Sample Code

Integrating a Core ML Model into Your App

Add a simple model to an app, pass input data to the model, and process the model's predictions.

Download

iOS 12.0+ | iPadOS 12.0+ | Xcode 15.2+

Overview

This sample app uses a trained model, `MarsHabitatPricer.mlmodel`, to predict habitat prices on Mars.

Add a model to your Xcode project

Add the model to your Xcode project by dragging the model into the project navigator.

You can see information about the model—including the model type and its expected inputs and outputs—by opening the model in Xcode. In this sample, the inputs are the number of solar panels and greenhouses, as well as the lot size of the habitat (in acres). The output is the predicted price of the habitat.

Create the model in code

Xcode also uses information about the model's inputs and outputs to automatically generate a custom programmatic interface to the model, which you use to interact with the model in your code. For `MarsHabitatPricer.mlmodel`, Xcode generates interfaces to represent the model (`MarsHabitatPricer`), the model's inputs (`MarsHabitatPricerInput`), and the model's output (`MarsHabitatPricerOutput`).

Use the generated `MarsHabitatPricer` class's initializer to create the model:

```
let marsHabitatPricer = try? MarsHabitatPricer(configuration: .init())
```

Get input values to pass to the model

This sample app uses a `UIPickerView` to get the model's input values from the user:

```
func selectedRow(for feature: Feature) -> Int {
    return pickerView.selectedRow(inComponent: feature.rawValue)
}

let solarPanels = pickerDataSource.value(for: selectedRow(for: .solarPanels), featu
let greenhouses = pickerDataSource.value(for: selectedRow(for: .greenhouses), featu
let size = pickerDataSource.value(for: selectedRow(for: .size), feature: .size)
```

Use the model to make predictions

The `MarsHabitatPricer` class has a generated `prediction(solarPanels: greenhouses:size:)` method that's used to predict a price from the model's input values—in this case, the number of solar panels, the number of greenhouses, and the size of the habitat (in acres). The result of this method is a `MarsHabitatPricerOutput` instance.

```
// Use the model to make a price prediction.
let output = try marsHabitatPricer.prediction(solarPanels: solarPanels,
                                              greenhouses: greenhouses,
                                              size: size)
```

Access the `price` property of `marsHabitatPricerOutput` to get a predicted price and display the result in the app's UI.

```
// Format the price for display in the UI.
let price = output.price
priceLabel.text = priceFormatter.string(for: price)
```

> **Note**
>
> The generated `prediction(solarPanels:greenhouses:size:)` method can throw an error. The most common type of error you'll encounter when working with Core ML occurs when the details of the input data don't match the details the model is expecting—for example, an image in the wrong format.

Build and run a Core ML app

Xcode compiles the Core ML model into a resource that's been optimized to run on a device. This optimized representation of the model is included in your app bundle and is what's used to make predictions while the app is running on a device.

See Also

Core ML models

- Getting a Core ML Model
  Obtain a Core ML model to use in your app.
- Updating a Model File to a Model Package
  Convert a Core ML model file into a model package in Xcode.

`class MLModel`
An encapsulation of all the details of your machine learning model.

- Model Customization
  Expand and modify your model with new layers.
- Model Personalization
  Update your model to adapt to new data.