Gradient boosting trees (GBTs)

Module Objectives

- After completing this set of lessons, you should be able to:
  - Understand the Pipelines API for Random Forests and Gradient-Boosted Trees
  - Describe default's Input and Output columns
  - Perform classification and regression with RFs and GBTs
  - Understand and use RFs and GBTs parameters
  - Outline the differences between RFs and GBTs regarding its parameters

Gradient-boosting trees

- Like Random forests, they are ensembles of Decision Trees
- Iteratively train decision trees in order to minimize a loss function
- Supports binary classification
- Supports regression
- Supports continuous and categorical features

Basic algorithm

- Iteratively trains a sequence of decision trees
- On each iteration, uses the current ensemble to make label predictions and compares it to true labels
- Re-labels dataset to put more emphasis on instances with poor predictions, according to a loss function
- With each iteration, reduces the loss function, thus correct for previous mistakes
- Supported loss functions:
  - classification: Log Loss (twice binomial negative log likelihood)
  - regression: Squared Error (L2 loss, default) and Absolute Error (L1 loss, more robust to outliers)

Gradient-Boosted Trees Parameters

–loss: loss function (Log Loss, for classification, Squared and Absolute errors, for regression)

–numIterations: number of trees in the ensemble

- each iteration produces one tree
- if it increases:
  - –model gets more expressive, improving training data accuracy
  - –test-time accuracy may suffer (if too large)

–learningRate: should NOT need to be tuned

- if behavior seems unstable, decreasing it may improve stability

Validation while training

- Gradient-Boosted Trees can overfit when trained with more trees
- The method `runWithValidation` allows validation while training
  - takes a pair of RDDs: training and validation datasets
- Training is stopped when validation error improvement is less than the tolerance specified as `validationTol` in `BoostingStrategy`
  - validation error decreases initially and later increases
  - there might be cases in which the validation error does not change monotonically
    - set a large enough negative tolerance
    - examine validation curve using `evaluateEachIteration`, which gives the error or loss per iteration
    - tune the number of iterations

Inputs and Outputs

Param name	Type(s)	Default	Description
labelCol	Double	"label"	Label to predict
featuresCol	Vector	"features"	Feature vector

Param name	Type(s)	Default	Description	Notes
predictionCol	Double	"prediction"	Predicted label	

GBT Classification (1)

```
from pyspark.ml.classification import GBTClassifier
from pyspark.ml.classification import GBTClassificationModel

gbtC = GBTClassifier().setLabelCol("indexedLabel").setFeaturesCol("indexedFeatures").setMaxIter(10)

pipelineGBTC = Pipeline().setStages([labelIndexer, featureIndexer, gbtC, labelConverter])

modelGBTC = pipelineGBTC.fit(trainingData)
```

GBT Classification (2)

```
predictionsGBTC = modelGBTC.transform(testData)

predictionsGBTC.select("predictedLabel", "label", "features").show(3)
```

```
+--------------+-----+--------------------+
|predictedLabel|label|            features|
+--------------+-----+--------------------+
|           1.0|  1.0|(692,[97,98,99,12...|
|           0.0|  0.0|(692,[98,99,100,1...|
|           0.0|  1.0|(692,[99,100,101,...|
+--------------+-----+--------------------+
only showing top 3 rows
```

GBT Classification (2)

```
gbtModelC = modelGBTC.stages[2]

print gbtModelC.toDebugString
```

```
GBTClassificationModel (uid=GBTClassifier_44089277a42aab7da71f) with 10 trees
  Tree 0 (weight 1.0):
    If (feature 378 <= 71.0)
     Predict: 1.0
    Else (feature 378 > 71.0)
     Predict: -1.0
  Tree 1 (weight 0.1):
    If (feature 490 <= 0.0)
     If (feature 133 in {3.0})
      Predict: 0.47681168808847
     Else (feature 133 not in {3.0})
      If (feature 180 <= 118.0)
       Predict: 0.47681168808847
      Else (feature 180 > 118.0)
       Predict: 0.47681168808847035
    Else (feature 490 > 0.0)
     Predict: -0.47681168808847
```

GBT regression

```
from pyspark.ml.regression import GBTRegressor
from pyspark.ml.regression import GBTRegressionModel

gbtR = GBTRegressor().setLabelCol("label").setFeaturesCol("indexedFeatures").setMaxIter(10)

pipelineGBTR = Pipeline().setStages([featureIndexer, gbtR])

modelGBTR = pipelineGBTR.fit(trainingData)
```

GBT regression

```
predictionsGBTR = modelGBTR.transform(testData)
predictionsGBTR.show(5)
```

```
+--------------------+-----+--------------------+----------+
|            features|label|     indexedFeatures|prediction|
+--------------------+-----+--------------------+----------+
|(692,[97,98,99,12...|  1.0|(692,[97,98,99,12...|       1.0|
|(692,[98,99,100,1...|  0.0|(692,[98,99,100,1...|       0.0|
|(692,[99,100,101,...|  1.0|(692,[99,100,101,...|       0.0|
|(692,[119,120,121...|  1.0|(692,[119,120,121...|       1.0|
|(692,[123,124,125...|  1.0|(692,[123,124,125...|       1.0|
+--------------------+-----+--------------------+----------+
only showing top 5 rows
```

Random Forests vs GBTs

- Number of trees
  - RFs: more trees reduce variance and the likelihood of overfitting; improves performance monotonically
  - GBTs: more trees reduce bias, but increase the likelihood of overfitting and performance can start to decrease if the number of trees grows too large
- Parallelization
  - RFs: can train multiples trees in parallel
  - GBTs: train one tree at a time
- Depth of trees
  - RFs: deeper trees
  - GBTs: shallower trees

Lesson Summary

- Having completed theis lesson, you should be able to:
  - –Understand the Pipelines API for Random Forests and Gradient-Boosted Trees
  - –Describe default's Input and Output columns
  - –Perform classification and regression with RFs and GBTs
  - –Understand and use RFs and GBTs parameters
  - –Outline the differences between RFs and GBTs regarding its parameters