Aprendizagem de máquina usando classes de Python

Resumo do *notebook* "Implementação de classe de modelos de neurônio único"

Uso básico de classes de Python para implementar modelos de aprendizagem de máquina.

Escopo: Este documento é um resumo do *notebook* do módulo 2, "Implementação de classe de modelo de neurônio único", do curso "Deep Learning: domínio das redes neurais", do MIT xPRO. Sua finalidade é oferecer uma visão geral dos aspectos principais desse *notebook*.

Requisitos: Este *notebook* retoma o conteúdo do *notebook* "Aprendizagem de máquina com Python" e implementa novamente os modelos de neurônio único, mas agora usando classes de Python. Este *notebook*, portanto, utiliza o código do *notebook* trabalhado anteriormente.

Índice

1.Implementação de classe de modelos de neurônio único

As funções desenvolvidas para modelos de regressão e classificação de neurônio único podem ser transformadas em métodos usando classes de Python. Além disso, as variáveis (os pesos) para um modelo específico também podem ser armazenadas como variáveis membros de uma instância específica da classe. Dessa forma, o modelo se torna um objeto único que integra tanto seus pesos específicos quanto os métodos que realizam cálculos com base nesses pesos.

Lembre-se:

- Um objeto é qualquer coisa que você deseja manipular ao trabalhar com código. Dependendo do tipo (classe) do objeto, pode haver atributos de dados associados.
- Uma classe é um modelo de código para criar e operar em um objeto.
- Uma instância é um objeto individual de uma classe.
- Um atributo de instância é uma variável que armazena dados ou propriedades para aquela instância específica.
- Um método é uma função que faz parte de uma classe e que normalmente opera em instâncias da classe.

Além de transformar funções em métodos dentro de uma classe, os programadores podem usar a biblioteca NumPy para representar vetores em vez de usar listas padrão de Python [ ]. NumPy oferece vantagens em relação às listas padrão, incluindo a capacidade de escrever código mais conciso e operações de maior desempenho. Para recursos adicionais, acesse NumPy.org e veja os tutoriais.

Este documento foca no uso de bibliotecas e classes de Python para um modelo de regressão de neurônio único. As mesmas técnicas e ferramentas do Python podem ser aplicadas a outros métodos de aprendizagem de máquina.

1.a. Biblioteca Python: NumPy

O seguinte código exemplifica como importar (import) NumPy e demonstra algumas das características mencionadas anteriormente.

```
# Importar a biblioteca
NumPy
import numpy as np

# Criar um array de NumPy a partir
de uma lista
x = np.array([1,2,3,4])
y = [1,2,3,4]
z = np.array(y)
print("x:", x)
print("y:", y)
print("z:", z)
print("Notice the difference between NumPy arrays x and z and the python list
y.")

# Ver as dimensões de um array usando shape
print("Shape of x:", x.shape)

# Criar um array multidimensional de 2x3 preenchido com 1
a = np.ones((2,3))
print("A multidimensional array")
print(a)
print("Shape of a:", a.shape)

# Realizar operações em vetores e arrays sem o uso de loops for
x = np.array([1,2])
w = np.array([2,2])
z = np.dot(x,w.T)
print("The dot product of x and transpose of w is:", z)

# Definir uma variável escalar para subtrair e multiplicar valores escalares
learning_rate = .01
# Definir um array de NumPy para realizar a subtração com floats, certifique-se de
inicializar o valor do seu array com float 0. e não o inteiro 0
# Tenha em mente que w_new = np.array([0, 0]) não
funcionaria
w_new = np.array([0.,0.])
w_new -= learning_rate * x
print("The value of w_new:", w_new)
```

A seguir, os resultados da impressão (print) do script acima ajudam a entender os benefícios de usar NumPy em relação à codificação sem as bibliotecas Python. Examine os resultados de impressão do script para identificar as características de NumPy.

2.Classe de modelos de neurônio único

Nesta seção, os modelos de neurônio único desenvolvidos na documentação "Aprendizagem de máquina com Python" foram reescritos para usar classes de Python e NumPy.

O script na próxima página exemplifica a criação de uma superclasse SingleNeuronModel para diferentes tipos de modelos de neurônio único, incluindo regressão e classificação.

