Perceptron detector de mensagens de spam

https://github.com/ProfessorFilipo/PythonBasico/tree/main/MachineLearning

Função `extrair_caracteristicas(mensagem)`

```python
def extrair_caracteristicas(mensagem):
    features = []
    palavras_chave = ['promoção', 'oferta', 'grátis', 'clique', 'dinheiro']
    mensagem_lower = mensagem.lower()
    for palavra in palavras_chave:
        features.append(1 if palavra in mensagem_lower else 0)
    return np.array(features)
```

- **Objetivo:** Extrair características binárias de uma mensagem com base na presença de palavras específicas.
- Converte a mensagem para minúsculas (`lower`) para facilitar a busca.
- Para cada palavra-chave, verifica se ela está na mensagem.
- Adiciona `1` se presente, `0` se ausente, formando um vetor de características (features).

Dados de treinamento

```python
treinamento = [
    ("Ganhe dinheiro rápido com esta oferta", 1),
    ("Reunião agendada para amanhã", 0),
    ("Clique aqui para uma surpresa grátis", 1),
    ("Seu boleto está disponível", 0),
    ("Promoção imperdível, aproveite agora", 1),
    ("Vamos conferir o relatório da reunião", 0),
    ("Oferta exclusiva: ganhe prêmios", 1),
    ("Atualização do seu cadastro", 0),
    ("Dinheiro fácil, clique já", 1),
    ("Horario da consulta marcada", 0)
]
```

- Uma lista de tuplas contendo uma mensagem e seu rótulo (`1`=spam, `0`=não spam).
- Usada para treinar o perceptron.

Preparação dos dados de treino

```
X_train = np.array([extrair_caracteristicas(msg) for msg, label in treinamento])
Y_train = np.array([label for msg, label in treinamento])
```

- Aplica `extrair_caracteristicas()` em todas as mensagens do treinamento, formando a matriz `X_train`.
- Cria o vetor `Y_train` com os valores de classificação (labels).

Inicialização dos pesos e do bias

```
np.random.seed(42)
pesos = np.random.randn(X_train.shape[1])
bias = 0.0
```

- Semente fixa para reprodutibilidade dos pesos aleatórios.
- `pesos` inicia com valores aleatórios (dois valores, pois há duas características).
- `bias` inicia em zero.

Configuração do treinamento

```
taxa_aprendizado = 0.1
epocas = 10
```

- `taxa_aprendizado`: quanto ajustamos os pesos a cada erro.
- `epocas`: número de passagens por todo o conjunto de dados para treinar.

Loop de treinamento

```
for epoca in range(epocas):
for x, y in zip(X_train, Y_train):
    linear_output = np.dot(x, pesos) + bias
    y_pred = 1 if linear_output >= 0 else 0
    erro = y - y_pred
    pesos += taxa_aprendizado * erro * x
    bias += taxa_aprendizado * erro
# Opcional
print(f'Época {epoca+1}: pesos={pesos}, bias={bias:.2f}')
```

- **Para cada época:**
  - Para cada exemplo de treino:
    - Calcula a saída linear (`np.dot(x, pesos) + bias`), que é uma combinação linear das características.
    - Decisão do perceptron: se a soma for >= 0, classifica como 1 (spam); senão, como 0.
    - Calcula o erro: diferença entre o rótulo verdadeiro `y` e o previsto `y_pred`.
    - Atualiza os pesos e o bias proporcional ao erro, ajustando a fronteira de decisão.
- Imprime os pesos e o bias após cada época, para acompanhar o aprendizado.

Classificação das mensagens de teste

```
print("\nClassificação das mensagens de teste:")
for msg in mensagens_testes:
x_test = extrair_caracteristicas(msg)
saida = np.dot(x_test, pesos) + bias
classe = "Spam" if saida >= 0 else "Não Spam"
print(f'"{msg}" => {classe}')
```

- Para cada mensagem de teste:
  - Extrai suas características.
  - Calcula a saída linear usando os pesos finais treinados.
  - Decide se a mensagem é "Spam" se a saída for >= 0 ou "Não Spam" se menor que zero.
- Imprime a mensagem e sua classificação.