Perceptron

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Sistemas Inteligentes

Sumário

Sumário de Códigos

Contributors

- **Marcos Antonio Tomé Oliveira**
  Graduando em Engenharia Mecatrônica

I Introdução

Esse relatório visa descrever de forma sucinta a implementação de uma rede neural do tipo perceptron de uma única camada.

pode melhorar minha documentação

II Pré-processamento

O pré-processamento da base de dados consiste em carregar o conjunto Iris, realizar a filtragem de classes para criar subconjuntos linearmente separáveis e não linearmente separáveis, e dividir esses subconjuntos em *folds* para validação cruzada.

```
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.model_selection import KFold
from sklearn.preprocessing import
    LabelEncoder

# Carregar a base Iris
iris = datasets.load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names
```

Listing 1. Carregamento da base Iris

O código acima importa bibliotecas essenciais e carrega a base Iris com suas respectivas características (‘X‘) e rótulos (‘y‘), além de armazenar os nomes das variáveis e das classes.

Criação do DataFrame e Divisão Linear/Não-Linear

A base é convertida para ‘DataFrame‘ e dividida em dois subconjuntos: um linearmente separável e outro não linearmente separável, para avaliar a robustez do modelo.

```
df = pd.DataFrame(X, columns=feature_names)
df["label"] = y

# Linearmente separavel: classes 0 e 1 (
    setosa e versicolor)
linear_df = df[df["label"].isin([0, 1])].
    reset_index(drop=True)

# Nao linearmente separavel: classes 1 e 2
    (versicolor e virginica)
nonlinear_df = df[df["label"].isin([1, 2])
    ].reset_index(drop=True)
```

Listing 2. Criação dos subconjuntos linear e não linear

Função de Divisão em Folds

A divisão do dataset acontece em três partes essenciais

- Essa função divide a base de dados em ‘n_splits‘ folds utilizando validação cruzada (‘KFold‘) e salva cada fold como arquivos CSV separados para treino e teste.
- Os dois conjuntos são salvos separadamente, cada um em 5 folds.

```
def save_kfolds(df, prefix, n_splits=5):
    kf = KFold(n_splits=n_splits, shuffle=
        True, random_state=42)
    X = df[feature_names].values
    y = df["label"].values

    for fold_num, (train_idx, test_idx) in
        enumerate(kf.split(X), start=1):
        train_data = df.iloc[train_idx]
        test_data = df.iloc[test_idx]

        train_path = f"build/data/{prefix}
            _fold{fold_num}_train.csv"
        test_path = f"build/data/{prefix}
            _fold{fold_num}_test.csv"

        train_data.to_csv(train_path, index
            =False)
        test_data.to_csv(test_path, index=
            False)
```

Listing 3. Função para salvar os folds

```
save_kfolds(linear_df, "iris_linear")
save_kfolds(nonlinear_df, "iris_nonlinear")

print("Conjuntos 'iris_linear' (classes 0 e
     1) e 'iris_nonlinear' (classes 1, 2)
    salvos em 5 folds cada.")
```

Listing 4. Salvamento dos conjuntos em arquivos CSV

III Implementação do Perceptron

- A implementação do perceptron em linguagem C utiliza uma estrutura (`struct`) que armazena os parâmetros da rede, como pesos, taxa de aprendizado, número de entradas e bias.
- A seguir, implementamos um Perceptron simples em linguagem C, com bias e pesos inicializados aleatoriamente.

III-A Criação do Perceptron

```c
Perceptron* create_perceptron(int num_inputs, float learning_rate) {
    Perceptron *p = (Perceptron*) malloc(sizeof(Perceptron));
    p->num_inputs = num_inputs;
    p->learning_rate = learning_rate;
    p->bias = 1.0;

    p->weights = (float*) malloc((num_inputs + 1) * sizeof(float));
    for (int i = 0; i <= num_inputs; i++) {
        p->weights[i] = ((float) rand() / RAND_MAX) * 2.0f - 1.0f;
    }

    return p;
}
```

Listing 5. Definição e inicialização do Perceptron

III-B Função de Ativação

A ativação do neurônio é realizada por uma função de degrau (step function), que retorna 1 se a soma ponderada das entradas for maior que zero, e 0 caso contrário.

- A função de ativação soma ponderadamente as entradas e aplica a função de degrau. Se a soma for maior que zero, a saída será 1; caso contrário, será 0.

```c
int activate(Perceptron *p, float *inputs) {
    float sum = 0.0;

    for (int i = 0; i < p->num_inputs; i++) {
        sum += inputs[i] * p->weights[i];
    }

    sum += p->bias * p->weights[p->num_inputs];

    return (sum > 0) ? 1 : 0;
}
```

Listing 6. Função de ativação

III-C Treinamento do Modelo

O treinamento ajusta os pesos do perceptron com base no erro entre a saída desejada e a saída estimada pela função de ativação.

