FINISHED

```
%dep

z.reset()
z.load("joda-time:joda-time:2.9.1")
```

```
DepInterpreter(%dep) deprecated. Remove dependencies and repositories through GUI interpreter
 menu instead.
DepInterpreter(%dep) deprecated. Load dependency through GUI interpreter menu instead.
res0: org.apache.zeppelin.dep.Dependency = org.apache.zeppelin.dep.Dependency@33f8cf7a
```

Took 11 sec. Last updated by anonymous at February 03 2017, 7:59:46 PM.

FINISHED

```
%spark

import org.apache.spark.rdd._
import scala.collection.JavaConverters._
import au.com.bytecode.opencsv.CSVReader
```

```
import org.apache.spark.rdd._
import scala.collection.JavaConverters._
import au.com.bytecode.opencsv.CSVReader
```

Took 2 sec. Last updated by anonymous at February 03 2017, 8:00:12 PM.

FINISHED

```
import java.io._
import org.joda.time._
import org.joda.time.format._
import org.joda.time.format.DateTimeFormat
import org.joda.time.DateTime
import org.joda.time.Days
```

```
import java.io._
import org.joda.time._
import org.joda.time.format._
import org.joda.time.format.DateTimeFormat
import org.joda.time.DateTime
import org.joda.time.Days
```

Took 3 sec. Last updated by anonymous at February 03 2017, 8:00:23 PM.

FINISHED

```
case class DelayRec(year: String,
                    month: String,
                    dayOfMonth: String,
                    dayOfWeek: String,
                    crsDepTime: String,
                    depDelay: String,
                    origin: String,
                    distance: String,
                    cancelled: String) {
```

```
    val holidays = List("01/01/2007", "01/15/2007", "02/19/2007", "05/28/2007", "06/07/2007",
        "09/03/2007", "10/08/2007" ,"11/11/2007", "11/22/2007", "12/25/2007",
        "01/01/2008", "01/21/2008", "02/18/2008", "05/22/2008", "05/26/2008", "07/04/2008",
        "09/01/2008", "10/13/2008" ,"11/11/2008", "11/27/2008", "12/25/2008")

    def gen_features: (String, Array[Double]) = {
        val values = Array(
            depDelay.toDouble,
            month.toDouble,
            dayOfMonth.toDouble,
            dayOfWeek.toDouble,
            get_hour(crsDepTime).toDouble,
            distance.toDouble,
            days_from_nearest_holiday(year.toInt, month.toInt, dayOfMonth.toInt)
        )
        new Tuple2(to_date(year.toInt, month.toInt, dayOfMonth.toInt), values)
    }

    def get_hour(depTime: String) : String = "%04d".format(depTime.toInt).take(2)
    def to_date(year: Int, month: Int, day: Int) = "%04d%02d%02d".format(year, month, day)

    def days_from_nearest_holiday(year:Int, month:Int, day:Int): Int = {
        val sampleDate = new org.joda.time.DateTime(year, month, day, 0, 0)

        holidays.foldLeft(3000) { (r, c) =>
        val holiday = org.joda.time.format.DateTimeFormat.forPattern("MM/dd/yyyy").parseDateT
        val distance = Math.abs(org.joda.time.Days.daysBetween(holiday, sampleDate).getDays)
        math.min(r, distance)
     }
    }
}
```

```
defined class DelayRec
```

Took 2 sec. Last updated by anonymous at February 03 2017, 8:00:29 PM. (outdated)

FINISHED

```
// function to do a preprocessing step for a given file
def prepFlightDelays(infile: String): RDD[DelayRec] = {
    val data = sc.textFile(infile)

    data.map { line =>
      val reader = new CSVReader(new StringReader(line))
      reader.readAll().asScala.toList.map(rec => DelayRec(rec(0),rec(1),rec(2),rec(3),rec(5),r
    }.map(list => list(0))
    .filter(rec => rec.year != "Year")
    .filter(rec => rec.cancelled == "0")
    .filter(rec => rec.origin == "ORD")
}
```

```
prepFlightDelays: (infile: String)org.apache.spark.rdd.RDD[DelayRec]
```

Took 2 sec. Last updated by anonymous at February 03 2017, 8:00:45 PM.

