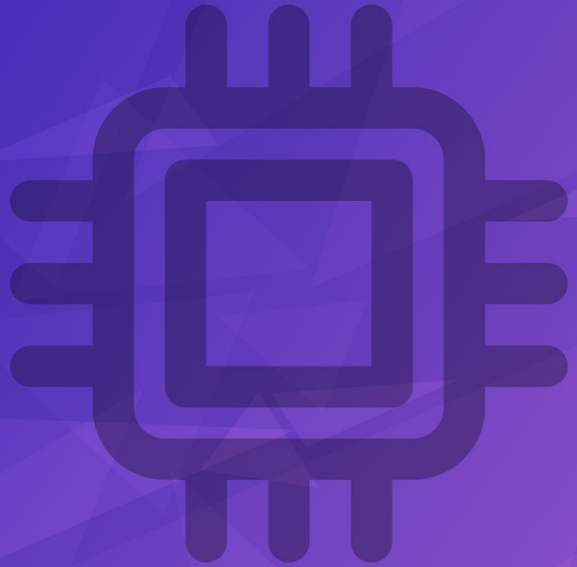
MACHINE LEARNING IN PRACTICE WITH SPARK MLlib: AN INTELLIGENT DATA ANALYZER.

STRATA+HADOOP WORLD

Eiti Kimura / Flávio Clésio
@Movile Brazil - Dec 2016



ABOUT US

Flávio Clésio

- Core *Machine Learning* at Movile
- MSc. in Production Engineering (Machine Learning in Credit Derivatives/NPL)
- Specialist in Database Engineering and Business Intelligence
- Blogger at *Mineração de Dados* (Data Mining) - http://mineracaodedados.wordpress.com

flavioclesio

ABOUT US

Eiti Kimura

- Software Architect and TI Coordinator at Movile
- Msc. in Electrical Engineering
- Apache Cassandra MVP (2014/2015 e 2015/2016)
- Apache Cassandra *Contributor* (2015)
- Cassandra Summit *Speaker* (2014 e 2015)
- Cassandra Summit *Reviewer* (2016)

eitikimura

WE MAKE LIFE BETTER THROUGH OUR APPS

movile

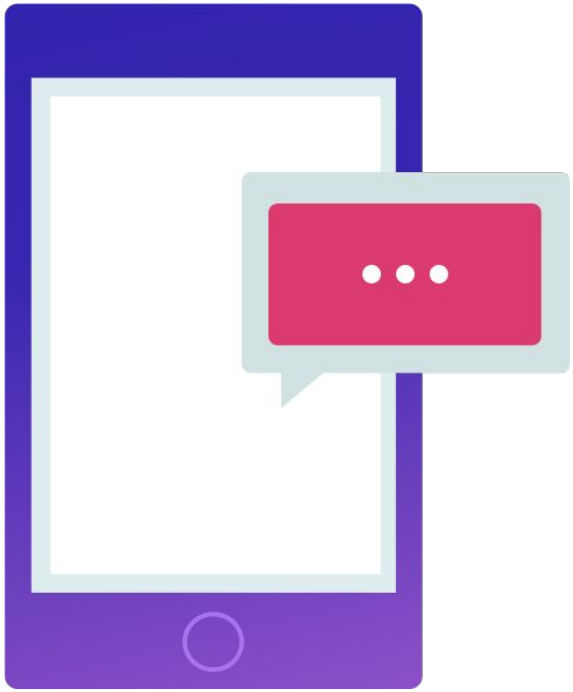
Movile is the company behind of several apps that makes the life easier

THE BEST CONTENT FOR KIDS

PlayKids is the #3 Top Grossing worldwide App for Children's. Kid Safe Toddler App.

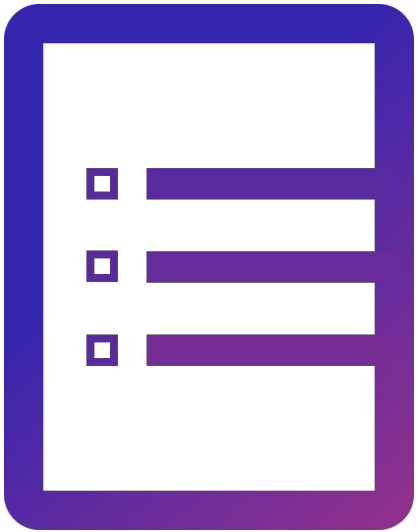
MESSAGING AND BILLING SERVICES FOR MOBILE CARRIERS

Huge case of SMS services for corporative and mobile content distribution.



AGENDA

- The Movile's Platform Case
- Regression Modeling: a bit of theory
- Practical Machine Learning Model Training
- Presenting Watcher-ai
- Results and Conclusions

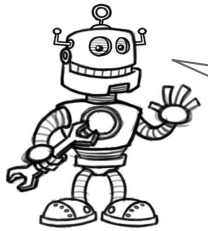


MOVILE'S SUBSCRIPTION AND BILLING PLATFORM IN ITS SIMPLEST FORM

- A distributed platform
- User's subscription management
- **MISSION CRITICAL** platform: can not stop under any circumstance

MAIN PROBLEM: MONITORING

How can we check if platform is fully functional based on data analysis only?



Tip: what if we ask help to an intelligent system?

HOW DATA LOOKS LIKE?