Para cada uma dessas classes, as variáveis membro w e w_0 permanecem as mesmas, juntamente com a função forward. Além disso, os valores iniciais desses pesos são determinados automaticamente pelos dados de entrada com a implementação de funções integradas de Python e NumPy.

As funções de ativação e gradiente não são implementadas na superclasse SingleNeuronModel, pois cada subclasse implementa sua própria versão das funções de ativação e gradiente para diferenciar seus modelos.

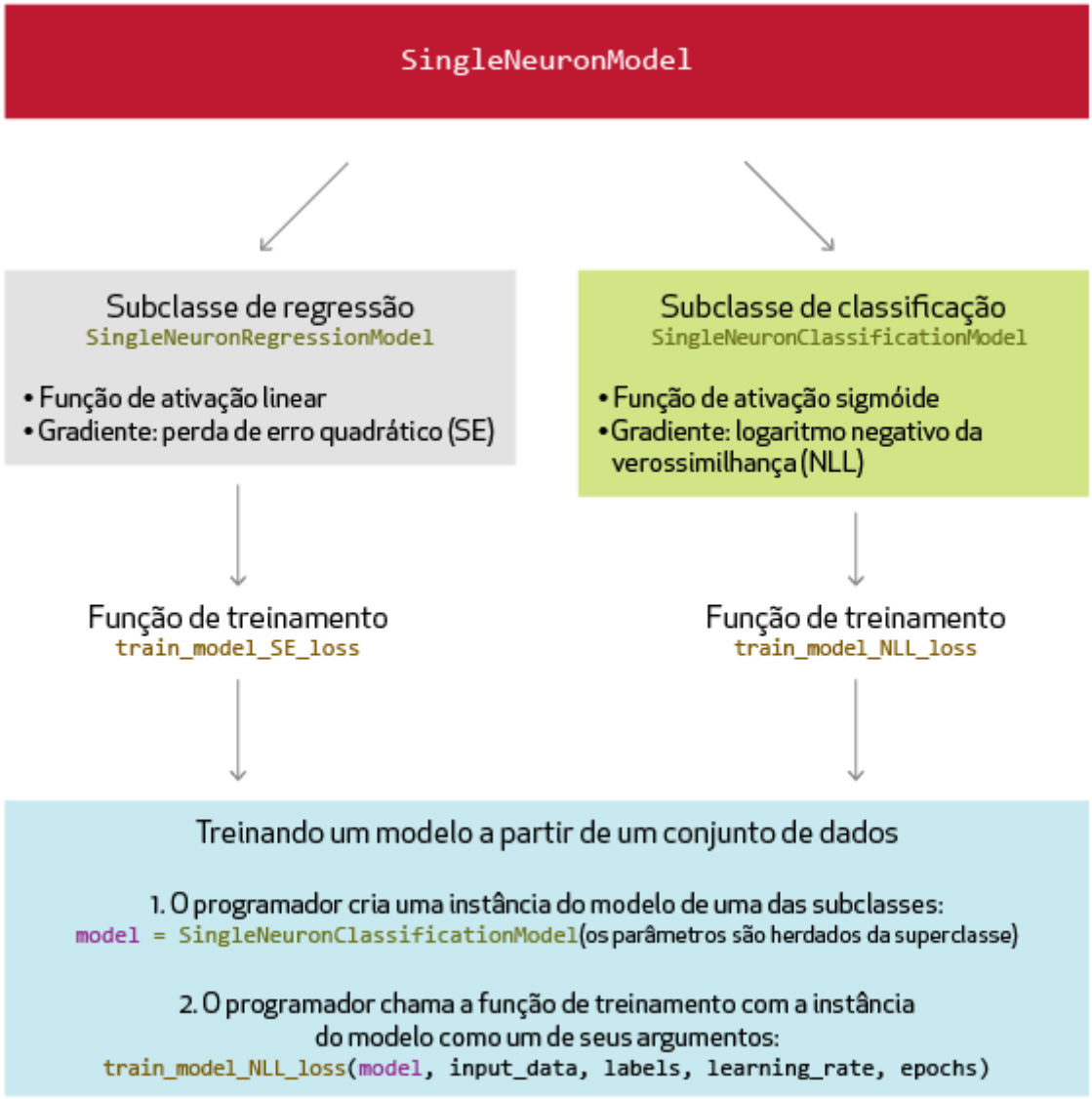
Importante: Este código é baseado no documento "Aprendizagem de máquina com Python" e bibliotecas Python. Tenha em mente que partes do código são omitidas para evitar repetição e focar em novos conceitos. Consulte o *notebook* do módulo 2 "Implementação de classe de modelos de neurônio único" para obter exemplos de código totalmente desenvolvidos.