FINISHED

```
val data_2007tmp = prepFlightDelays("/Users/datascienceadmin/Downloads/flights_2007.csv.bz2")
val data_2007 = data_2007tmp.map(rec => rec.gen_features._2)
val data_2008 = prepFlightDelays("/Users/datascienceadmin/Downloads/flights_2008.csv.bz2").map

data_2007tmp.toDF().registerTempTable("data_2007tmp")
```

```
data_2007.take(5).map(x => x.mkString(" ")).foreach(println)
```

```
data_2007tmp: org.apache.spark.rdd.RDD[DelayRec] = MapPartitionsRDD[6] at filter at <console>:
58
data_2007: org.apache.spark.rdd.RDD[Array[Double]] = MapPartitionsRDD[7] at map at <console>:5
2
data_2008: org.apache.spark.rdd.RDD[Array[Double]] = MapPartitionsRDD[15] at map at <console>:
50
warning: there was one deprecation warning; re-run with -deprecation for details
-8.0,1.0,25.0,4.0,11.0,719.0,10.0
41.0,1.0,28.0,7.0,15.0,925.0,13.0
45.0,1.0,29.0,1.0,20.0,316.0,14.0
-9.0,1.0,17.0,3.0,19.0,719.0,2.0
180.0,1.0,12.0,5.0,17.0,316.0,3.0
```

Took 22 sec. Last updated by anonymous at February 03 2017, 8:01:36 PM.

FINISHED

```
%sql
select dayofWeek, case when depDelay > 15 then 'delayed' else 'ok' end , count(1)
from data_2007tmp group by dayofweek , case when depDelay > 15 then 'delayed' else 'ok' end
```

dayofWeek	CASE WHEN (CAST(depDelay AS DOUBLE) > CAST(15 AS DOUBLE)) THEN delayed ELS
3	ok
4	delayed
3	delayed
5	ok
2	ok
6	ok
4	ok
5	delayed
7	delayed

Took 55 sec. Last updated by anonymous at February 03 2017, 8:03:59 PM.

FINISHED

```
%sql select cast( cast(crsDepTime as int) / 100 as int) as hour,  case when depDelay > 15 then
count(1) as count from  data_2007tmp group by  cast( cast(crsDepTime as int) / 100 as int),
else 'ok' end
```

hour	delay	cou
12	ok	13,2
13	ok	20,9
20	delayed	10,5
10	ok	17,8
19	ok	12,7
15	ok	14,5
15	delayed	7,70
21	ok	8,04
8	ok	20,4

Took 1 min 0 sec. Last updated by anonymous at February 03 2017, 8:06:21 PM.

FINISHED

```
%spark

import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.feature.StandardScaler
```

```
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.feature.StandardScaler
```

Took 1 sec. Last updated by anonymous at February 03 2017, 8:06:34 PM. (outdated)

FINISHED

```
def parseData(vals: Array[Double]): LabeledPoint = {
    LabeledPoint(if (vals(0)>=15) 1.0 else 0.0, Vectors.dense(vals.drop(1)))
  }
```

```
parseData: (vals: Array[Double])org.apache.spark.mllib.regression.LabeledPoint
```

Took 1 sec. Last updated by anonymous at February 03 2017, 8:06:40 PM. (outdated)

FINISHED

```
// Prepare training set
val parsedTrainData = data_2007.map(parseData)
parsedTrainData.cache
val scaler = new StandardScaler(withMean = true, withStd = true).fit(parsedTrainData.map(x =>
val scaledTrainData = parsedTrainData.map(x => LabeledPoint(x.label, scaler.transform(Vectors
scaledTrainData.cache
```

```
parsedTrainData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = Ma
pPartitionsRDD[49] at map at <console>:60
res11: parsedTrainData.type = MapPartitionsRDD[49] at map at <console>:60
scaler: org.apache.spark.mllib.feature.StandardScalerModel = org.apache.spark.mllib.feature.St
andardScalerModel@323dd8b6
scaledTrainData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = Ma
pPartitionsRDD[52] at map at <console>:63
res12: scaledTrainData.type = MapPartitionsRDD[52] at map at <console>:63
```

Took 1 min 1 sec. Last updated by anonymous at February 03 2017, 8:07:50 PM.