- 120 Millions + of billing requests attempt a day
- 4 main mobile carriers drive the operational work

carrier weight	date time	avg resp. time	succ. charges	no credit	general errors	total attempts
1	2016-10-31 0-8 pm	1014 ms	99.107	24.232.849	3.239.499	27.571.455
1	2016-11-01 0-8 pm	1204 ms	106.232	23.989.076	4.024.136	28.119.444
1	2016-11-02 0-8 pm	1186 ms	114.013	24.513.752	3.217.619	27.845.384
1	2016-11-03 0-8 pm	1117 ms	118.110	23.714.608	3.205.513	27.038.231
1	2016-11-04 0-8 pm	1138 ms	124.246	22.553.776	5.135.307	27.813.329
1	2016-11-05 0-8 pm	942 ms	102.674	23.556.432	4.072.168	27.731.274

DATA AND ALGORITHM MODELING

Sample of data (predicting the number of success)

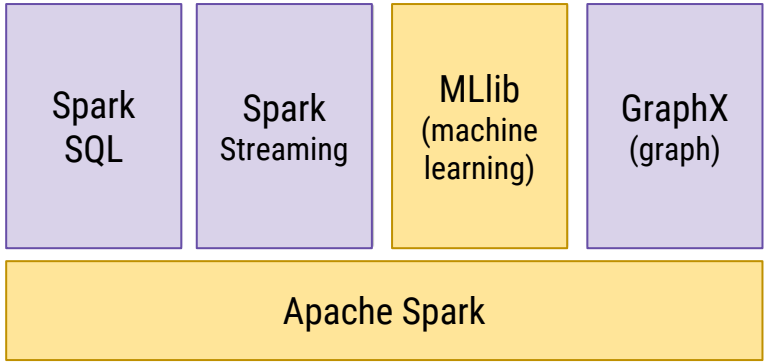
# success	carrier_weight	hour	week	response_time	#no_credit	#errors	# attempts
61.083,	[4.0,	17h,	3.0,	1259.0,	24.751.650,	2.193.67,	26.314.551]
label/target		features					

SUPERVISED LEARNING

Linear Regression

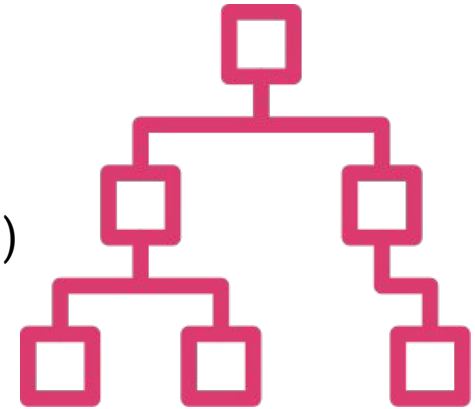
SPARK MLLIB

- MLlib is Apache Spark's scalable machine learning library.
- MLlib contains many algorithms and utilities, including Classification, Regression, Clustering, Recommendation, Pipelines and so on...



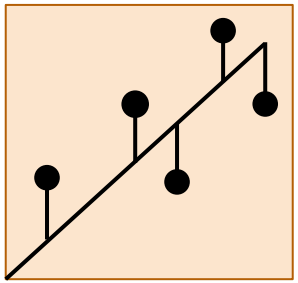
TESTED ALGORITHMS USING SPARK MLLIB

- Linear Model with Stochastic Gradient (SDG)
- Lasso with SGD Model (L1 Regularization)
- Ridge Regression with SGD Model (L2 Regularization)



REGRESSION MODELING: A BIT OF THEORY

- Linear regression it's a statistical method that investigates the relationship and interdependency between variables to get a numerical result.



Regression Algorithms

A LITTLE BIT OF MATH

$$y = \alpha + (\beta 1 * x1) + (\beta 2 * x2) + (\beta n * xn) + \varepsilon$$

as:

y = Value to be predicted (dependent variable)

α = Intercept (Where the slope gets the Y axis and the value of X is 0) - Endogenous Factors

β = Regression Coefficients

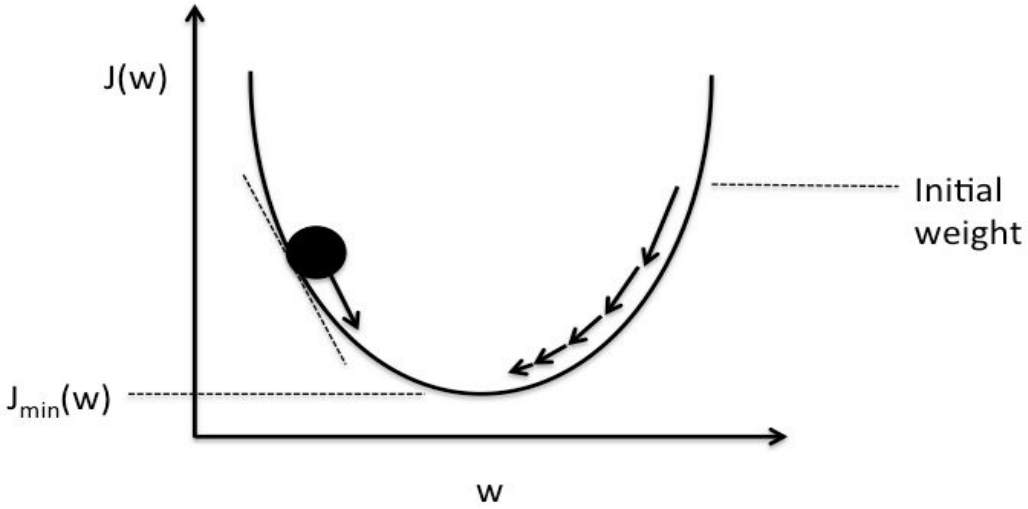
x1...xn = Values of independent variables (*e.g.* columns of a database)

ε = Random noise, non explicit errors - Exogenous Factors

LET'S TALK ABOUT REGULARIZATION

- To avoid the overfitting problem **Spark MLlib** embed some regularization methods like LASSO (L1) and Ridge (L2).
- LASSO (L1) regularization have as feature a penalty adds the penalty equivalent to the absolute value of the magnitude of the coefficients.
- Ridge L2) regularization method the penalty is equivalent to the magnitude of the coefficients raised to the square.

