Parallel Machine Learning and Artificial Intelligence

Dr. Handan Liu

h.liu@northeastern.edu

Northeastern University

Spring 2020

Northeastern University
College of Engineering

Parallel Machine Learning

Preface

- XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable.
- It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.
- The XGBoost code can run on major distributed environment (MPI, Hadoop, SGE, etc.) and can solve problems beyond billions of examples in industrial scale.
- A large number of Kaggle players chose it for data mining competitions, and won the championship in Kaggle competitions.

XGBoost Parallelism

- The parallelism in gradient boosting can be implemented in the construction of individual trees.

What will we learn?

- XGBoost Basic
  - Introduction to Gradient Boost and XGBoost (some math)
  - Installation XGBoost in your local machine and cluster
  - How to prepare data for XGBoost
  - How to evaluate the performance of trained XGBoost models
- XGBoost Advanced
- XGBoost Tuning

What will we learn?

- XGBoost Basic
- XGBoost Advanced
  - How to serialize trained models to make predictions.
  - How to calculate importance scores and use them for feature selection.
  - How to monitor the performance of a model during training and set conditions for early stopping.
  - How to harness the XGBoost parallelism for training models faster.
  - How to rapidly speed up model training of XGBoost models using Amazon cloud infrastructure. (if possible)
- XGBoost Tuning

What will we learn?

- XGBoost Basic
- XGBoost Advanced
- XGBoost Tuning
  - An introduction to XGBoost parameters and heuristics for good parameter values.
  - How to tune the number and size of trees in a model.
  - How to tune the learning rate and number of trees in a model.
  - How to tune the sampling rates in stochastic variation of the algorithm.
  - How to set the parallel environments
  - XGBoost on GPU

Introduction to Gradient Boosting

Elements of Supervised Learning

- XGBoost is used for supervised learning problems, where we use the training data (with multiple features) x_i to predict a target variable y_i.
- The **model** in supervised learning usually refers to the mathematical structure of by which the prediction y_i is made from the input x_i.
- The **parameters** are the undetermined part that we need to learn from data. In linear regression problems, the parameters are the coefficients θ.

Objective Function: Training Loss + Regularization

- Objective function that is everywhere

$$Obj(\theta) = L(\theta) + \Omega(\theta)$$

Training Loss measures how well model fit on training data

Regularization measures complexity of model

- Loss on training data:
  - Square loss: $L(\theta) = \sum_i (y_i - \hat{y}_i)^2$
  - Logistic loss: $L(\theta) = \sum_i [y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})]$
- Regularization: how complicated the model is?
  - L2 norm: $\Omega(\omega) = \lambda \parallel \omega \parallel^2$
  - L1 norm (lasso): $\Omega(\omega) = \lambda \parallel \omega \parallel_1$

Objective Function: $Obj(\theta) = L(\theta) + \Omega(\theta)$

- Ridge regression: $\sum_{i=1}^{n}(y_i - \omega^T x_i)^2 + \lambda \parallel \omega \parallel^2$
  - Linear model, square loss, L2 regularization

- Lasso: $\sum_{i=1}^{n}(y_i - \omega^T x_i)^2 + \lambda \parallel \omega \parallel_1$
  - Linear model, square loss, L1 regularization

- Logistic regression:

$$\sum_{i=1}^{n}[y_i \ln\left(1 + e^{-\omega^T x_i}\right) + (1 - y_i)\ln\left(1 + e^{\omega^T x_i}\right) + \lambda \parallel \omega \parallel^2$$

  - Linear model, logistic loss, L2 regularization

Objective and Bias-Variance Trade-off

$$Obj(\theta) = L(\theta) + \Omega(\theta)$$

Training Loss measures how well model fit on training data

Regularization measures complexity of model

- Why do we want to contain two component in the objective?
- Optimizing training loss encourages predictive models
- Optimizing regularization encourages simple models

Regression Tree (CART)

- Regression tree (also known as classification and regression tree):
  - Decision rules same as in decision tree
  - Contains one score in each leaf value
- Tree Ensemble Methods
  - Very widely used, look for GBM, random forest…
  - Invariant to scaling of inputs, so you do not need to do careful features normalization.
  - Learn higher order interaction between features.
  - Can be scalable, and are used in Industry.

Regression Tree is not just for regression!

- Regression tree ensemble defines how you make the prediction score, it can be used for
  - Classification, Regression, Ranking….
  - ….
- It all depends on how you define the objective function!
- So far we have learned:
  - Using Square loss
    - ✓ Will results in common gradient boosted machine (GBM)
  - Using Logistic loss
    - ✓ Will results in LogitBoost

How Gradient Boosting Works

Gradient boosting involves three elements:

- A loss function to be optimized.
    - A benefit of the gradient boosting framework is that a new boosting algorithm does not have to be derived for each loss function that may want to be used, instead, it is a generic enough framework that any differentiable loss function can be used.
- A weak learner to make predictions.
- An additive model to add weak learners to minimize the loss function.

How Gradient Boosting Works

Gradient boosting involves three elements:

- A loss function to be optimized.
- A weak learner to make predictions.
  - Decision trees are used as the weak learner in gradient boosting.
  - Specifically regression trees are used that output real values for splits and whose output can be added together, allowing subsequent models outputs to be added and *correct* the residuals in the predictions.
  - Trees are constructed in a greedy manner, choosing the best split points based on purity scores like Gini or to minimize the loss.

- An additive model to add weak learners to minimize the loss function.