FINISHED

```
// Prepare test/validation set
val parsedTestData = data_2008.map(parseData)
parsedTestData.cache
val scaledTestData = parsedTestData.map(x => LabeledPoint(x.label, scaler.transform(Vectors.de
scaledTestData.cache
```

```
parsedTestData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = Map
PartitionsRDD[53] at map at <console>:58
res15: parsedTestData.type = MapPartitionsRDD[53] at map at <console>:58
scaledTestData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = Map
PartitionsRDD[54] at map at <console>:67
res16: scaledTestData.type = MapPartitionsRDD[54] at map at <console>:67
```

Took 2 sec. Last updated by anonymous at February 03 2017, 8:08:00 PM.

FINISHED

```
scaledTrainData.take(3).map(x => (x.label, x.features)).foreach(println)
```

```
(0.0,[-1.6160463330366632,1.0549272994666004,0.03217026353736743,-0.518924417544128,0.03408393
342430724,-0.28016830994663705])
(1.0,[-1.6160463330366632,1.3961052168540338,1.5354307758475594,0.362432098412101,0.4316551188
434341,-0.023273887437333846])
(1.0,[-1.6160463330366632,1.5098311893165115,-1.4710902487728246,1.4641277433573872,-0.7436888
225169944,0.062357586732433884])
```

Took 1 sec. Last updated by anonymous at February 03 2017, 8:08:08 PM.

FINISHED

```
// Function to compute evaluation metrics
def eval_metrics(labelsAndPreds: RDD[(Double, Double)]) : Tuple2[Array[Double], Array[Double]]
    val tp = labelsAndPreds.filter(r => r._1==1 && r._2==1).count.toDouble
    val tn = labelsAndPreds.filter(r => r._1==0 && r._2==0).count.toDouble
    val fp = labelsAndPreds.filter(r => r._1==1 && r._2==0).count.toDouble
    val fn = labelsAndPreds.filter(r => r._1==0 && r._2==1).count.toDouble

    val precision = tp / (tp+fp)
    val recall = tp / (tp+fn)
    val F_measure = 2*precision*recall / (precision+recall)
    val accuracy = (tp+tn) / (tp+tn+fp+fn)
    new Tuple2(Array(tp, tn, fp, fn), Array(precision, recall, F_measure, accuracy))

}
```

```
eval_metrics: (labelsAndPreds: org.apache.spark.rdd.RDD[(Double, Double)])(Array[Double], Arra
y[Double])
```

Took 1 sec. Last updated by anonymous at February 03 2017, 8:08:21 PM.

FINISHED

```
import org.apache.spark.rdd._
import org.apache.spark.rdd.RDD
```

```
import org.apache.spark.rdd._
import org.apache.spark.rdd.RDD
```

Took 2 sec. Last updated by anonymous at February 03 2017, 8:20:51 PM.

FINISHED

```
class Metrics(labelsAndPreds: org.apache.spark.rdd.RDD[(Double, Double)]) extends java.io.Ser

    private def filterCount(lftBnd:Int,rtBnd:Int):Double = labelsAndPreds
                                                          .map(x => (x._1.toInt, x._2.toInt))
                                                          .filter(_ == (lftBnd,rtBnd)).count(

    lazy val tp = filterCount(1,1)  // true positives
    lazy val tn = filterCount(0,0)  // true negatives
    lazy val fp = filterCount(0,1)  // false positives
    lazy val fn = filterCount(1,0)  // false negatives
    lazy val precision = tp / (tp+fp)
    lazy val recall = tp / (tp+fn)
    lazy val F1 = 2*precision*recall / (precision+recall)
    lazy val accuracy = (tp+tn) / (tp+tn+fp+fn)
}
```

Lab 2 Untitled Untitled Untitled Untitled Untitled Untitled Untitled Untitled Untitled Untitled Untitled Untitled U

```
defined class Metrics
```

Took 0 sec. Last updated by anonymous at February 03 2017, 8:08:39 PM.

