# NATIONAL TECHNOLOGICAL INSTITUTE OF MEXICO
# TECHNOLOGICAL INSTITUTE OF TIJUANA

ACADEMIC SUBDIRECTION
Systems and Computing Department

SEMESTER
August – December 2021

ACADEMIC CAREER
Eng. Information and Comunications Technologies

SUBJECT AND KEY:
Big Data BDD – 1704TI9A

STUDENT'S NAME AND REGISTRATION:
Velázquez Farrera César Alejandro

NAME OF WORK:
Practice #3 – Gradient Boosting Classifier

UNIT:
Unit II

NAME OF TEACHER:
M.C. José Christian Romero Hernández

**Tijuana, Baja California**

**Due Date:** November 5th, 2021

## Introduction

In this present document, the topic of Gradient Boosting Classifier, how it works and examples will be explained. The example in this document is done with the documentation provided by Apache Spark in the following URL: [https://spark.apache.org/docs/2.4.7/mllib-ensembles.html#gradient-boosted-trees-gbts]

**Gradient Boosted Tree Classifier**

Gradient Boosting iteratively trains a sequence of decision trees. On each iteration, the algorithm uses the current ensemble to predict the label of each training instance and then compares the predictions with the true label. To correct the ensemble, a loss function is used to compare the input of the model and the true label. The dataset is re-labeled to put more emphasis on training instances with poor predictions. Thus, in the next iteration, the decision tree will help correct the previous mistakes. It will continue to iterate until it reaches a specified number of iterations.

**Pseudo-algorithm**

**Step 1.** Initialize the model with a constant value:

$$f(0) = argmin() \sum_{i=1}^{n} \alpha(y, r)$$

**Step 2.** For m = 1 to M:

1. Compute the pseudo-residuals:

   $$\alpha imb = -[(\alpha L(y_i F(X_i)))/(\alpha F(X_i))]$$

2. Fit to a base derivative on hm(x)

   $$i/p(X_i, R_i)$$

**Step 3.** Use the sum:

$$\sum_{i=1}^{n} L(y_i Fm_{-1}(X_i + \alpha))$$

**Step 4.** Update the model:

$$Fm(x) = Fm_1(x) + \alpha_{imb} hm(x)$$

## Development

The example uses a Spark-Shell enviroment, powered by Java Runtime and as a input language Scala is used. Spark is available in MacOS, Windows and Linux Distros.

The example below demostrates how to load a LibSVM file, parse it as an RDD of the column labeled pojnt and then perform classifications using decision tree with GilinImpurity as a measure, and lastly add a maximum tree depth of 5.

**Step 1.** Load the following libraries that will be used throughout the example, these are:

- ml.Pipeline
- ml.classification.GBTClassificationModel
- ml.classification.GBTClassifier
- ml.evaluation.MulticlassClassificationEvaluator
- ml.feature.IndexToString
- ml.feature.StringIndexer
- ml.feature.VectorIndexer

**Step 2.** Load and parse the data file into a DataFrame

```
val data = MLUtils.loadLibSVMFile(sc, "data/mllib/sample_libsvm_data.txt")
```

**Step 3.** Split the Data Frame into two parts (training_set, test_Set) 70% of the data held-out for training purposes and the rest for testing.

```
val splits = data.randomSplit(Array(0.7, 0.3))

val (trainingData, testData) = (splits(0), splits(1))
```

**Step 4.** Declare the default parameters for classification.

```
val boostingStrategy = BoostingStrategy.defaultParams("Classification")

boostingStrategy.numIterations = 3 // Note: Use more iterations in practice.

boostingStrategy.treeStrategy.numClasses = 2
boostingStrategy.treeStrategy.maxDepth = 5
```

**Step 5.** Empty categorical features information, it will indicate that all the features are continuous.

```
boostingStrategy.treeStrategy.categoricalFeaturesInfo = Map[Int, Int]()
```

**Step 6.** Train a GradientBoostedTree model.

```
val model = GradientBoostedTrees.train(trainingData, boostingStrategy)
```

**Step 7.** Evaluate the model with the test set and compute the test error.

```
val labelAndPreds = testData.map { point =>
    val prediction = model.predict(point.features)
    (point.label, prediction)
}
val testErr = labelAndPreds.filter(r => r._1 != r._2).count.toDouble / testData.count()
println(s"Test Error = $testErr")
println(s"Learned classification GBT model:\n ${model.toDebugString}")
```

## Conclusion

Gradient Boosted Tree Classifier is a machine learning model that is easy to implement and understand. The steps necessary to implement a model of this type for use are few and easy to interpret. In short, the procedure consisted of: loading libraries, loading the data and segmenting it in two, declaring the "boosting" parameters, training the model and finally training the model and evaluating it.

The mathematics related to this model is easy to understand if you have a basic understanding of differential calculus and algebra.