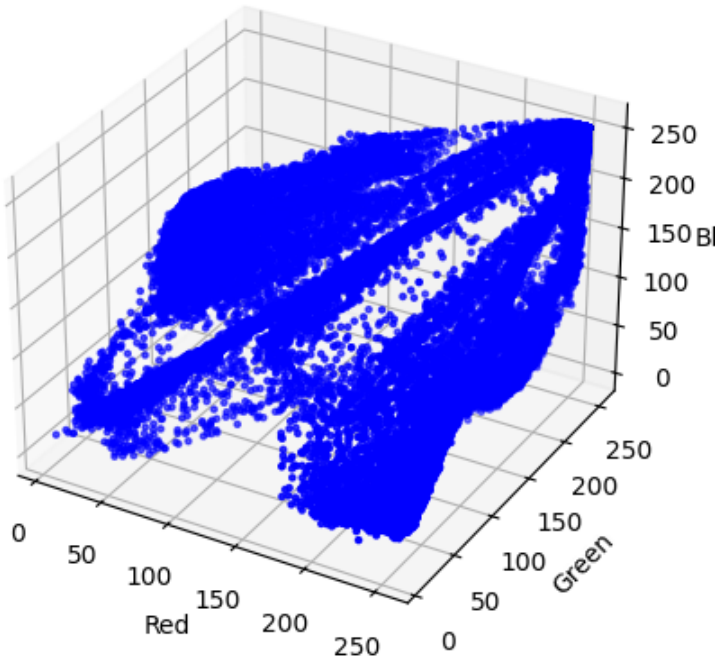

Figura 04 - Distribuição dos pixels de 'halteres.py'

2.2. Kmeans

Nesta etapa, foi realizado o método de Elbow (E) para determinar o número ideal de classes para o dataset criado, seguido da aplicação do algoritmo K-Means para agrupar os dados em clusters com base nesse número ótimo de classes.

1. **Implementação do método de Elbow:**

Foi definida uma função chamada `**find_optimal_k(data)**` que calcula as distorções para diferentes números de clusters (de 1 a 10) utilizando o algoritmo **K-Means**. Essas distorções são então utilizadas para identificar o ponto de inflexão que representa o número ótimo de clusters, utilizando a biblioteca ´**knee**´ (Figura 05).

```
def find_optimal_k(data):
   distortions = []
   for i in range(1, 11):
       kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
       kmeans.fit(data)
       distortions.append(kmeans.inertia_)
       kn = KneeLocator(range(1, 11), distortions, curve='convex',
direction='decreasing')
   return kn.elbow
```

Figura 05 - Função `*KneeLocator()*`

2. **Implementação do k-Means:**

Foi criada uma função denominada `**cluster_and_show_image(image_path, k_clusters)**` que carrega a imagem, remodela o array da imagem para o formato necessário para o K-Means e executa o algoritmo K-Means com o número de clusters determinado pelo método de Elbow. A imagem original é então comprimida com base nos clusters encontrados e uma visualização das imagens original e comprimida é apresentada.

O número ótimo de clusters foi determinado como sendo **k = 3**. Em seguida, o algoritmo K-Means foi aplicado para agrupar os pixels da imagem em três clusters distintos. A imagem original e a imagem comprimida, mostrando a redução na quantidade de cores, foram exibidas para visualização e análise (Figura 06).