- A função 'train' ajusta os pesos do perceptron com base no erro entre a saída esperada e a saída obtida. O algoritmo de aprendizado é do tipo supervisionado e baseado na Regra de Hebb modificada com taxa de aprendizado.
- Essa função lê os dados de teste de um arquivo CSV, aplica o perceptron para prever os rótulos e calcula a acurácia percentual com base nas classificações corretas.

```c
void train(Perceptron *p, float *inputs, int desired_output) {
    int guess = activate(p, inputs);
    int error = desired_output - guess;

    if (error != 0) {
        for (int i = 0; i < p->num_inputs; i++) {
            p->weights[i] += p->learning_rate * error * inputs[i];
        }
        p->weights[p->num_inputs] += p->learning_rate * error * p->bias;
    }
}
```

Listing 7. Função de treinamento

III-D Avaliação do Modelo

A função de avaliação compara as predições do perceptron com os rótulos reais da base de teste, computando a acurácia total.

```
void evaluate(Perceptron *p, const char *
    test_path) {
    FILE *test_file = fopen(test_path, "r")
        ;
    if (!test_file) {
        perror("Erro ao abrir base de teste
            ");
        return;
    }

    char line[1024];
    int correct = 0, total = 0;

    fgets(line, sizeof(line), test_file);

    while (fgets(line, sizeof(line),
        test_file)) {
        float inputs[4];
        int label;

        if (sscanf(line, "%f,%f,%f,%f,%d",
                   &inputs[0], &inputs[1],
                       &inputs[2], &inputs
                       [3], &label) == 5) {
            int prediction = activate(p,
                inputs);
            if (prediction == label)
                correct++;
            total++;
        }
    }

    fclose(test_file);

    float accuracy = 100.0f * correct /
        total;
    printf("Acuracia no teste: %.2f%% (%d
        corretos de %d)\n", accuracy,
        correct, total);
}
```

Listing 8. Função de avaliação

III-E Liberação de Memória

```
void destroy_perceptron(Perceptron *p) {
    free(p->weights);
    free(p);
}
```

Listing 9. Função de destruição

IV Plotagem dos Dados

Para visualizar a performance do modelo durante o treinamento, é realizada a plotagem da acurácia ao longo das épocas para cada fold.

- Após o treinamento do perceptron, os resultados de cada fold são plotados para análise de desempenho visual.
- Os dados de acurácia por época são lidos de arquivos '.csv' e representados graficamente para cada fold.
- Cada linha no gráfico representa a evolução da acurácia durante o treinamento em um dos folds.

Leitura dos logs e Plotagem

```
import pandas as pd
import matplotlib.pyplot as plt
import os
```

Listing 10. Importação de bibliotecas para plotagem

```
plt.figure(figsize=(10, 6))

for fold in range(1, 6):
    log_path = f"build/data/train_log_fold{
        fold}.csv"
    df = pd.read_csv(log_path)

    plt.plot(df['epoch'], df['accuracy'],
        label=f'Fold {fold}')
```

Listing 11. Plotagem da acurácia dos folds

```
plt.xlabel('Epoca')
plt.ylabel('Acuracia')
plt.title('Acuracia por Epoca - Todos os
    Folds')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('build/
    acuracia_treinamento_folds.png')
plt.show()
```

Listing 12. Configuração e salvamento do gráfico

Plotagem dos dados

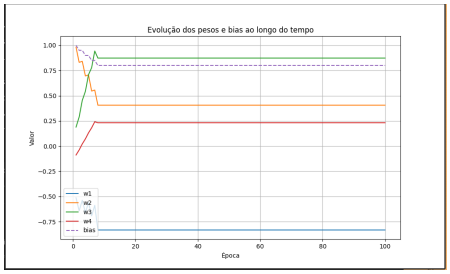

Figura 1. Evolução dos pesos e bias com o tempo - linearmente separável

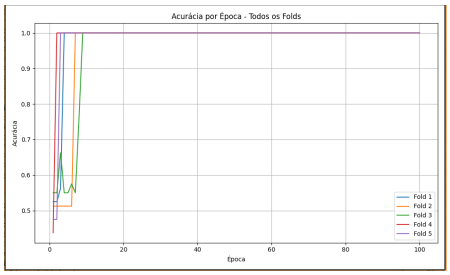

Figura 2. Evolução do erro com o passar das épocas - linearmente separavel

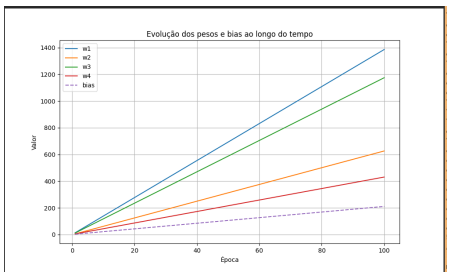

Figura 3. Evolução dos pesos e bias com o tempo - não linearmente sepáravel

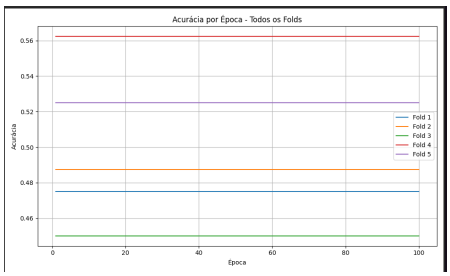

Figura 4. Evolução do erro com o passar das epócas - não linearmente sepáravel