Tutorial: Implementação do Perceptron com Datasets Clássicos

Introdução

O **Perceptron** é um dos algoritmos mais fundamentais do aprendizado de máquina, proposto por Frank Rosenblatt em 1957. É um classificador binário linear que forma a base para o entendimento de redes neurais mais complexas.

Conceitos Fundamentais:

- **Classificador Linear**: Separa classes usando um hiperplano
- **Supervisionado**: Aprende a partir de exemplos rotulados
- **Online**: Pode aprender incrementalmente
- **Convergência Garantida**: Para dados linearmente separáveis

---

Parte 1: Implementação da Classe Perceptron

1.1 Estrutura Básica

Vamos implementar o Perceptron passo a passo, entendendo cada componente:

```
import numpy as np

class Perceptron:
    """
    Implementação do algoritmo Perceptron para classificação binária.

    Parâmetros:
    -----------
    learning_rate : float
        Taxa de aprendizado (entre 0.0 e 1.0)
        - Valores menores: aprendizado mais lento, mas mais estável
        - Valores maiores: convergência mais rápida, mas pode oscilar

    n_epochs : int
        Número de passadas pelo dataset de treino
    """

    def __init__(self, learning_rate=0.01, n_epochs=100):
        self.learning_rate = learning_rate
        self.n_epochs = n_epochs
        self.weights = None
        self.bias = None
        self.errors_history = []  # Para acompanhar o progresso
```

1.2 Função de Ativação

A função de ativação define a saída do neurônio:

```
    def activation(self, x):
        """
        Função de ativação step (degrau).

        A função step é definida como:
        - Se x >= 0: retorna 1 (classe positiva)
        - Se x < 0: retorna 0 (classe negativa)

        Parâmetros:
        -----------
        x : float ou array
            Entrada líquida do neurônio (weighted sum + bias)

        Retorna:
        --------
        int ou array de ints
            Classe predita (0 ou 1)
        """
        return np.where(x >= 0, 1, 0)
```

1.3 Método de Treinamento

O coração do algoritmo - onde o aprendizado acontece:

```
    def fit(self, X, y, X_val=None, y_val=None):
        """
        Treina o perceptron usando a regra de atualização:
        w = w + η * (y_real - y_pred) * x

        Parâmetros:
        -----------
        X : array-like, shape = [n_samples, n_features]
            Dados de treinamento
        y : array-like, shape = [n_samples]
            Rótulos verdadeiros (0 ou 1)
        X_val, y_val : arrays opcionais
            Dados de validação para acompanhar o progresso
        """
        n_samples, n_features = X.shape

        # PASSO 1: Inicialização dos pesos
        # Começamos com pesos zero (também comum: pequenos valores aleatórios)
        self.weights = np.zeros(n_features)
        self.bias = 0

        # PASSO 2: Loop de treinamento
        for epoch in range(self.n_epochs):
            errors = 0

            # PASSO 3: Para cada exemplo de treinamento
            for idx, x_i in enumerate(X):
                # 3.1: Calcula a saída líquida (net input)
                # net = w1*x1 + w2*x2 + ... + wn*xn + b
                linear_output = np.dot(x_i, self.weights) + self.bias

                # 3.2: Aplica função de ativação
                y_predicted = self.activation(linear_output)

                # 3.3: Calcula o erro
                error = y[idx] - y_predicted

                # 3.4: Atualiza pesos e bias (Regra Delta)
                # Se error = 0: não há atualização
                # Se error = 1: move fronteira para incluir ponto
                # Se error = -1: move fronteira para excluir ponto
                update = self.learning_rate * error
                self.weights += update * x_i
                self.bias += update

                # Conta erros para monitoramento
                errors += int(update != 0.0)

            self.errors_history.append(errors)

            # Parada antecipada se convergiu
            if errors == 0:
                print(f"Convergiu na época {epoch + 1}")
                break
```