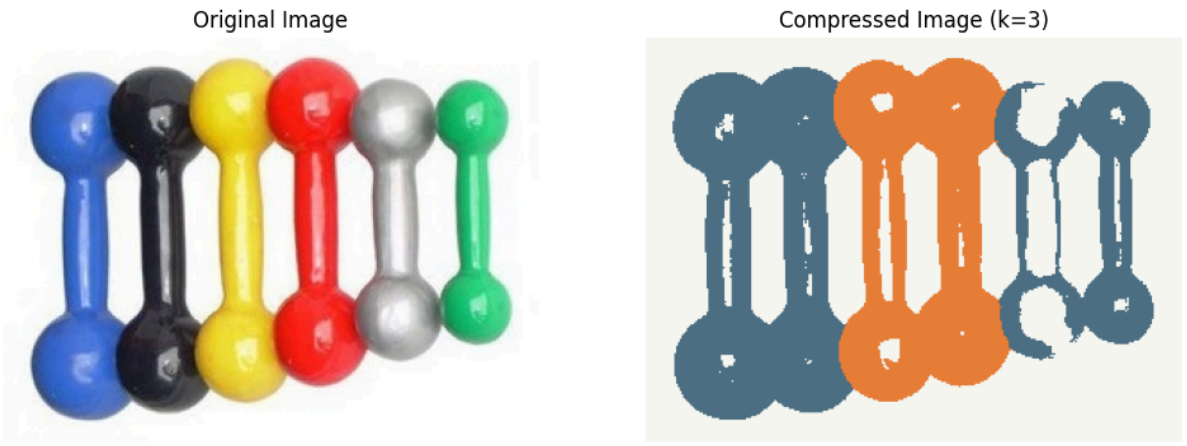

Figura 06 - K-Means

2.3. Criação de um dataset

Na sequência foi realizado o processo de coleta de amostras para a criação de um dataset contendo no mínimo **30 amostras** de cada uma das **7 classes**, sendo 6 classes representando diferentes halteres e uma classe representando o fundo.

Foram definidas as classes, cada uma representando um tipo diferente de haltere e o fundo da imagem. A função de callback `**on_mouse**` foi utilizada para capturar os eventos de clique do mouse, onde os valores dos pixels correspondentes às coordenadas clicadas foram armazenados em um array. Esses valores de pixel, juntamente com a classe da amostra, foram então registrados em um arquivo CSV (Figura 07).

O código foi executado com sucesso, coletando um mínimo de 30 amostras de cada uma das 7 classes. Durante a execução do código, foram exibidos os nomes das classes e o progresso da coleta de amostras. Ao final, todas as amostras foram salvas em um arquivo CSV para uso posterior (Figura 08).

```
Amostras classe haltere6-Verde
Amostra 0: 105,180,18
Amostra 1: 105,181,19
Amostra 2: 107,185,17
Amostra 3: 121,187,38
Amostra 4: 192,247,130
Amostra 5: 129,211,44
```

Figura 07 - Coleta de amostras

```
0,191,100,49
0,196,102,59
0,194,104,63
0,230,173,152
0,209,125,100
0,182,96,54
0,238,156,114
0,247,168,141
0,228,144,109
0,218,137,102
```

Figura 08 - Amostras salvas no arquivo *dataset.csv*

2.4. Visualização do dataset

Na quarta atividade, foi realizada a visualização do dataset em um espaço tridimensional (R, G, B) para observar a separabilidade das classes. Cada amostra foi representada como um ponto no espaço 3D, onde as coordenadas correspondem aos valores dos canais de cor RGB.

Primeiramente, os dados foram carregados a partir do arquivo CSV gerado na atividade anterior, utilizando a biblioteca Pandas. As classes e as cores associadas a cada classe foram definidas (Figura 09). Em seguida, foi criado um gráfico tridimensional usando a biblioteca matplotlib, onde cada classe foi representada por uma cor distinta.

O gráfico resultante mostra a distribuição das amostras no espaço tridimensional RGB. Cada classe é representada por pontos de uma cor específica, facilitando a observação da separabilidade das classes. Esta visualização fornece insights importantes sobre a estrutura dos dados e é fundamental para avaliar a adequação dos modelos de classificação (Figura 10).

	Classe	Blue	Green	Red
0	0	191	100	49
1	0	196	102	59
2	0	194	104	63
3	0	230	173	152
4	0	209	125	100