STOCHASTIC GRADIENT DESCENT



Schematic of gradient descent.

THINK ABOUT A TAILOR MADE SUIT

STOCHASTIC GRADIENT DESCENT

- Example: Stochastic Gradient Descent (Tailoring a Suit, from Quora)

1)Tailor makes initial estimate. *(See the parameters of the model)*

2) A random guy (or a subset of the full group) tries the **suit** and gives feedback.

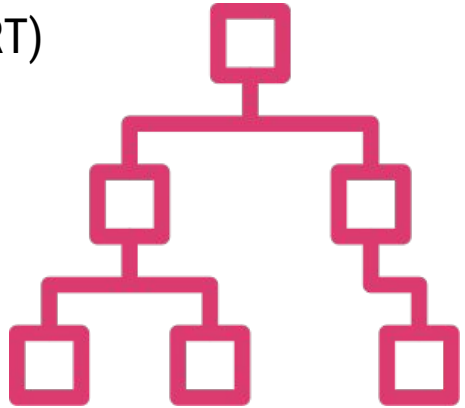
(take a sample of dataset)

3) Make a small adjustment according to feedback. *(change to reduce the error of the model)*

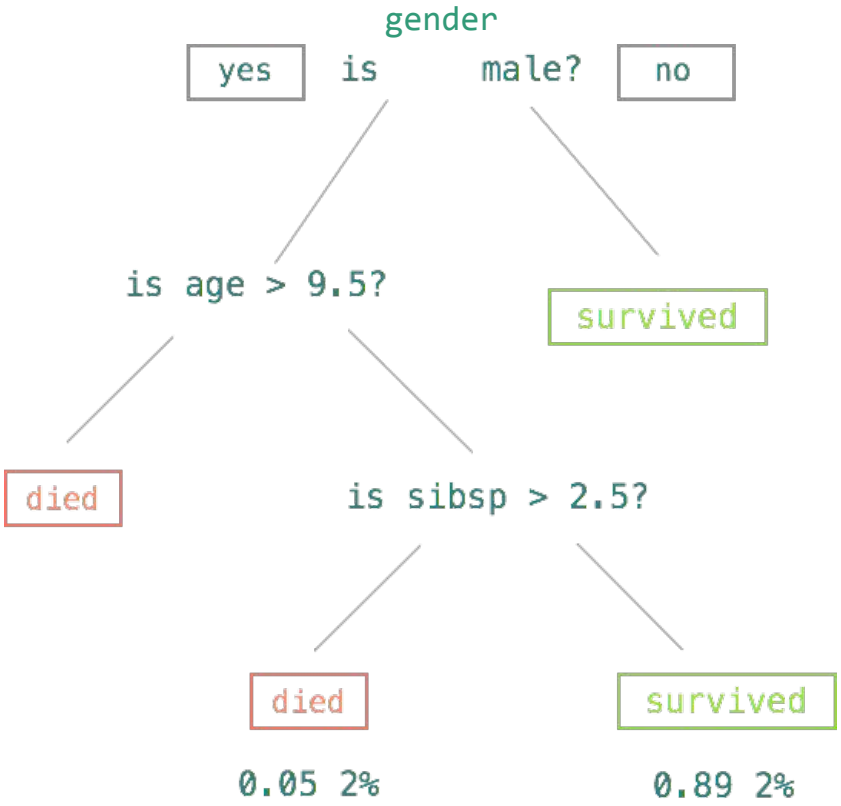
4) While the tailor still have time for this, he goes to 2. *(iterate and repeat the process)*

TESTED ALGORITHMS

- Decision Tree with Regression Properties (Not CART)

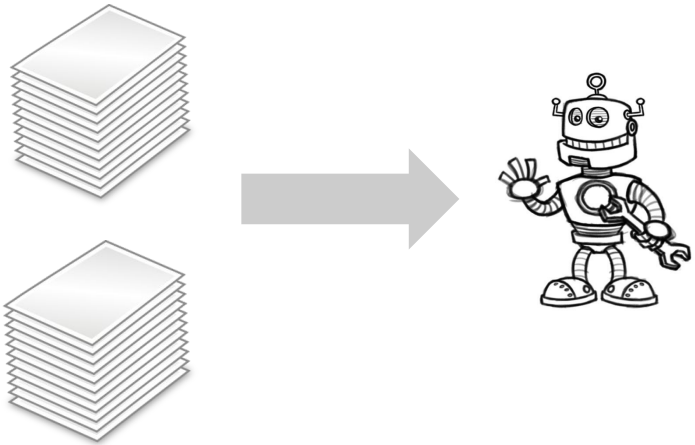


REGRESSION TREE



MODELING LIFECYCLE STEP-BY-STEP

Dataset



SPARK NOTEBOOK

http://spark-notebook.io/

SPARK NOTEBOOK Regression-Benchmarks

Scala [2.10.4] Spark [1.6.0] Hadoop [2.2.0] {Hive ✓} {Parquet ✓}

File Edit View Insert Cell Kernel Help

Code

Cell Toolbar: None

In [ ]:
```
def buildLabelValue(list: List[Double]) : Double = {
  // index = 4 is the number of success of the hour,
  return if (list(4) != 0.0) Math.log(list(4)) else
}

def buildFeatures(list: List[Double]) : List[Double]
   // remove the index 4, which means the number of
   return list.patch(4, Nil, 1)
}
```

