SwiftXGBoost Documentation

Release stable

Jul 07, 2020

Contents

This page contains links to all the related documents on the swift package. The package itself is hosted at github.

CHAPTER 1

Contents

1.1 Swift Package Introduction

This document gives a basic walkthrough of xgboost swift package.

List of other helpful links

- Swift examples
- Swift API Reference

1.1.1 Install XGBoost

To install XGBoost, please follow instructions at GitHub.

1.1.2 Include in your project

SwiftXGBoost uses Swift Package Manager, to use it in your project, simply add it as a dependency in your Package.swift file:

```
.package(url: "https://github.com/kongzii/SwiftXGBoost.git", from: "0.7.0")
```

1.1.3 Python compatibility

With PythonKit package, you can import Python modules:

```
let numpy = Python.import("numpy")
let pandas = Python.import("pandas")
```

And use them in the same way as in Python:

```
let dataFrame = pandas.read_csv("Examples/Data/veterans_lung_cancer.csv")
```

And then use them with SwiftXGBoost, check AftSurvival for a complete example.

1.1.4 TensorFlow compatibility

If you are using S4TF toolchains, you can utilize tensors directly:

```
let tensor = Tensor<Float>(shape: TensorShape([2, 3]), scalars: [1, 2, 3, 4, 5, 6])
let tensorData = try DMatrix(name: "tensorData", from: tensor)
```

1.1.5 Data Interface

The XGBoost swift package is currently able to load data from:

- LibSVM text format file
- Comma-separated values (CSV) file
- NumPy 2D array
- Swift for Tensorflow 2D Tensor
- XGBoost binary buffer file

The data is stored in a DMatrix class.

To load a libsvm text file into DMatrix class:

```
let svmData = try DMatrix(name: "train", from: "Examples/Data/data.svm.txt",␣
↪format: .libsvm)
```

To load a CSV file into DMatrix:

```
// labelColumn specifies the index of the column containing the true label
let csvData = try DMatrix(name: "train", from: "Examples/Data/data.csv",␣
↪format: .csv, labelColumn: 0)
```

Note: Use Pandas to load CSV files with headers.

Currently, the DMLC data parser cannot parse CSV files with headers. Use Pandas (see below) to read CSV files with headers.

To load a NumPy array into DMatrix:

```
let numpyData = try DMatrix(name: "train", from: numpy.random.rand(5, 10),␣
↪label: numpy.random.randint(2, size: 5))
```

To load a Pandas data frame into DMatrix:

```
let pandasDataFrame = pandas.DataFrame(numpy.arange(12).reshape([4, 3]),␣
↪columns: ["a", "b", "c"])
let pandasLabel = numpy.random.randint(2, size: 4)
let pandasData = try DMatrix(name: "data", from: pandasDataFrame.values,␣
↪label: pandasLabel)
```

Saving DMatrix into an XGBoost binary file will make loading faster:

```
try pandasData.save(to: "train.buffer")
```

Missing values can be replaced by a default value in the DMatrix constructor:

```
let dataWithMissingValues = try DMatrix(name: "data", from: pandasDataFrame.
↪values, missingValue: 999.0)
```

Various float fields and uint fields can be set when needed:

```
try dataWithMissingValues.set(field: .weight, values: [Float](repeating: 1,␣
↪count: try dataWithMissingValues.rowCount()))
```

And returned:

```
let labelsFromData = try pandasData.get(field: .label)
```

1.1.6 Setting Parameters

Parameters for Booster can also be set.

Using the set method:

```
let firstBooster = try Booster()
try firstBooster.set(parameter: "tree_method", value: "hist")
```

Or as a list at initialization:

```
let parameters = [Parameter(name: "tree_method", value: "hist")]
let secondBooster = try Booster(parameters: parameters)
```

1.1.7 Training

Training a model requires a booster and a dataset.

```
let trainingData = try DMatrix(name: "train", from: "Examples/Data/data.csv", format:␣
↪.csv, labelColumn: 0)
let boosterWithCachedData = try Booster(with: [trainingData])
try boosterWithCachedData.train(iterations: 100, trainingData: trainingData)
```

After training, the model can be saved:

```
try boosterWithCachedData.save(to: "0001.xgboost")
```

The model can also be dumped to a text:

```
let textModel = try boosterWithCachedData.dumped(format: .text)
```

A saved model can be loaded as follows:

```
let loadedBooster = try Booster(from: "0001.xgboost")
```

1.1.8 Prediction

A model that has been trained or loaded can perform predictions on data sets.

From Numpy array:

```
let testDataNumpy = try DMatrix(name: "test", from: numpy.random.rand(7, 12))
let predictionNumpy = try loadedBooster.predict(from: testDataNumpy)
```

From Swift array:

```
let testData = try DMatrix(name: "test", from: [69.0,60.0,7.0,0,0,0,1,1,0,1,0,0],␣
↪shape: Shape(1, 12))
let prediction = try loadedBooster.predict(from: testData)
```

1.1.9 Plotting

You can also save the plot of importance into a file:

```
try boosterWithCachedData.saveImportanceGraph(to: "importance") // .svg extension␣
↪will be added
```

1.1.10 C API

Both Booster and DMatrix are exposing pointers to the underlying C.

You can import a C-API library:

```
import CXGBoost
```

And use it directly in your Swift code:

```
try safe { XGBoosterSaveModel(boosterWithCachedData.booster, "0002.xgboost") }
```

safe is a helper function that will throw an error if C-API call fails.

1.1.11 More

For more details and examples, check out GitHub repository.