Figura 09 - Classes e cores BGR

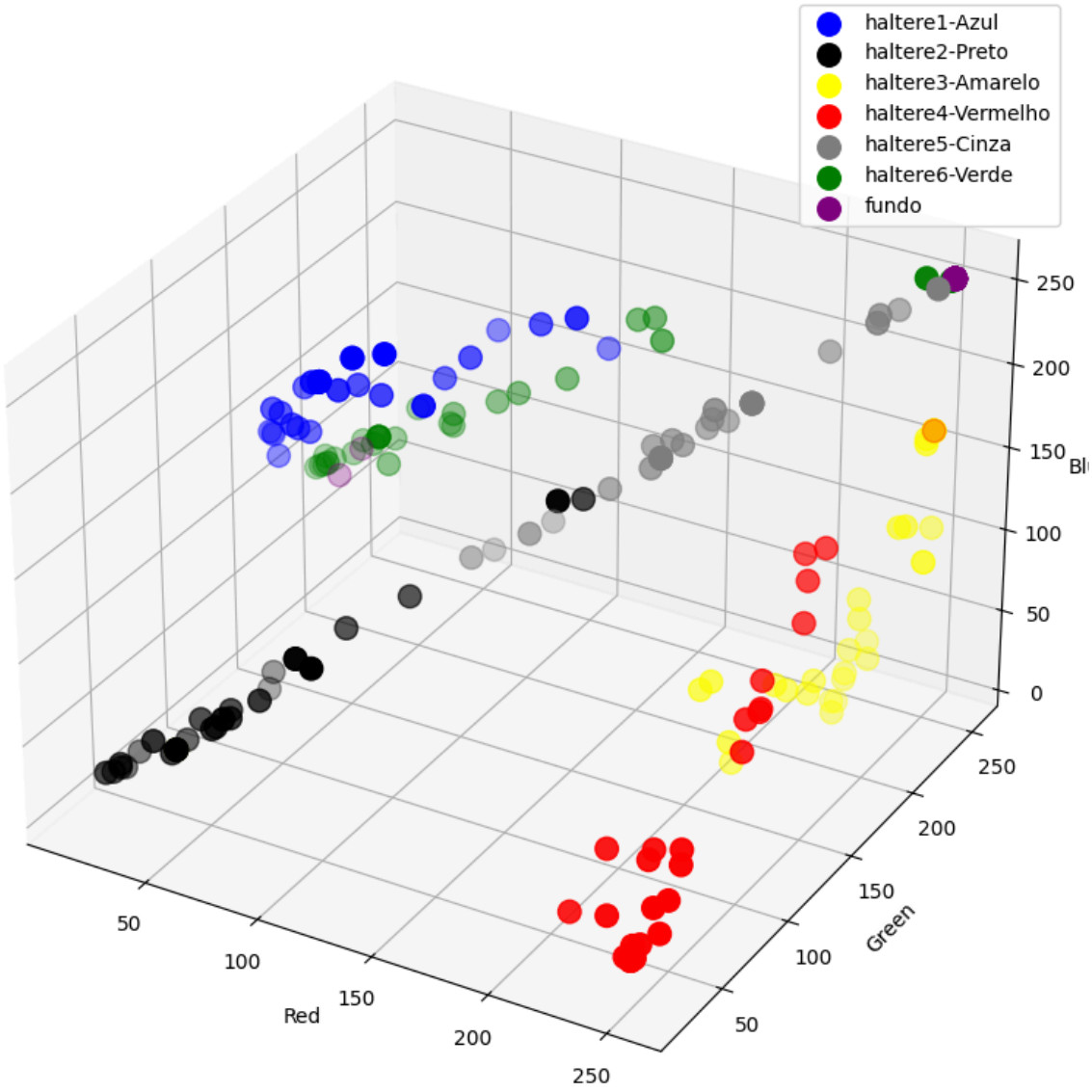

Figura 10 - Visualização do dataset

2.5. SVM e MLP

Nesta etapa, a imagem foi classificada utilizando dois modelos de aprendizado de máquina: **Support Vector Machine** (**SVM**) e **Multi-Layer Perceptron** (**MLP**). Os modelos foram treinados e testados com um conjunto de dados contendo 50 amostras de cada uma das 7 classes, com uma divisão de 80% para treinamento e 20% para teste.

1. **SVM (Support Vector Machine)**

O SVM é um algoritmo que busca uma linha de separação entre duas classes distintas, analisando os dois pontos, um de cada grupo, mais próximos da outra classe.