In [ ]:
```
// reading pre processed dataset
val rdd = sc.objectFile[List[Double]](ROOT_DIR +"/rc

// building the LabelPoint, using success as Label
val labelSet = rdd.map{l => val label = buildLabelVa
                            val features = buildFeat
                            LabeledPoint(label, Vect
rdd.take(10).foreach(println)
```

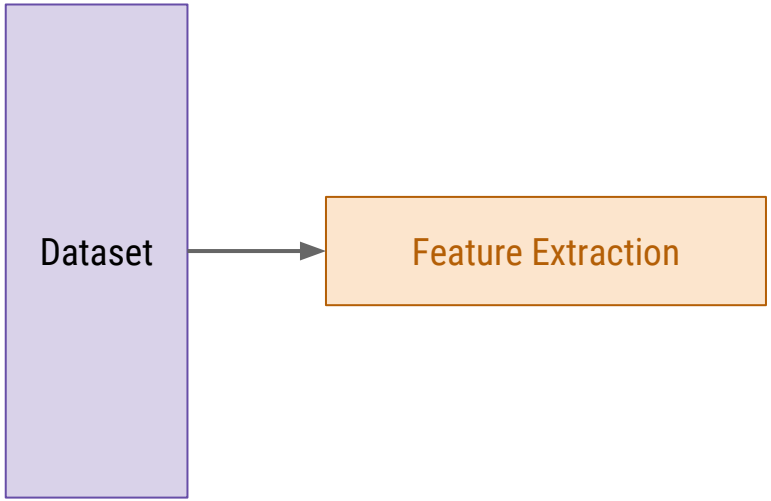
LOADING DATA

scala code snippet

```scala
// reading dataset
val rdd = sc.objectFile[List[Double]](ROOT_DIR + "/rdd-processed")
```

carrier_weight	hour	week	resp_time	# success	no_credit	# errors	# attempts
List(4.0,	17.0,	3.0,	1709.4,	39511.8,	2386316.3,	291279.6,	2717107.8)
List(2.0,	8.0,	5.0,	749.9,	51910.5,	1.27E7,	1951005.1,	1.47E7)
List(4.0,	11.0,	5.0,	1690.0,	18519.0,	562289.5,	173717.3,	754525.9)
List(2.0,	22.0,	1.0,	911.4,	257598.2,	4.05E7,	1.3E7,	5.4E7)
List(4.0,	7.0,	5.0,	1386.3,	1775.3,	391668.5,	75062.6,	468506.5)
List(1.0,	23.0,	4.0,	561.8,	195032.6,	2.8E7,	5279717.1,	3.41E7) ...

MODELING LIFECYCLE



FEATURE EXTRACTION

scala code snippet

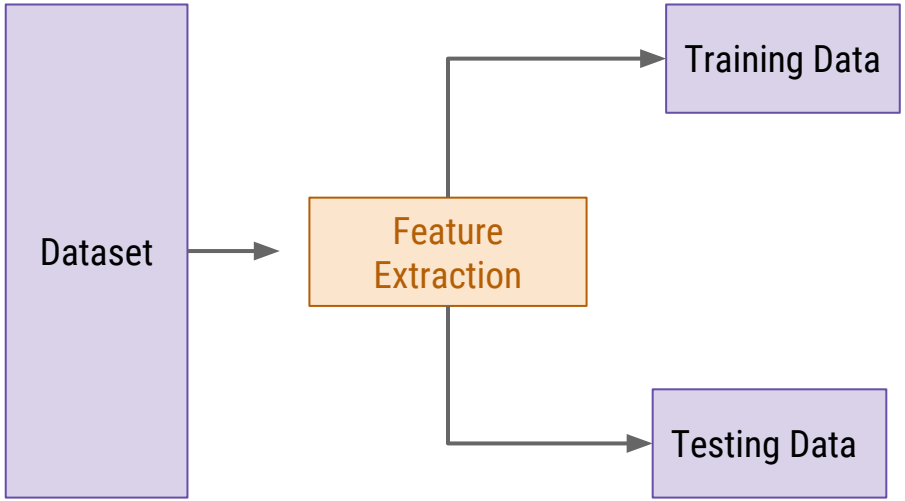
```scala
def buildLabelValue(list: List[Double]) : Double = {
  // index = 4 is the number of success, that is what we want to predict
  return if (list(4) != 0.0) Math.log(list(4))else 0.0
}

def buildFeatures(list: List[Double]) : List[Double] = {
   // remove the index 4, which means the number of success
   return list.patch(4, Nil, 1)
}

// building the LabelPoint, using success as Label
val labelSet = rdd.map{l => val label = buildLabelValue(l)
                            val features = buildFeatures(l)
                            LabeledPoint(label, Vectors.dense(features.toArray))}
```

```
labelSet: RDD[org.apache.spark.mllib.regression.LabeledPoint]
```

