Classificação de Imagens com KNN

GAC105 Programação Paralela e Concorrente

Profª. Marluce Rodrigues Pereira

Grupo:

Lucas Carvalho Ferreira

Thallys Henrique Martins

Yasmim Danzieri Abbondanza Laurentino

Lavras, 2025

Descrição do Problema do Mundo Real

A **classificação de imagens** é uma tarefa essencial em diversas áreas da computação, incluindo reconhecimento de padrões, diagnóstico médico, segurança e automação. Um dos desafios clássicos nessa área é a identificação de dígitos manuscritos, que possui aplicações em sistemas de reconhecimento de caracteres, como leitura automática de cheques, preenchimento de formulários e digitalização de documentos escritos à mão.

Para resolver esse problema, utilizaremos a base de dados MNIST (Modified National Institute of Standards and Technology), que contém 60.000 imagens de treinamento e 10.000 imagens de teste, cada uma representando um dígito manuscrito de 0 a 9. As imagens são em tons de cinza e têm dimensões de 28x28 pixels.

A classificação de imagens é um problema desafiador devido à grande quantidade de dados envolvidos e à complexidade das características que precisam ser extraídas para uma classificação precisa. Além disso, a escalabilidade é uma preocupação, especialmente quando se lida com grandes conjuntos de dados, como em aplicações de big data.

Descrição do Algoritmo KNN (K-Nearest Neighbors)

O algoritmo escolhido para resolver o problema de classificação de imagens é o **K-Nearest Neighbors (KNN)**, que é um método de aprendizado supervisionado simples, mas eficaz, para classificação e regressão. O KNN é um algoritmo de instância baseado em exemplos, o que significa que ele não aprende explicitamente um modelo durante o treinamento, mas sim armazena todo o conjunto de dados de treinamento e faz previsões com base na similaridade entre os dados de entrada e os exemplos armazenados.

Funcionamento do KNN:

1. Armazenamento dos Dados de Treinamento: O KNN armazena todos os exemplos de treinamento (imagens e seus rótulos correspondentes) em um espaço multidimensional, onde cada dimensão representa uma característica da imagem (por exemplo, intensidade de pixels, textura, etc.).

2. Cálculo de Distância: Quando uma nova imagem (dado de amostra) é fornecida, o algoritmo calcula a distância entre essa imagem e todas as imagens no conjunto de treinamento. A distância pode ser calculada usando métricas como a distância Euclidiana, Manhattan, ou outras métricas de similaridade. Neste trabalho foi usada a distância Euclidiana.

3. Seleção dos K Vizinhos Mais Próximos: O algoritmo seleciona os K exemplos de treinamento que estão mais próximos da nova imagem, com base nas distâncias calculadas.

4. Votação: Para classificação, o algoritmo realiza uma votação entre os K vizinhos mais próximos. A classe mais frequente entre esses K vizinhos é atribuída à nova imagem. Para regressão, a média dos valores dos K vizinhos é calculada.

5. Resultado: A nova imagem é classificada com base na votação ou na média dos vizinhos mais próximos.

Relatório Técnico

O objetivo do código é classificar imagens do conjunto MNIST utilizando um classificador KNN com paralelização multithreading para aumentar a eficiência computacional. A implementação busca reduzir o tempo de execução ao distribuir a carga de trabalho entre múltiplos núcleos de processamento.

Para a resolução deste trabalho foi utilizada a linguagem de programação Python, junto das bibliotecas numpy, sklearn, joblib e matplotlib.