Isto é, o SVM escolhe a reta — também chamada de hiperplano em maiores dimensões— entre dois grupos que se distanciam mais de cada um (no caso abaixo, a reta vermelha) (Figura 11).

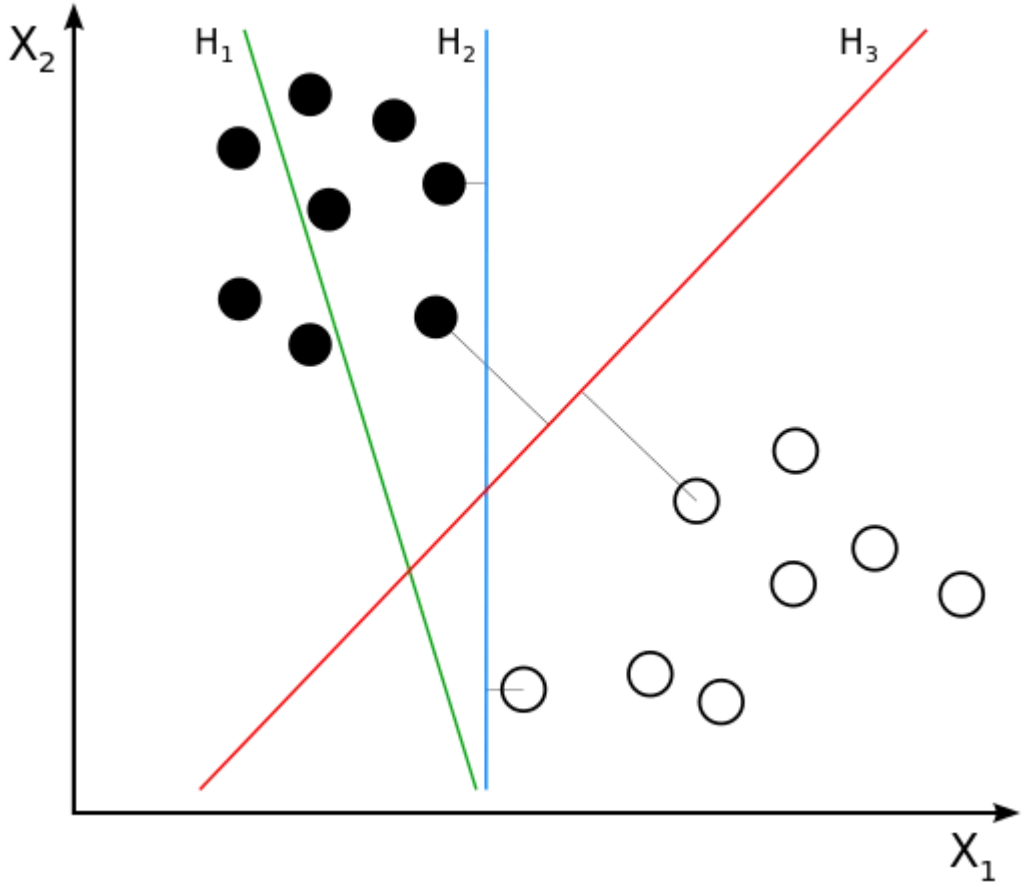

Figura 11 - A reta ótima (H3) é a mais distante dos dois grupos, considerando apenas os pontos de cada grupo mais próximos à reta (como indicado pelas linhas cinzas. Fonte: Wikimedia Commons. [1]

Após descoberta essa reta, o programa conseguirá predizer a qual classe pertence um novo dado ao checar de qual lado da reta ele está. [2]

Para o nosso modelo, os relatórios de classificação foram:

- **Acurácia**: 0.97
- **F1-Score**: 0.97

A alta acurácia do SVM pode ser atribuída a uma boa separabilidade entre as classes no espaço de características, uma adequada escolha de parâmetros do modelo, um conjunto de dados representativo e balanceado, e uma divisão apropriada entre dados de treinamento e teste.