MODELING LIFECYCLE



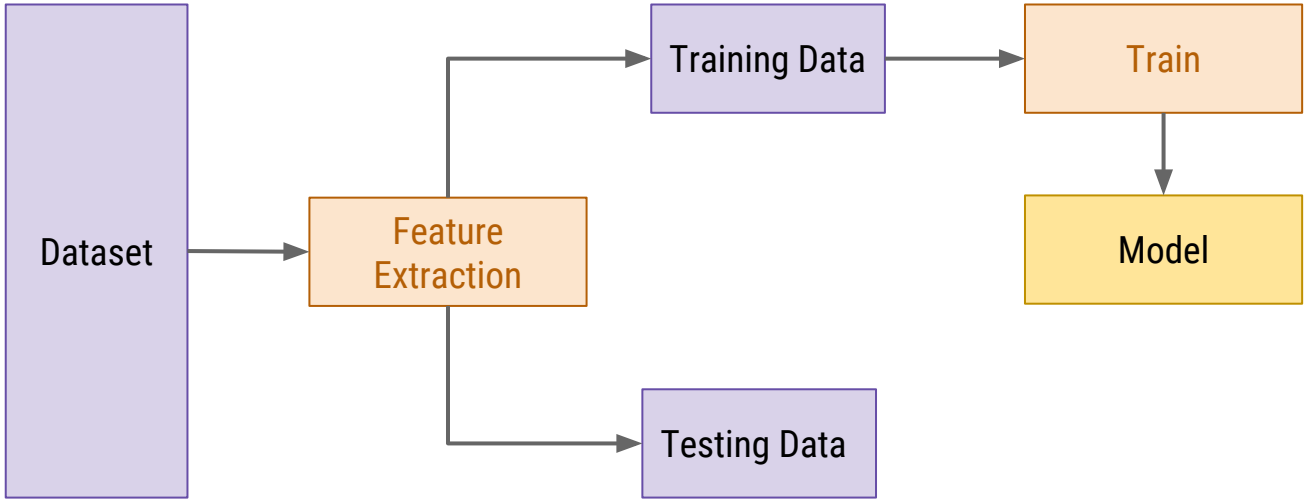
SPLITTING DATASET

scala code snippet

```
//Split data into training and test
val splits = labelSet.randomSplit(Array(0.70, 0.30), seed = 13L)
val training = splits(0)
val test = splits(1)
```

The main idea is to use 70% of data to train the model and 30% to evaluate the model performance

MODELING LIFECYCLE



standardizes

features by scaling

filtering

the dataset and grouping

AUXILIARY FUNCTIONS

scala code snippet

```
// 1 - standardizeTrainingSet
val scaler = new StandardScaler(withMean = false, withStd = true)
                    .fit(rdd.map(x => x.features))
// carrier weight list for filtering
val range = List(1.0, 2.0, 4.0, 5.0)
```

```
// 2 - filterTrainingSet
range.map{carrierWeight =>
          val trainingSet = rdd.filter(l => l.features.apply(0) == carrierWeight)
                               .map(x => LabeledPoint(x.label,
                                         scaler.transform(x.features)))
          (idx, trainingSet)
       }.toMap
```

training

Linear Regression SGD

Ridge Regression SGD

Lasso Regression SGD

Decision Tree w/ Regression

the champion is...

TRAINING: LINEAR REGRESSION WITH SGD

scala code snippet

```
def buildSGDModelMap(rdd:Map[Double, RDD[LabeledPoint]]) : Map[Double,
LinearRegressionModel] = {

  val carrierWeight = List(1.0, 2.0, 4.0, 5.0)
  return carrierWeight.map{idx =>
              // Building the model
              val numIterations = 100
              var regression = new LinearRegressionWithSGD().setIntercept(true)
              regression.optimizer.setStepSize(0.1)
              regression.optimizer.setNumIterations(numIterations)

              // get dataset for a specific carrier weight
              val dataset = rdd.get(idx).orNull;
              (idx, regression.run(dataset)) //<< training starts here
         }.toMap
}
```

```
val mapTraining = standardizeTrainingSet(training)
val mapSGDModel = buildSGDModelMap(mapTraining) //map with a model for each carrier
```

TRAINING: LASSO/RIDGE REGRESSION

scala code snippet

```
// instantiating the algorithm and setting the params
val regression = new LassoWithSGD()
regression.optimizer.setStepSize(0.1)
regression.optimizer.setNumIterations(100)

// training model
val model:LassoModel = regression.run(dataset)
```

```
// instantiating the algorithm and setting the params
val regression = new RidgeRegressionWithSGD()
regression.optimizer.setStepSize(0.1)
regression.optimizer.setNumIterations(100)

val model:RidgeModel = regression.run(dataset)
```

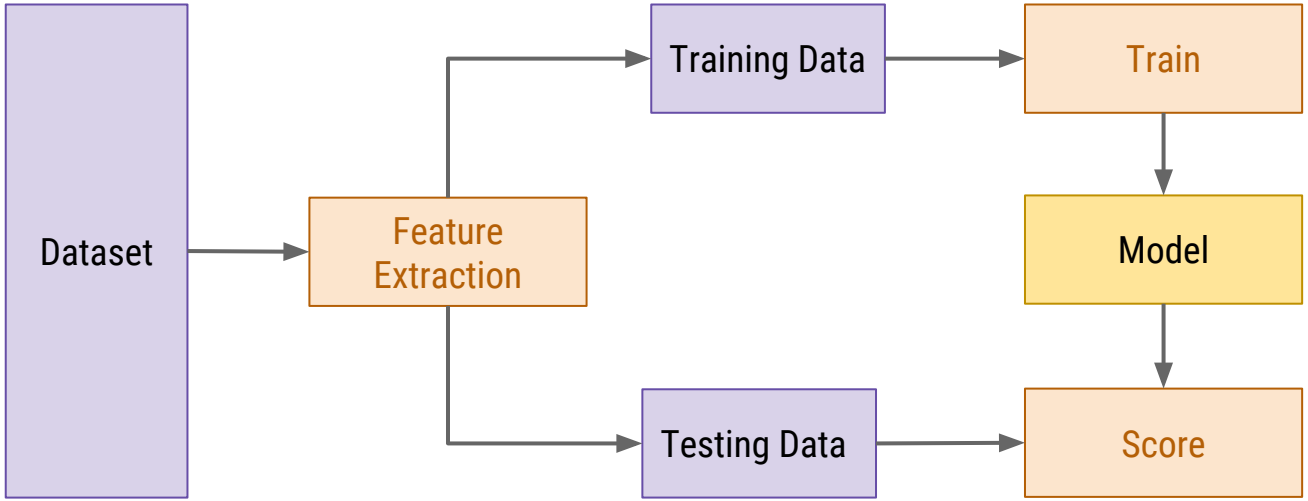
TRAINING: DECISION TREE WITH REGRESSION

scala code snippet

```
val categoricalFeaturesInfo = Map[Int, Int]()
val impurity = "variance" // indicates it is a regression tree
val maxDepth = 7  // the tree's max depth
val maxBins = 32  // max number of bins (groups)

val model = DecisionTree.trainRegressor(filteredSet, categoricalFeaturesInfo,
                                        impurity, maxDepth, maxBins);
```