Metodologia

1. **Carregamento do Conjunto de Dados:** Definimos uma função chamada **carregar_mnist()** para realizar o carregamento do conjunto de dados utilizando a função fetch_openml da biblioteca sklearn.datasets. 'mnist_784' trata-se do dataset MNIST, enquanto que as_frame indica que os dados não devem ser carregados como dataframes e sim como array numpy. A função retorna, através da utilização do train_test_split, também da biblioteca sklearn, a divisão dos dados de treinamento e testes, onde 10% dos dados ficam definidos como teste e os outros 90% como treinamento.

```
def carregar_mnist():
    mnist = fetch_openml('mnist_784', version=1, as_frame=False)
    caracteristicas, rotulos = mnist.data.astype(np.float32), mnist.target.astype(int)
    return train_test_split(caracteristicas, rotulos, test_size=0.1, random_state=42)
```

2. **Cálculo da Classificação Usando KNN:** A função **classificar_amostra()** calcula a distância Euclidiana entre uma amostra de teste e todas as amostras de treinamento utilizando np.linalg.norm da biblioteca numpy. Os vizinhos mais próximos são identificados e a classe mais frequente entre eles é atribuída como previsão.

```
def classificar_amostra(teste_amostra, treinar_caract, treinar_rotulos, vizinhos):
    distancia = np.linalg.norm(treinar_caract - teste_amostra, axis=1)
    k_vizinhos = np.argsort(distancia)[:vizinhos]
    k_rotulos = treinar_rotulos[k_vizinhos]
    return np.bincount(k_rotulos).argmax()
```

3. **Paralelização do Algoritmo:** A função **knn_classify** distribui a execução do método **classificar_amostra** em múltiplas threads para reduzir o tempo de execução. Para essa resolução foi utilizada a função Parallel da biblioteca joblib, indicando como preferência a opção de "threads" devido a possibilidade de compartilhamento de memória entre as threads.

```
def knn_classify(treinar_caract, treinar_rotulos, teste_caract, vizinhos=3, n_jobs=-1):
    return np.array(Parallel(n_jobs=n_jobs, prefer="threads")(
        delayed(classificar_amostra)(teste_amostra, treinar_caract, treinar_rotulos, vizinhos) for teste_amostra in teste_caract
    ))
```

4. **Avaliação do Modelo:** A precisão do classificador é calculada comparando as previsões com os rótulos reais das primeiras 100 imagens do conjunto de testes.

```
precisao = np.mean(previsao == teste_rotulos[:100])
print(f'Precisão das classificações: {precisao * 100:.2f}%')
```

5. **Visualização dos Resultados:** Cinco imagens aleatórias do conjunto de teste são selecionadas e exibidas junto com suas classificações utilizando a biblioteca matplotlib. A figura é salva como predictions.png para que o programa execute com sucesso mesmo em ambientes sem interface gráfica, como o WSL (Windows Subsystem for Linux).

```
fig, axes = plt.subplots(1, 5, figsize=(10, 3))
for i, ax in enumerate(axes):
    ax.imshow(teste_caract[indices_aleatorios[i]].reshape(28, 28), cmap='gray')
    ax.set_title(f'Previsto: {previsao[indices_aleatorios[i]]}')
    ax.axis('off')

plt.savefig('predictions.png')
```

6. **Uso de Inteligência Artificial (IA):** O ChatGPT, IA da empresa OpenAI, foi utilizado como auxílio para a elaboração deste relatório, assim como para decidir qual seria a melhor biblioteca para realizar a paralelização do problema abordado.

Resultados

O código exibe a precisão obtida pelo modelo e gera uma imagem com previsões visuais para algumas amostras do conjunto de testes. A execução paralelizada permite uma classificação mais eficiente em comparação com uma implementação sequencial. Para fins de comprovação, testamos uma versão do código sem a paralelização e a versão final, com paralelização.

Versão	Sequencial	Paralelizada
Tempo	00:00:27	00:00:12

Conclusão

A implementação de KNN paralelizado utilizando joblib demonstrou ser uma solução eficaz para melhorar a eficiência computacional do classificador de imagens MNIST. Para aumentar ainda mais a escalabilidade, pode-se explorar técnicas como redução de dimensionalidade ou otimização com estruturas de dados como KD-Trees.