1.4 Método de Predição

```
    def net_input(self, X):
        """Calcula a entrada líquida (weighted sum + bias)"""
        return np.dot(X, self.weights) + self.bias

    def predict(self, X):
        """
        Faz predições para novos dados.

        Parâmetros:
        -----------
        X : array-like, shape = [n_samples, n_features]
            Dados para predição

        Retorna:
        --------
        array, shape = [n_samples]
            Classes preditas
        """
        return self.activation(self.net_input(X))
```

Parte 2: Visualização das Regiões de Decisão

Esta função é crucial para entender visualmente como o perceptron separa as classes:

```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

def plot_decision_regions(X, y, classifier, resolution=0.02):
    """
    Visualiza as regiões de decisão de um classificador 2D.

    Como funciona:
    1. Cria uma grade de pontos cobrindo todo o espaço de features
    2. Classifica cada ponto da grade
    3. Colore as regiões baseado na classificação
    4. Sobrepõe os pontos de treinamento

    Parâmetros:
    -----------
    X : array-like, shape = [n_samples, 2]
        Features (deve ter exatamente 2 dimensões para visualização)
    y : array-like, shape = [n_samples]
        Rótulos das classes
    classifier : objeto
        Classificador treinado com método predict()
    resolution : float
        Resolução da grade (menor = mais detalhado, mas mais lento)
    """

    # PASSO 1: Configurar cores e marcadores
    markers = ('o', 's')  # círculo para classe 0, quadrado para classe 1
    colors = ('red', 'blue')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # PASSO 2: Criar grade de pontos (meshgrid)
    # Definir limites do gráfico com margem
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    # Criar arrays de coordenadas
    # xx1: matriz com coordenadas x repetidas em cada linha
    # xx2: matriz com coordenadas y repetidas em cada coluna
    xx1, xx2 = np.meshgrid(
```

```
    xx1, xx2 = np.meshgrid(
        np.arange(x1_min, x1_max, resolution),
        np.arange(x2_min, x2_max, resolution)
    )

    # PASSO 3: Classificar cada ponto da grade
    # Achatar as matrizes e criar pares (x1, x2)
    grid_points = np.array([xx1.ravel(), xx2.ravel()]).T

    # Prever a classe de cada ponto
    Z = classifier.predict(grid_points)

    # Reformatar para o shape da grade
    Z = Z.reshape(xx1.shape)

    # PASSO 4: Plotar regiões coloridas
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # PASSO 5: Plotar pontos de treinamento
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(
            x=X[y == cl, 0],  # coordenadas x dos pontos da classe cl
            y=X[y == cl, 1],  # coordenadas y dos pontos da classe cl
            alpha=0.8,
            c=colors[idx],
            marker=markers[idx],
            label=f'Classe {cl}',
            edgecolor='black'
        )

    plt.legend(loc='upper left')
```