MODELING LIFECYCLE



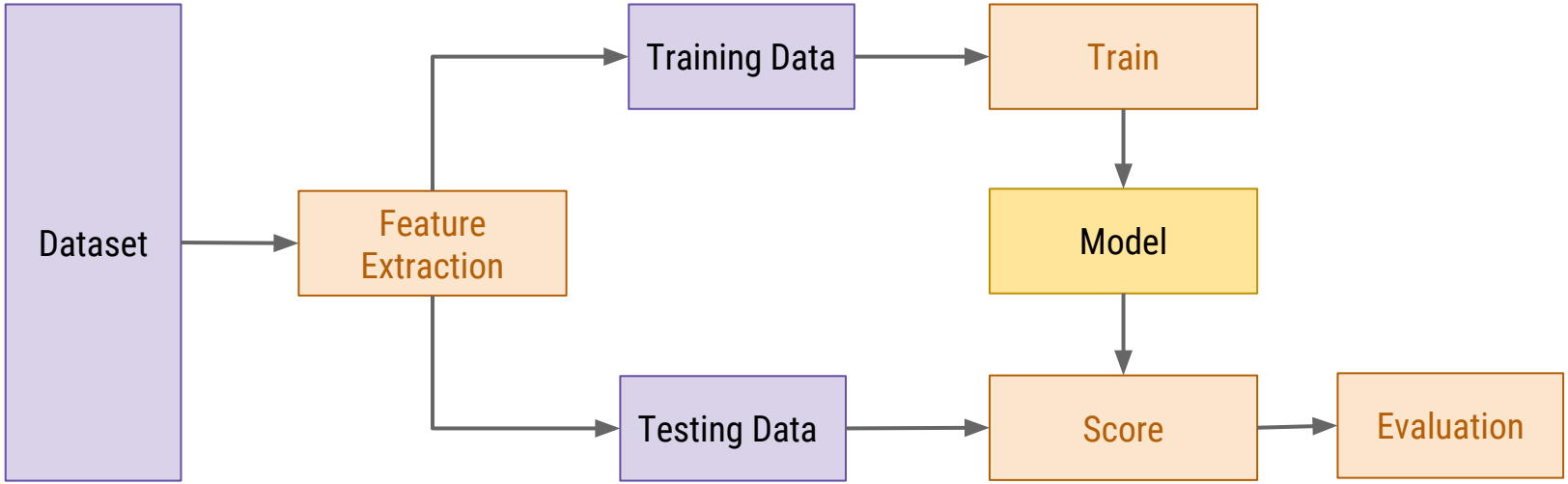
SCORE: AUXILIARY FUNCTIONS

scala code snippet

```
// predict values using the trained model
val labelsAndPredictions = test.map { point =>
  val carrier = point.features.apply(0) // 1st param is the carrier weight
  val model = getModelForCarrier(carrier)
  val prediction = model.predict(point.features) //<< prediction happens here
  (point.label, prediction)
}
```

Expected Value	Predicted Value
(12.196509845132933,	9.97275651498185),
(11.956245114516188,	11.632408901614912),
(11.840353189256883,	10.02762098460309),
(12.130296102598983,	9.320716165463033),
(11.682180075417563,	11.503286266374285),
(12.094170705574166,	13.010471821918166),
(11.832490497556401,	6.910122430921404) ...

MODELING LIFECYCLE



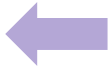
MODEL EVALUATION CRITERIA

- OMTM (One Metric That Matters)
- Root Mean Squared Error (RMSE)

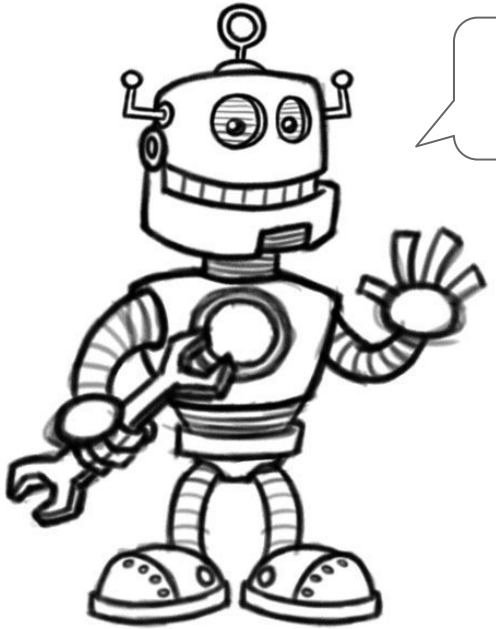
$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

