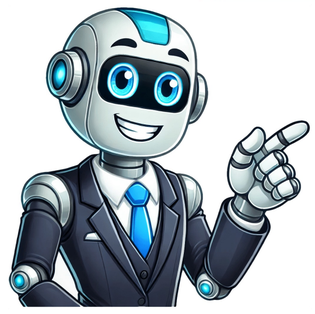
I'm not a bot



Machine learning with pytorch and scikit-learn

Get full access to Machine Learning with PyTorch and Scikit-Learn for 60K+ other titles, including a free 10-day trial of O'Reilly. Live events, courses curated by job role, and more are also available. This book in the bestselling Python Machine Learning series is a comprehensive guide to machine learning using PyTorch's simple framework. A print or Kindle purchase includes a free PDF eBook. Key Features: - Learn applied machine learning with a solid theoretical foundation - Clear explanations take you deep into theory and practice - Updated and expanded to cover PyTorch, transformers, XGBoost, graph neural networks, and best practices Description: Machine Learning with PyTorch and Scikit-Learn is a comprehensive guide covering the essentials of machine learning and deep learning with PyTorch. It serves as both a tutorial and reference, teaching you principles for building models and applications. Key Topics: - Explore frameworks, models, and techniques for machines to learn from data - Train machine learning classifiers on images, text, and more - Build and train neural networks, transformers, and boosting algorithms - Discover best practices for evaluating and tuning models Who is this book for? This book is for developers and data scientists who want to create practical machine learning and deep learning applications using scikit-learn and PyTorch. You'll need a good understanding of calculus and linear algebra before starting. This tutorial is designed to take intermediate to advanced learners of machine learning to the next level by teaching them how to build and deploy robust models using PyTorch and Scikit-learn. Learners will discover how to implement common algorithms and techniques, optimize their models for better performance, test and debug their work, and deploy their models in production environments. To maximize the benefits of this tutorial, learners should have a solid grasp of linear algebra, calculus, probability, statistics, and Python programming skills, as well as familiarity with the necessary technologies including PyTorch, Scikit-learn, NumPy, Pandas, Matplotlib, and Scikit-image. Machine learning is a subset of artificial intelligence that involves training algorithms to make predictions or decisions based on data. Key concepts in machine learning include supervised and unsupervised learning, regression, classification, clustering, dimensionality reduction, as well as PyTorch's dynamic computation graph and automatic differentiation. PyTorch and Scikit-learn serve as the core tools for building and deploying machine learning models. By combining algorithms and techniques, these libraries enable learners to build robust models that can be applied in a variety of settings. To prevent overfitting, use cross-validation to evaluate model performance, feature selection techniques to reduce dimensionality, and early stopping to prevent overfitting. Common pitfalls include: - Overfitting: when a model is too complex and fits the training data too closely - Underfitting: when a model is too simple and fails to capture the underlying patterns in the data - Data leakage: when the model is trained on data that is not representative of the population - Model drift: when the model is trained on data that is no longer representative of the population Implementation Guide: Step 1: Importing Libraries and Loading Data ```python import torch import torch.nn as nn import torch.optim as optim from sklearn.datasets import load_iris from sklearn.model_selection import train_test_split from sklearn.preprocessing import StandardScaler ``` Step 2: Data Preprocessing ```python # Load the Iris dataset iris = load_iris() X = iris.data y = iris.target # Split the data into training and testing sets X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) # Standardize the features using the StandardScaler scaler = StandardScaler() X_train = scaler.fit_transform(X_train) X_test = scaler.transform(X_test) ``` Step 3: Building and Training the Model ```python class Net(nn.Module): def __init__(self): super(Net, self).__init__() self.fc1 = nn.Linear(4, 10) # input layer (4) -> hidden layer (10) self.fc2 = nn.Linear(10, 3) # hidden layer (10) -> output layer (3) def forward(self, x): x = torch.relu(self.fc1(x)) # activation function for hidden layer x = self.fc2(x) return x model = Net() criterion = nn.CrossEntropyLoss() optimizer = optim.SGD(model.parameters(), lr=0.01) for epoch in range(10): optimizer.zero_grad() outputs = model(X_train) loss = criterion(outputs, y_train) loss.backward() optimizer.step() print('Epoch {}: Loss = {:.4f}'.format(epoch+1, loss.item())) ``` Step 4: Evaluating the Model ```python model.eval() with torch.no_grad(): outputs = model(X_test) _, predicted = torch.max(outputs, 1) accuracy = (predicted == y_test).sum().item() / len(y_test) print('Test Accuracy: {:.2f}%'.format(accuracy * 100)) ``` Code Examples: Example 1: Linear Regression ```python import numpy as np from sklearn.linear_model import LinearRegression X = np.random.rand(100, 1) y = 3 + 2 * X + np.random.randn(100, 1) model = LinearRegression() model.fit(X, y) ``` Example 2: Logistic Regression ```python import numpy as np from sklearn.linear_model import LogisticRegression X = np.random.rand(100, 1) y = (X > 0.5).astype(int) model = LogisticRegression() model.fit(X, y) ``` 1. # Train a linear regression model model = LinearRegression() model.fit(X, y) # Print the coefficients print('Coefficient: {:.2f}'.format(model.coef_[0])) print('Intercept: {:.2f}'.format(model.intercept_)) Example 2: Decision Tree import numpy as np from sklearn.tree import DecisionTreeClassifier # Generate some random data X = np.random.rand(100, 1) y = np.where(X[:, 0] < 0.5, 0, 1) # Train a decision tree model model = DecisionTreeClassifier() model.fit(X, y) # Print the feature importances print('Feature Importances: {}'.format(model.feature_importances_)) Example 3: Support Vector Machine import numpy as np from sklearn.svm import SVC # Generate some random data X = np.random.rand(100, 1) y = np.where(X[:, 0] < 0.5, 0, 1) # Train a support vector machine model model = SVC() model.fit(X, y) # Print the coefficients print('Coefficients: {}'.format(model.coef_)) print('Intercept: {}'.format(model.intercept_)) With PyTorch and Scikit-Learn, this book provides a comprehensive guide for machine learning and deep learning using PyTorch. It serves as both a step-by-step tutorial and a reference that you can come back to as you build your machine learning systems. The time has finally come to talk about my new book. Initially, this project started out as an updated version of "Python Machine Learning". But we made such significant changes that we felt it warranted a fresh title. So, you're probably wondering what's changed. In this post, I'm excited to share all the details with you. First, let me give you a quick overview of how the book is structured. This comprehensive guide covers both "traditional" machine learning and deep learning. The first part introduces you to fundamental concepts like data preprocessing, model evaluation, and hyperparameter tuning using scikit-learn. But things get really interesting from chapter 11 onwards. This chapter marks a turning point in the book, as we dive into implementing multilayer neural networks from scratch in NumPy and exploring backpropagation step by step. The second half of the book focuses on deep learning, covering topics like image and text classification, generating images, and even graph-structured data. We also touch on reinforcement learning, a subfield that's gaining popularity. You might notice that this structure is similar to "Python Machine Learning", 3rd edition. But don't worry, we've made some significant changes - including two brand new chapters and several rewritten sections. One of the major updates you'll notice is that we've transitioned from TensorFlow to PyTorch for deep learning code examples. This was a big undertaking, but I'm grateful to Yuxi (Hayden) Liu for helping me with this transition. With over 770 pages in total, we had to be careful not to overload the book while still keeping it print-friendly 😉. And that's not all - we've also added a section on using PyTorch Lightning, a library that makes organizing code and projects a breeze, especially when working with multiple GPUs. As someone who's worked closely with the PyTorch Lightning team, I can attest to its power and convenience. Looking forward to sharing more PyTorch Lightning insights in future chapters, regardless of your deep learning background. We'll dive into transformers for natural language processing, covering their evolution from recurrent neural networks and exploring various architectures like GPTs and BERTs. You don't need a supercomputer for this - we'll show you how to use freely available pre-trained models and fine-tune them on new tasks. Transformers are currently leading the way in state-of-the-art natural language processing, but that's not all. We're also going to explore graph neural networks, which allow us to work with graph-structured data like social network graphs and molecules. You'll learn how these networks work step by step, from structuring graphs as inputs to deep neural networks. Collaboration is key in this chapter, with contributions from Ben Kaufman, a Ph.D. student I'm co-advising with. Our goal is to help you adopt graph neural networks for tasks like molecular property prediction. Stay tuned for more PyTorch Lightning content and a fresh new look at machine learning and AI-based approaches, including our recent review article on bioactive ligand discovery and GPCR-ligand recognition. I'm thrilled with the fresh new layout of my book on Heatmaps in R, which features slimmer margins to accommodate more content while maintaining page limits. The addition of figure captions enhances the visual presentation. One notable improvement is the consistency in font size for mathematical symbols, making the math sections easier to read. Furthermore, syntax colors have been added, which significantly improves code readability. While printing might require some adjustment due to inline code's dark background, it may appeal to coders accustomed to dark backgrounds in their editors or terminals. The print version is available only in grayscale to keep costs reasonable. I've found my e-ink reader suitable for reading, and the book looks fine even on a black-and-white device. If you prefer color, consider an alternative tablet or explore the GitHub repository for full-color figures and embedded Jupyter notebooks. was a lot of work. You take notes, create a structure, make figures, and then eventually fill in the paragraphs one by one. Its a labor of love for me. Im open to hearing if you have any questions or feedback. Please dont hesitate to reach out. The discussion forum is the best place to do so.