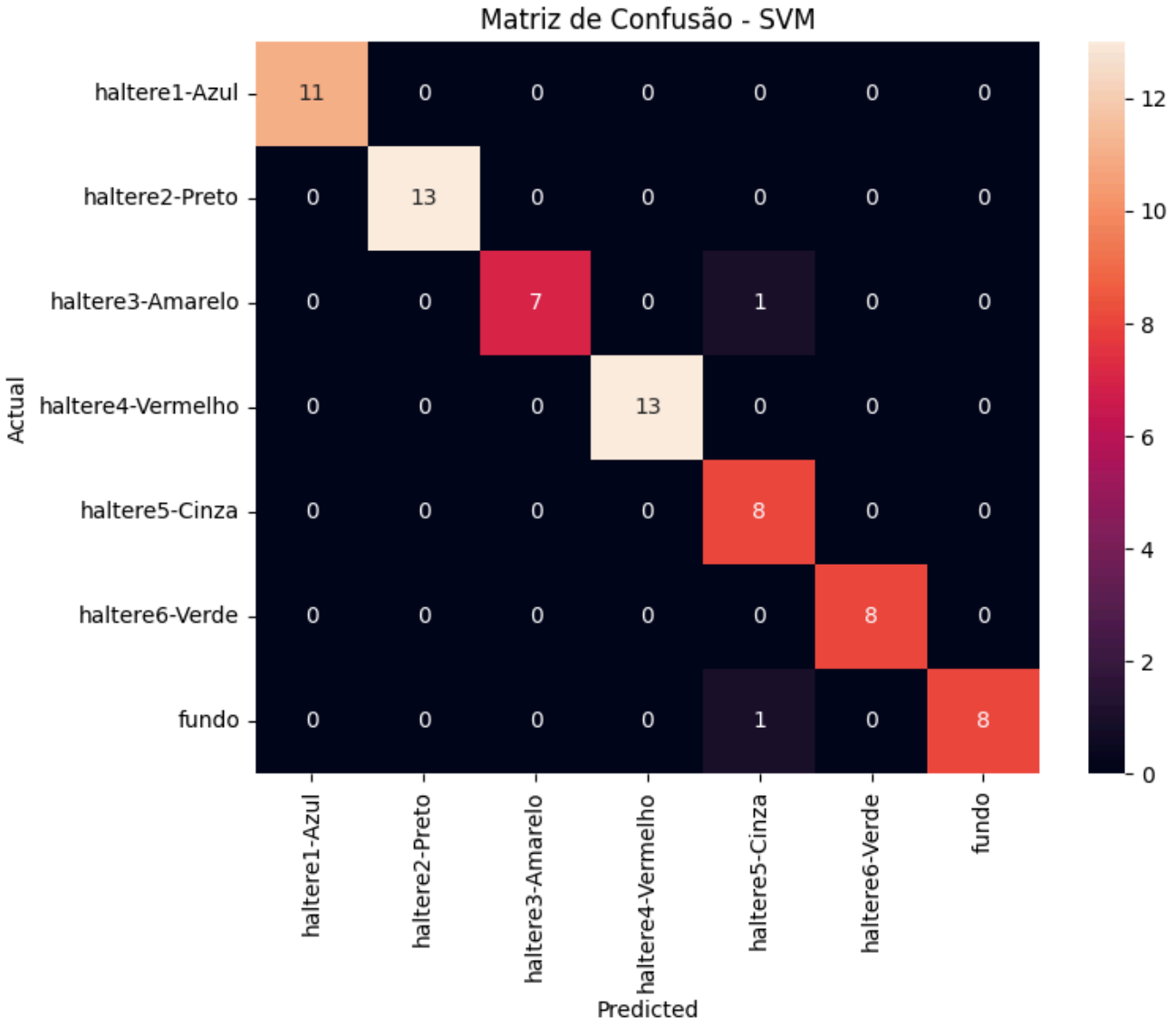

Figura 12 - Matriz de confusão - SVM

2. Multi-Layer Perceptron (MLP)

Perceptron Multicamadas (PMC ou MLP — Multi Layer Perceptron) é uma rede neural com uma ou mais camadas ocultas com um número indeterminado de neurônios. A camada oculta possui esse nome porque não é possível prever a saída desejada nas camadas intermediárias. [3]

Nosso modelo MLP foi treinado com duas camadas ocultas de 100 neurônios cada, utilizando os dados de treinamento. As previsões foram feitas no conjunto de teste e a precisão do treinamento, o F1-Score e a matriz de confusão foram calculados e exibidos.