TIME TO EVALUATE THE RESULTS

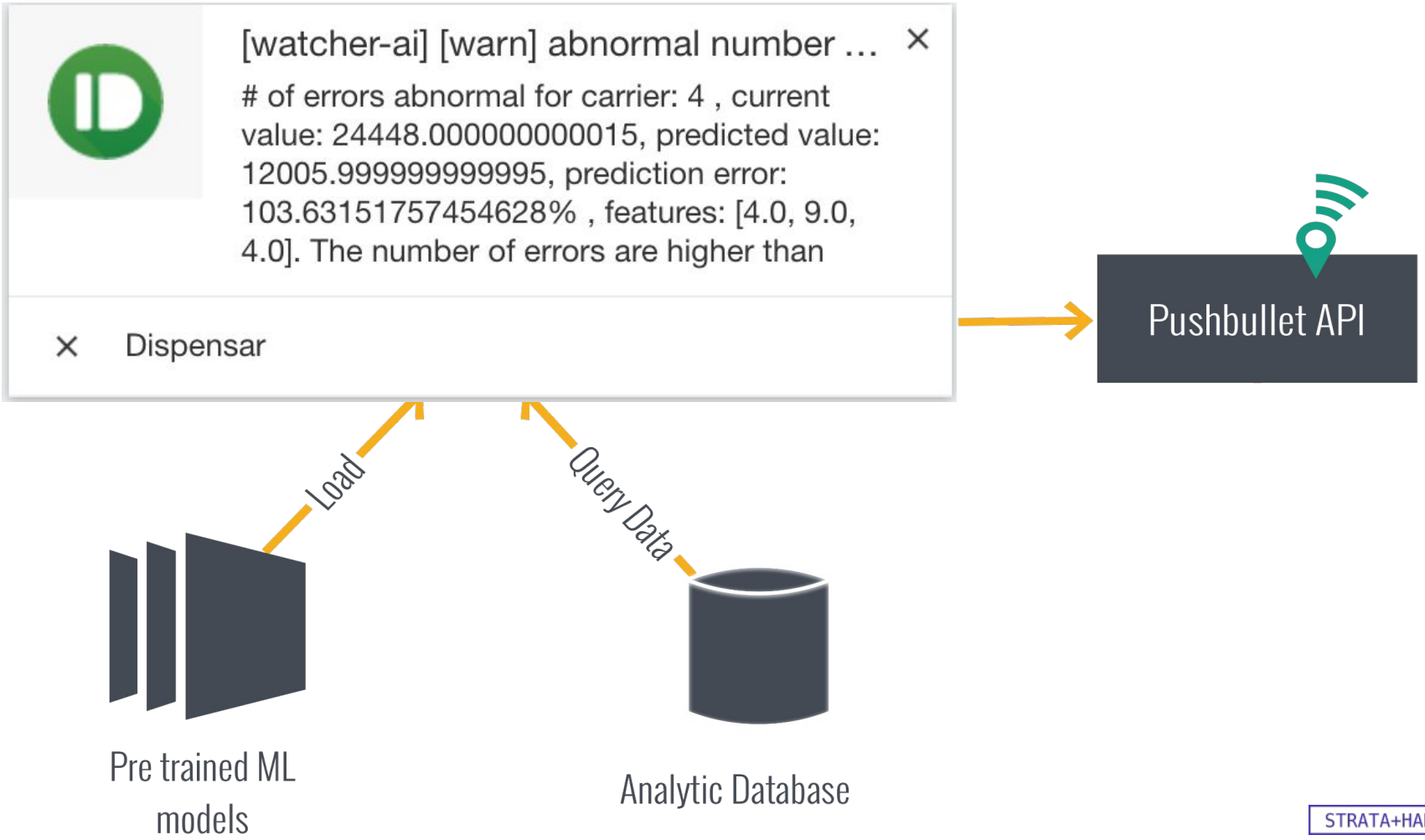
Machine Learning Tested Model	Accuracy	RMSE
Linear Model With SGD	44%	1.64
Lasso with SGD Model	43%	1.65
Ridge Regression with SGD Model	43%	1.66
Decision Tree Model	98%	0.20

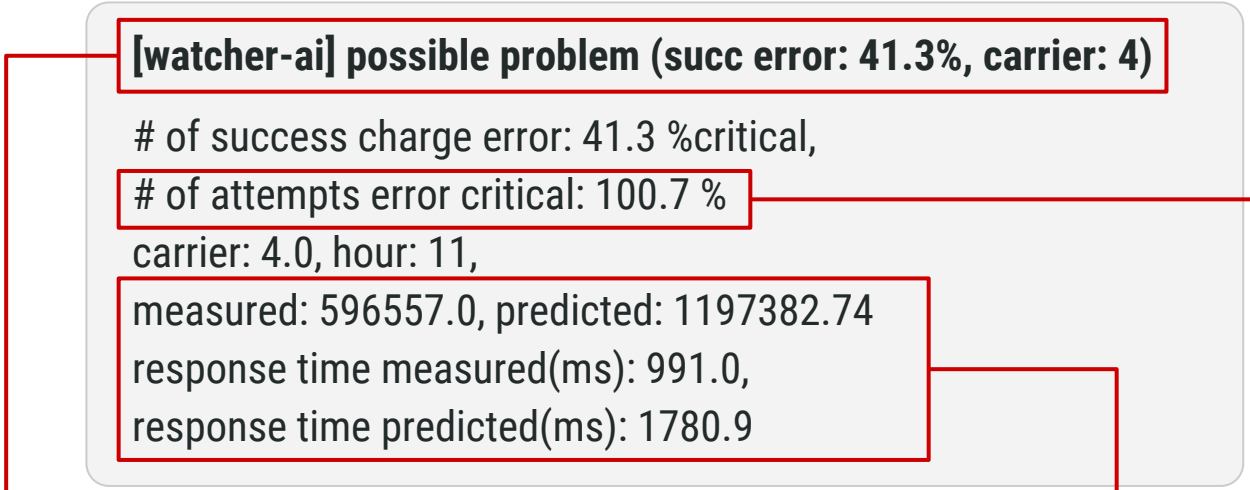


WATCHER-AI INTRODUCTION



Hi I'm Watcher-ai!
It is nice to see you here


[watcher-ai] [warn] abnormal number …
# of errors abnormal for carrier: 4 , current value: 24448.000000000015, predicted value: 12005.999999999995, prediction error: 103.63151757454628% , features: [4.0, 9.0, 4.0]. The number of errors are higher than
Dispensar
Pushbullet API
Load
Query Data
Pre trained ML models
Analytic Database