🎯 Parte 3: Exemplo Completo - Blobs Sintéticos

Vamos implementar um exemplo completo com dataset de blobs sintéticos:

```
 import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# PASSO 1: Gerar o Dataset
print("=" * 50)
print("EXEMPLO: BLOBS SINTÉTICOS")
print("=" * 50)

# make_blobs cria clusters gaussianos
X, y = datasets.make_blobs(
    n_samples=200,        # Total de pontos
    n_features=2,         # Número de features (2 para visualização)
    centers=2,            # Número de clusters (classes)
    cluster_std=1.5,      # Desvio padrão dos clusters
    center_box=(-5, 5),   # Limites para os centros
    random_state=42       # Seed para reprodutibilidade
)

print(f"Dataset gerado:")
print(f"- Amostras: {X.shape[0]}")
print(f"- Features: {X.shape[1]}")
```

```
print(f"- Classes: {np.unique(y)}")

# PASSO 2: Dividir em Treino e Teste
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,      # 30% para teste
    random_state=42,
    stratify=y          # Mantém proporção das classes
)

print(f"\nDivisão treino/teste:")
print(f"- Treino: {len(X_train)} amostras")
print(f"- Teste: {len(X_test)} amostras")

# PASSO 3: Normalização (Importante!)
"""
Por que normalizar?
- Garante que todas features tenham a mesma escala
- Previne que features com valores grandes dominem
- Acelera convergência
- Método: z-score (média=0, desvio=1)
"""
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)  # Fit no treino
X_test_std = scaler.transform(X_test)        # Apenas transform no teste

# PASSO 4: Treinar o Perceptron
ppn = Perceptron(learning_rate=0.01, n_epochs=50)
ppn.fit(X_train_std, y_train)

# PASSO 5: Avaliar o Modelo
y_pred = ppn.predict(X_test_std)
accuracy = accuracy_score(y_test, y_pred)

print(f"\nResultados:")
print(f"- Acurácia: {accuracy:.2%}")
print(f"- Erros finais no treino: {ppn.errors_history[-1]}")

# Verificar convergência
if 0 in ppn.errors_history:
    conv_epoch = ppn.errors_history.index(0)
    print(f"- Convergiu na época: {conv_epoch + 1}")
else:
    print("- Não convergiu completamente")

# PASSO 6: Visualizar Resultados
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Subplot 1: Regiões de Decisão
axes[0].set_title('Regiões de Decisão - Blobs')
plot_decision_regions(X_train_std, y_train, classifier=ppn)
axes[0].set_xlabel('Feature 1 (normalizada)')
axes[0].set_ylabel('Feature 2 (normalizada)')

# Subplot 2: Curva de Convergência
axes[1].plot(range(1, len(ppn.errors_history) + 1), ppn.errors_history, marker='o')
axes[1].set_xlabel('Épocas')
axes[1].set_ylabel('Número de erros')
axes[1].set_title('Convergência do Treinamento')
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# PASSO 7: Análise dos Pesos Aprendidos
print(f"\nPesos aprendidos:")
```

```
print(f"- w1: {ppn.weights[0]:.4f}")
print(f"- w2: {ppn.weights[1]:.4f}")
print(f"- bias: {ppn.bias:.4f}")

# A equação da fronteira de decisão é:
# w1*x1 + w2*x2 + bias = 0
# ou seja: x2 = -(w1/w2)*x1 - (bias/w2)
if ppn.weights[1] != 0:
    slope = -ppn.weights[0]/ppn.weights[1]
    intercept = -ppn.bias/ppn.weights[1]
    print(f"\nEquação da fronteira de decisão:")
    print(f"x2 = {slope:.2f} * x1 + {intercept:.2f}")
```

Exercícios Práticos

Agora é sua vez! Implemente os seguintes exemplos seguindo a estrutura apresentada:

Exercício 1: Iris Dataset (Setosa vs Versicolor)

Objetivo: Classificar duas espécies de flores Iris que são linearmente separáveis.

Dicas de Implementação:

1. **Carregar o dataset**:

```
from sklearn import datasets
iris = datasets.load_iris()

# IMPORTANTE: Use apenas as classes 0 e 1 (Setosa e Versicolor)
# Classe 2 (Virginica) não é linearmente separável das outras
mask = iris.target != 2
X = iris.data[mask]
y = iris.target[mask]

# Sugestão: Use apenas 2 features para visualização
# Por exemplo: índices [0, 2] = comprimento da sépala e comprimento da pétala
X = X[:, [0, 2]]
```

2. **Passos a seguir**:
    - Dividir em treino/teste (70/30)
    - Normalizar os dados
    - Treinar o perceptron
    - Plotar as regiões de decisão
    - Calcular e reportar a acurácia
3. **Resultado esperado**:
    - Acurácia próxima a 100%
    - Convergência em poucas épocas
4. **Pergunta para reflexão**:
    - O que acontece se você usar Versicolor vs Virginica (classes 1 e 2)?

Exercício 2: Moons Dataset

Objetivo: Demonstrar as limitações do perceptron com dados não-linearmente separáveis.

Dicas de Implementação:

1. **Gerar o dataset**:

```
from sklearn.datasets import make_moons

X, y = make_moons(
    n_samples=200,
    noise=0.15,      # Adiciona ruído realista
    random_state=42
)
```

2. **Observações importantes**:
    - Este dataset tem formato de duas "luas" entrelaçadas
    - NÃO é linearmente separável
    - O perceptron NÃO conseguirá uma boa acurácia
3. **Análise esperada**:
    - Acurácia em torno de 50-60%
    - Erros nunca chegam a zero
    - A visualização mostrará que a linha reta não consegue separar as luas
4. **Pergunta para reflexão**:
    - Como você modificaria o algoritmo para resolver este problema?