O seguinte bloco de código cria as subclasses SingleNeuronRegressionModel e SingleNeuronClassificationModel e adapta as funções de ativação e gradiente de acordo com o respectivo modelo de aprendizagem de máquina.

```
# Reimplementar nosso modelo de regressão de neurônio único usando a classe base
SingleNeuronModel

class SingleNeuronRegressionModel(SingleNeuronModel):
    # Função de ativação linear para o modelo de regressão
    def activation(self, z):
        retornar z

    # Gradiente da saída em relação aos pesos para a ativação linear
    def gradient(self, x):
        self.grad_w = x self.grad_w_0 = 1.


# Nova implementação do modelo de classificação de neurônio único
class SingleNeuronClassificationModel(SingleNeuronModel):
# Função de ativação sigmoide para classificação
    def activation(self, z):
    return 1 / (1 + np.exp(-z) + self.non_zero_tolerance)

    # Gradiente da saída em relação aos pesos, para a ativação sigmoide
    def gradient(self, x):
        self.grad_w = self.a * (1-self.a) * x
        self.grad_w_0 = self.a * (1-self.a)
```

2.a. Treinar um modelo a partir de um conjunto de dados

Este diagrama oferece uma representação visual do uso de classes em Python e ferramentas do NumPy para modelos de regressão e classificação.



Vemos que os métodos comuns esperados de um modelo – incluindo as capacidades de "encaminhar" o cálculo de uma saída dada uma entrada e de "atualizar" os pesos do modelo com base em cálculos de gradiente – são capturados na superclasse. Em seguida, diferentes tipos de modelos, como modelos de regressão ou modelos de classificação, podem ser especializados como subclasses. Essas subclasses encapsulam a função de ativação específica a ser usada para aquele tipo de modelo, além de fornecer métodos para calcular os gradientes do modelo. Funções que implementam nossos procedimentos de treinamento com descida de gradiente (com as funções de perda apropriadas) são chamadas em instâncias específicas dessas classes (ou seja, em um “modelo” específico) para aprender ou treinar os pesos do modelo para produzir as melhores previsões dos dados de treinamento.

O código abaixo usa a implementação de classe para instanciar e treinar um modelo de classificação com um único neurônio:

```
# Dados de entrada: Array de NumPy de pontos de dados
linearmente separáveis em 2D
input_data = np.array([[1, 1],[1, 5],[-2, 3],[3, -4])
# Rótulos correspondentes
labels = [1, 1, 0, 1]

# Taxa de aprendizado e épocas
learning_rate = 0.01
epochs = 100

# Criar uma instância da subclasse SingleNeuronClassifictionModel
# O parâmetro in_features é herdado da superclasse SingleNeuronModel
# O parâmetro in_features é igual ao comprimento do primeiro ponto de
dados em 2D
model = SingleNeuronClassificationModel(in_features=len(input_data[0]))

# Chamar a função de treinamento com a instância como argumento
train_model_NLL_loss(model, input_data, labels, learning_rate, epochs)
print("\nFinal weights:")
print(model.w, model.w_0)
```

Conclusões: Classes e bibliotecas Python nos permitem escrever código mais flexível e modular, tornando os modelos de aprendizagem de máquina desenvolvidos em Python mais eficientes e legíveis.

Com o entendimento de classes para modelos de aprendizagem de máquina, os programadores podem usar a biblioteca PyTorch. O próximo módulo deste curso apresenta o PyTorch e seus recursos, que fornecem muitas classes e métodos predefinidos para a implementação direta de modelos de aprendizagem de máquina.