% of calculated number of attempts prediction error

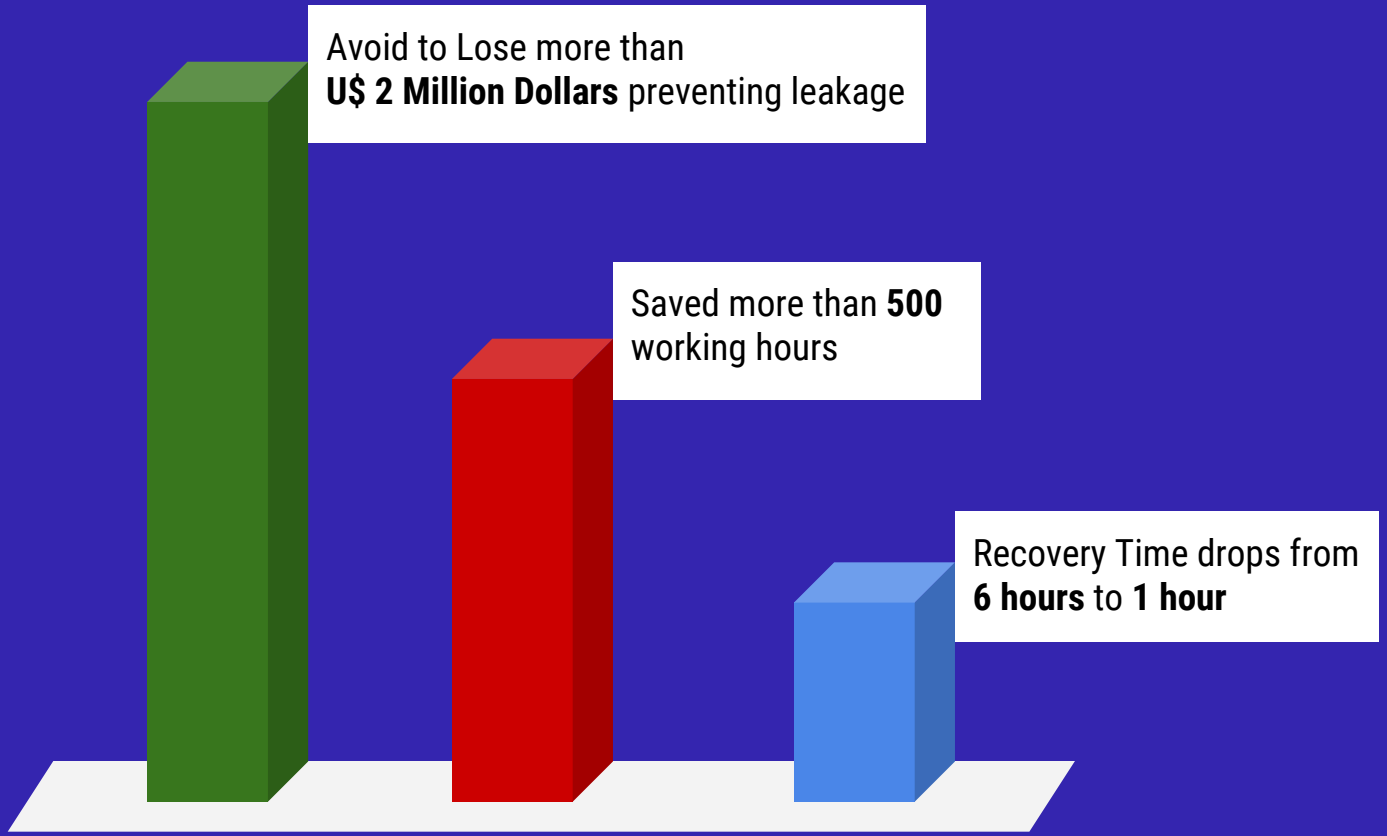
informs % of error and carrier weight

measured and predicted values for number of attempts and response time

MOVING TO *results*

PRELIMINARY RESULTS

- Was designed to be *Last barrier of defense* system
- Helped to catch troubles in the last 9 months
- Brought light in several problems of monitoring at Movile
- Catch any discrepancy in hourly fashion


Avoid to Lose more than
U$ 2 Million Dollars preventing leakage
Saved more than 500
working hours
Recovery Time drops from
6 hours to 1 hour

IT'S JUST THE *beginning*

It isn't the end.

IT IS JUST A WARM UP

- Automatic refeed and training using collected data. Analyse more data to predict possible errors with carrier
- Notify more people and specific teams (more complex problems)
- Refactory to be more generic, so other teams can add their own algorithm
- Why not interact with Watcher to guide the analysis?
- Don't limit your mind, there is a lot to keep improving...

THANK YOU!

STRATA+HADOOP WORLD

github.com/fclesio/watcher-ai-samples

@eitikimura

eiti-kimura-movile

eiti.kimura@movile.com

@flavioclesio

fclesio

flavio.clesio@movile.com

design by Tiago Cardoso