- Acurácia: 0.84
- F1-Score: 0.81

Uma acurácia de 0.84 e 0.81 para o MLP é considerada boa, embora um pouco menor em comparação com a alta acurácia do SVM. Essas taxas podem ser devidas a complexidades adicionais nos dados que o MLP pode ter dificuldade em capturar, ou pode ser devido a uma configuração subótima do modelo. Também pode ser um reflexo da natureza mais complexa do MLP em comparação com o SVM, exigindo um ajuste mais cuidadoso dos hiperparâmetros e arquitetura da rede neural para alcançar altas taxas de acerto. Observe que a classe **fundo**, não foi classificada corretamente.

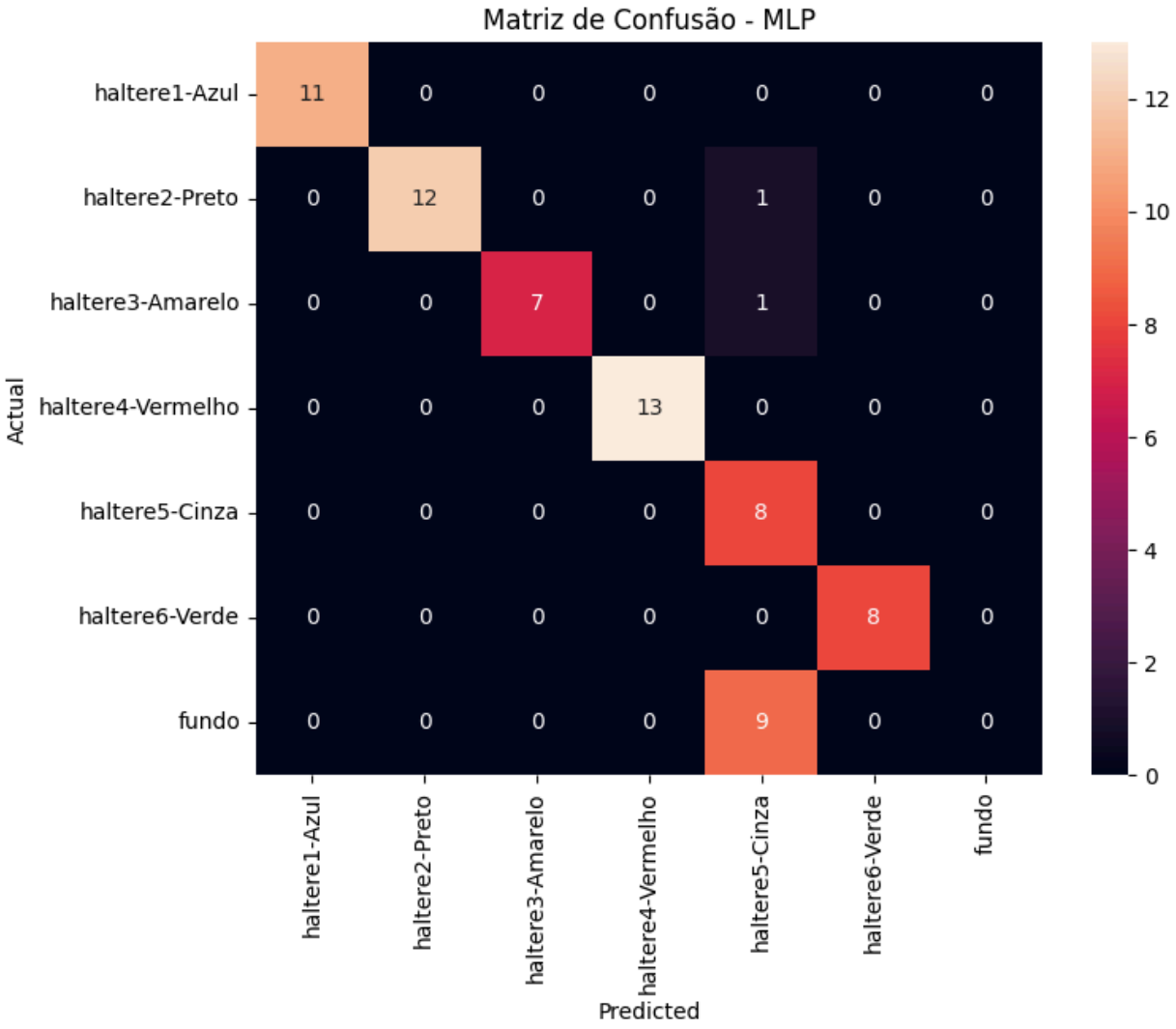

Figura 13 - Matriz de confusão - MLP

2.6. GridSearchCV no MLP

Nesta etapa, foi utilizado o GridSearchCV, uma ferramenta essencial em machine learning, para encontrar os melhores hiperparâmetros para o modelo Multi-Layer Perceptron (MLP). O objetivo é otimizar o desempenho do modelo ao explorar diferentes combinações de hiperparâmetros.

O **GridSearchCV** permite definir um espaço de busca para os hiperparâmetros do modelo. Para este experimento, foram consideradas cinco opções de configurações de camadas ocultas [(50,), (100,), (200,), (50, 50), (100, 100)] e duas opções de taxas de aprendizado [0.001, 0.01] (Figura 14). O GridSearchCV então realiza uma busca exaustiva em todas as combinações desses hiperparâmetros, utilizando validação cruzada para avaliar o desempenho de cada configuração [4].

```
param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (200,), (50, 50), (100, 100)],
    'learning_rate_init': [0.001, 0.01]
}

mlp = MLPClassifier(max_iter=1000)
grid_search = GridSearchCV(mlp, param_grid, cv=3, scoring='accuracy')
```

Figura 14 - Parâmetros utilizados

1. **Implementação do GridSearchCV**

O classificador Multi-Layer Perceptron (MLP) foi inicializado com um número máximo de iterações de 1000 para permitir uma convergência adequada durante o treinamento. Em seguida, foi configurado o objeto GridSearchCV, utilizando o classificador MLP, o espaço de hiperparâmetros definido e especificando o número de dobras de validação cruzada como 3 (cv=3). Posteriormente, a busca de grade foi executada nos dados de treinamento, explorando todas as combinações possíveis de hiperparâmetros para encontrar a configuração ótima que maximiza o desempenho do modelo.

