**Gradient Boosting in Apache Ignite.**

Gradient Boosting (GDB) is a machine learning technique, which produces a prediction model in the form of an [ensemble](https://en.wikipedia.org/wiki/Ensemble_learning) of weak prediction models. The gradient boosting algorithm tries to solve the minimization error problem on a learning sample in a functional space where each function is a model. Each model in this composition tries to predict a gradient of error for points in the feature space and these predictions will be summed with some weight to model the answer. This algorithm may be used for regression and classification problems. For more information see [Wikipedia](https://en.wikipedia.org/wiki/Gradient_boosting).

In Ignite ML there is an implementation of a general GDB algorithm and GDB-on-trees algorithm. General GDB (GDBRegressionTrainer and GDBBinaryClassifierTrainer) allows the use of any trainer for training each model in composition. GDB on trees uses some optimizations specifics for trees like indexes for avoiding sorting during the building phase of Decision Trees.

**Model**

The Apache Ignite ML module implementations of the GDB algorithm use GDBModel wrapping ModelsComposition for representing the composition of a few models. ModelsComposition implements a common Model interface and can be used as a usual model:

GDBModel model = ...;  
double prediction = model.apply(featureVector);

In case of GDB, GDBModel uses WeightedPredictionsAggregator as the model answer reducer. This aggregator computes an answer of meta-model as “result = bias + p1\*w1 + p2\*w2 + ...” where pi - answer of i-th model, wi - weight of model in composition. GDB use mean value of labels for bias-parameter in the aggregator.

**Trainer**

Training of GDB is represented by GDBRegressionTrainer, GDBBinaryClassificationTrainer and GDBRegressionOnTreesTrainer, GDBBinaryClassificationOnTreesTrainer for general GDB and GDB on trees respectively. All trainers have these parameters:

* gradStepSize - sets constant weight of each model in composition, in future versions of Ignite ML, this parameter may be computed dynamically;
* cntOfIterations - sets maximum of models in composition after training;
* checkConvergenceFactory - sets factory for construction of convergence checker used for preventing overfitting and learning of many useless models while training.

For classifier trainers there is an additional parameter:

* loss - sets loss computer on some learning example from the training dataset.

There are several factories for convergence checkers:

* ConvergenceCheckerStubFactory creates checker that always returns false for convergence check. So, in this case, model composition size will have cntOfIterations models;
* MeanAbsValueConvergenceCheckerFactory creates checker that computes the mean value of the absolute gradient values on each example from the dataset and returns true if this it is less than the used-defined threshold:
* MedianOfMedianConvergenceCheckerFactory creates checker that computes the median of the median absolute gradient values on each data partition. Such a method is less sensitive for anomalies in the learning dataset but GDB may take longer to converge.

Example of training:

// Set up trainer

GDBTrainer trainer = new GDBBinaryClassifierOnTreesTrainer(

learningRate, countOfIterations, new LogLoss()

).withCheckConvergenceStgyFactory(new MedianOfMedianConvergenceCheckFactory(precision));  
  
// Build the model  
GDBModel mdl = trainer.fit(  
 datasetBuilder,  
 featureExtractor,  
 labelExtractor  
);

**Example**

To see how GDB Classifier can be used in practice, try this [example](https://github.com/apache/ignite/blob/master/examples/src/main/java/org/apache/ignite/examples/ml/tree/boosting/GDBOnTreesClassificationTrainerExample.java) that is available on GitHub and delivered with every Apache Ignite distribution.