Addictive Model

- Trees are added one at a time, and existing trees in the model are not changed. A gradient descent procedure is used to minimize the loss when adding trees.
- After calculating the loss, to perform the gradient descent procedure, we must *add* a tree to the model that reduces the loss (i.e. follow the gradient) -- by parameterizing the tree → moving in the right direction by reducing the residual loss.
- A fixed number of trees are *added* or training stops once loss reaches an acceptable level or no longer improves on an external validation dataset.

Improvements to Basic Gradient Boosting

- Gradient boosting is a greedy algorithm and can overfit a training dataset quickly. It can benefit from regularization methods that penalize various parts of the algorithm and generally improve the performance of the algorithm by reducing overfitting.
- There are 4 enhancements to basic gradient boosting:
  - Tree Constraints
  - Shrinkage or Weighted Updates
  - Random Sampling / Stochastic Gradient Boosting
  - Penalized Learning / Penalized Gradient Boosting

Tree Constraints

- It is important that the weak learners have skill but remain weak. There are a number of ways that the trees can be constrained:
    1) Number of trees
    2) Tree depth
    3) Number of nodes or number of leaves, like depth
    4) Number of observations per split
    5) Minimum improvement to loss

Weighted Updates

- The predictions of each tree are added together sequentially. The contribution of each tree to this sum can be weighted to slow down the learning by the algorithm.
    - This weighting is called a shrinkage or a learning rate.

- The effect of weighted updates

- It is common to have small values in the range of 0.1 to 0.3, as well as values less than 0.1.

Stochastic Gradient Boosting

- This same benefit can be used to reduce the correlation between the trees in the sequence in gradient boosting models.
  - This variation of boosting is called stochastic gradient boosting.

- A few variants of stochastic boosting that can be used:
  - Subsample rows before creating each tree.
  - Subsample columns before creating each tree
  - Subsample columns before considering each split.

Penalized Gradient Boosting

- Classical decision trees like CART are not used as weak learners, instead a modified form called a regression tree is used that has numeric values in the leaf nodes (also called terminal nodes).

- The values in the leaves of the trees can be called weights in some literature. As such, the leaf weight values of the trees can be regularized using popular regularization functions.

Introduction to XGBoost

What is XGBoost

- XGBoost stands for eXtreme Gradient Boosting.
- XGBoost is a software library that you can download and install on your machine, then access from a variety of interfaces. Specifically, XGBoost supports the following main interfaces:
  - Command Line Interface (CLI).
  - C++ (the language in which the library is written).
  - Python interface as well as a model in scikit-learn.
  - R interface as well as a model in the caret package.
  - Julia support.
  - Java and JVM languages like Scala and platforms like Hadoop.

XGBoost Features

The library is laser focused on computational speed and model performance. It does offer a number of advanced features.

- Model Features
- System Features
- Algorithm Features

Model Features

- Three main forms of gradient boosting are supported:
  - Gradient Boosting algorithm also called gradient boosting machine (GBM) including the learning rate.
  - Stochastic Gradient Boosting (SGB) with sub-sampling at the row, column and column per split levels.
  - Regularized Gradient Boosting with both L1 and L2 regularization.

System Features

- The library provides a system for use in a range of computing environments, not least:
  - Parallelization of tree construction using all of your CPU cores during training.
  - Distributed Computing for training very large models using a cluster of machines.
  - Out-of-Core Computing for very large datasets that don't fit into memory.
  - Cache Optimization of data structures and algorithm to make best use of hardware.

Algorithm Features

- Some key algorithm implementation features include:
  - Sparse Aware implementation with automatic handling of missing data values.
  - Block Structure to support the parallelization of tree construction.
  - Continued Training so that you can further boost an already fitted model on new data.

- XGBoost is free open source software available for use under the permissive Apache-2 license.

Why use XGBoost?

1. XGBoost Execution Speed.
   - Generally, XGBoost is fast. Really fast when compared to other implementations of gradient boosting and bagged decision trees.

2. XGBoost Model Performance.
   - XGBoost dominates structured or tabular datasets on classification and regression predictive modeling problems.

What Algorithm Does XGBoost Use?

- The XGBoost library implements the gradient boosting decision tree algorithm.
    - This algorithm goes by lots of different names such as gradient boosting, multiple additive regression trees, stochastic gradient boosting or gradient boosting machines.

- This approach supports both *regression* and *classification predictive* modeling problems.

Develop Your First XGBoost Model in Python Scikit-learn

Install XGBoost Library in Anaconda

- On Local machine, install on Windows, macOS or Linux:
  - conda install -c anaconda py-xgboost
  - OR: git clone --recursive https://github.com/dmlc/xgboost
  - Or: sudo pip install xgboost

- One the cluster, install on your $HOME
  - $ conda install -c anaconda py-xgboost --prefix=$HOME/xgboost
  - $ python
  - >>> import sys
  - >>> sys.path.append(''$HOME/xgboost'')

- You can learn more about how to install XGBoost for different platforms on the XGBoost Installation Guide.

  http://xgboost.readthedocs.io/en/latest/build.html

- For up-to-date instructions for installing XGBoost for Python see the XGBoost Python Package.

  https://github.com/dmlc/xgboost/tree/master/python-package

Example: Predict Onset of Diabetes

- Pima Indians onset of diabetes dataset
  - 8 input variables that describe medical details of patients, and
  - 1 output variable to indicate whether the patient will have an onset of diabetes within 5 years.

- Load and Prepare Data:
- Train the XGBoost Model
- Make Predictions with XGBoost Model

The End!

Assignment:

1. Read the materials listed in the class.
2. Practice the codes after class.