2. **Resultados**

- **Melhores parâmetros encontrados:**

{'hidden_layer_sizes': (100,), 'learning_rate_init': 0.001}

- **Precisão do treinamento:** 0.9107374361320827

- **Matriz de Confusão**

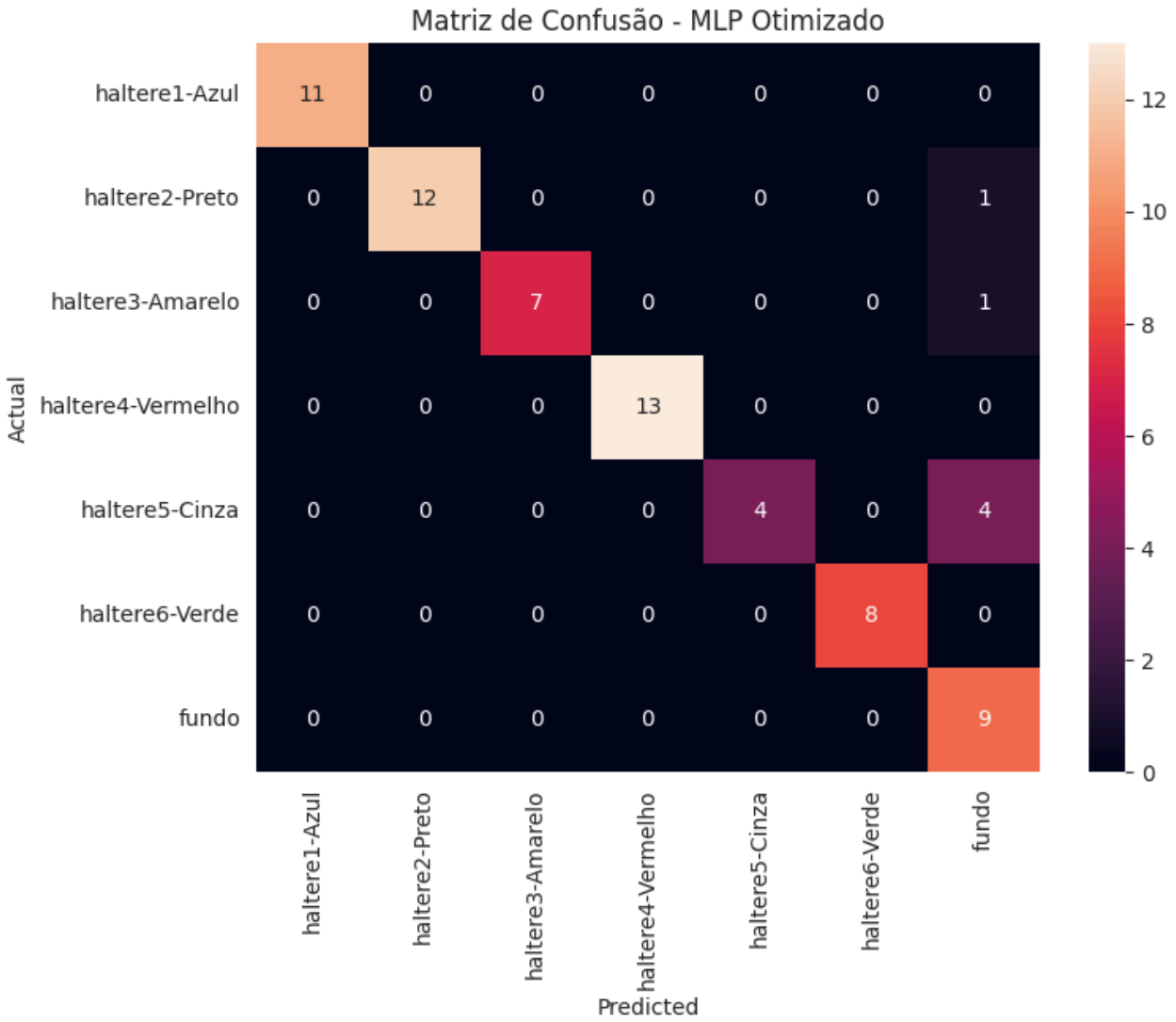

Figura 15 - Matriz de confusão - MLP Otimizado

.

Após ajustes nos hiperparâmetros e treinamento adicional, observamos uma melhoria significativa no desempenho do nosso modelo de rede neural, com uma acurácia aumentando de 0.84 para 0.91.

Embora essa melhoria seja promissora, é importante notar que, mesmo com aprimoramentos, o desempenho ainda não superou o modelo SVM em termos de identificação eficaz do conjunto de dados. O SVM parece ter uma vantagem em distinguir entre os exemplos de **branco (fundo)** e **cinza**, possivelmente devido à natureza linearmente separável dessas classes no espaço de características utilizado.

3. Referências bibliográficas

[1] File:Svm separating hyperplanes (SVG).svg - Wikimedia Commons. Disponível em: <https://commons.wikimedia.org/wiki/File:Svm_separating_hyperplanes_(SVG).svg>.
[2] COUTINHO, B. Modelos de Predição | SVM. Disponível em: <https://medium.com/turing-talks/turing-talks-12-classifica%C3%A7%C3%A3o-por-svm-f4598094a3f1>.
[3] MOREIRA, S. Rede Neural Perceptron Multicamadas. Disponível em: <https://medium.com/ensina-ai/rede-neural-perceptron-multicamadas-f9de8471f1a9>.
[4] Otimização com Grid Search. Disponível em: <https://pt.linkedin.com/pulse/otimiza%C3%A7%C3%A3o-com-grid-search-gabriel-constantin#:~:text=O%20processo%20GridSearchCV%20constr%C3%B3i%20e>. Acesso em: 6 abr. 2024.