Exercício 3: Breast Cancer Wisconsin

Objetivo: Aplicar perceptron em um problema médico real.

Dicas de Implementação:

1. **Carregar o dataset**:

```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
X = cancer.data
y = cancer.target

print(f"Features: {cancer.feature_names}")
print(f"Classes: {cancer.target_names}")  # ['malignant' 'benign']
```

2. **Duas versões para implementar**:
    - **Versão A**: Use apenas 2 features para visualização (ex: índices [0, 1])
    - **Versão B**: Use todas as 30 features (sem visualização)
3. **Métricas adicionais**:

```
from sklearn.metrics import classification_report, confusion_matrix

# Após fazer predições:
print(classification_report(y_test, y_pred, target_names=cancer.target_names))
```

4. **Análise importante**:
    - Compare a acurácia com 2 features vs 30 features
    - Analise a matriz de confusão: falsos positivos vs falsos negativos

---

Exercício 4: Dataset de Classificação com Ruído

Objetivo: Trabalhar com dados que têm sobreposição entre classes.

Dicas de Implementação:

1. **Gerar o dataset**:

```
from sklearn.datasets import make_classification

X, y = make_classification(
    n_samples=200,
    n_features=2,
    n_redundant=0,
    n_informative=2,
    n_clusters_per_class=1,
    class_sep=1.5,        # Controla separação (menor = mais sobreposição)
    flip_y=0.05,          # 5% de ruído nos rótulos
    random_state=42
)
```

2. **Experimentos para fazer**:
    - Varie `class_sep` de 0.5 a 3.0
    - Varie `flip_y` de 0 a 0.2
    - Observe como a acurácia muda
3. **Implementar early stopping**:

```
# Modifique o método fit para parar se a acurácia de validação não melhorar
# por N épocas consecutivas
```

Exercício 5: Dataset Linearmente Separável Personalizado

Objetivo: Criar seu próprio dataset e entender a geometria da solução.

Dicas de Implementação:

1. **Criar dataset customizado**:

```
# Crie dois grupos de pontos bem separados
np.random.seed(42)

# Classe 0: centro em (-2, -2)
class_0 = np.random.randn(50, 2) + [-2, -2]

# Classe 1: centro em (2, 2)
class_1 = np.random.randn(50, 2) + [2, 2]

X = np.vstack([class_0, class_1])
y = np.hstack([np.zeros(50), np.ones(50)])
```

2. **Análise geométrica**:
    - Calcule e plote a equação da reta de decisão
    - Verifique que todos os pontos estão do lado correto
    - Experimente mover os centros mais próximos até o perceptron falhar

---

Relatório Final

Para cada exercício, seu relatório deve incluir:

1. **Descrição do Dataset**
    - Número de amostras e features
    - Distribuição das classes
    - É linearmente separável?
2. **Resultados**
    - Acurácia no treino e teste
    - Número de épocas até convergência
    - Tempo de treinamento
3. **Visualizações**
    - Gráfico de convergência
    - Regiões de decisão (quando possível)
    - Matriz de confusão

4. **Análise**
    - O perceptron foi adequado para este problema?
    - Que melhorias você sugeriria?
    - Comparação com suas expectativas

Critérios de Avaliação

- **Correção** (40%): O código funciona corretamente?
- **Análise** (30%): A interpretação dos resultados está correta?
- **Visualização** (20%): Os gráficos são claros e informativos?
- **Código** (10%): O código está limpo, comentado e organizado?

Dicas Finais

1. **Sempre normalize seus dados** - É crucial para convergência
2. **Monitore a convergência** - Use gráficos de erro
3. **Experimente com learning rates** - Muito alto oscila, muito baixo demora
4. **Entenda as limitações** - Perceptron só funciona para dados linearmente separáveis

Boa sorte com as implementações!