FINISHED

```
%spark

import org.apache.spark.mllib.classification.LogisticRegressionWithSGD

// Build the Logistic Regression model
val model_lr = LogisticRegressionWithSGD.train(scaledTrainData, numIterations=100)

// Predict
val labelsAndPreds_lr = scaledTestData.map { point =>
    val pred = model_lr.predict(point.features)
    (pred, point.label)
}
val m_lr = eval_metrics(labelsAndPreds_lr)._2
println("precision = %.2f, recall = %.2f, F1 = %.2f, accuracy = %.2f".format(m_lr(0), m_lr(1)
```

```
import org.apache.spark.mllib.classification.LogisticRegressionWithSGD
warning: there was one deprecation warning; re-run with -deprecation for details
model_lr: org.apache.spark.mllib.classification.LogisticRegressionModel = org.apache.spark.mll
ib.classification.LogisticRegressionModel: intercept = 0.0, numFeatures = 6, numClasses = 2, t
hreshold = 0.5
labelsAndPreds_lr: org.apache.spark.rdd.RDD[(Double, Double)] = MapPartitionsRDD[140] at map a
t <console>:78
m_lr: Array[Double] = Array(0.3735363068960268, 0.6427763108261033, 0.47249298123322336, 0.591
5277487847792)
precision = 0.37, recall = 0.64, F1 = 0.47, accuracy = 0.59
```

Took 1 min 2 sec. Last updated by anonymous at February 03 2017, 8:09:47 PM.

FINISHED

```
println(model_lr.weights)
```

```
[-0.05519239973775392,0.0058773883559942045,-0.03625359858318008,0.3903949271784436,0.04994314
670964247,7.940537333813815E-4]
```

Took 0 sec. Last updated by anonymous at February 03 2017, 8:09:50 PM.

FINISHED

```
%spark
```

```
import org.apache.spark.mllib.tree.DecisionTree

// Build the Decision Tree model
val numClasses = 2
val categoricalFeaturesInfo = Map[Int, Int]()
val impurity = "gini"
val maxDepth = 10
val maxBins = 100
val model_dt = DecisionTree.trainClassifier(parsedTrainData, numClasses, categoricalFeaturesI
```

Lab 2

Zeppelin

Lab 2

Predict Untitled Untitled Untitled Untitled Untitled Untitled Untitled Untitled Untitled Untitled Untitled U

```
// Predict
val labelsAndPreds_dt = parsedTestData.map { point =>
val pred = model_dt.predict(point.features)
(pred, point.label)
}
val m_dt = eval_metrics(labelsAndPreds_dt).2
println("precision = %.2f, recall = %.2f, F1 = %.2f, accuracy = %.2f".format(m_dt(0), m_dt(1)
```

default

```
import org.apache.spark.mllib.tree.DecisionTree
numClasses: Int = 2
categoricalFeaturesInfo: scala.collection.immutable.Map[Int,Int] = Map()
impurity: String = gini
maxDepth: Int = 10
maxBins: Int = 100
model_dt: org.apache.spark.mllib.tree.model.DecisionTreeModel = DecisionTreeModel classifier o
f depth 10 with 1845 nodes
labelsAndPreds_dt: org.apache.spark.rdd.RDD[(Double, Double)] = MapPartitionsRDD[184] at map a
t <console>:83
m_dt: Array[Double] = Array(0.40568143388569494, 0.25139360409069955, 0.31042335161991513, 0.6
821280529627531)
precision = 0.41, recall = 0.25, F1 = 0.31, accuracy = 0.68
```

Took 8 sec. Last updated by anonymous at February 03 2017, 8:10:03 PM.

FINISHED

```
%spark

import org.apache.spark.mllib.tree.RandomForest
import org.apache.spark.mllib.tree.configuration.Strategy

val treeStrategy = Strategy.defaultStrategy("Classification")
val numTrees = 20
val featureSubsetStrategy = "auto" // Let the algorithm choose
val model_rf = RandomForest.trainClassifier(parsedTrainData, treeStrategy, numTrees, featureSu
```

```
import org.apache.spark.mllib.tree.RandomForest
import org.apache.spark.mllib.tree.configuration.Strategy
treeStrategy: org.apache.spark.mllib.tree.configuration.Strategy = org.apache.spark.mllib.tre
e.configuration.Strategy@2f8f7ebe
numTrees: Int = 20
featureSubsetStrategy: String = auto
model_rf: org.apache.spark.mllib.tree.model.RandomForestModel =
TreeEnsembleModel classifier with 20 trees
```

Took 20 sec. Last updated by anonymous at February 03 2017, 8:10:50 PM.

FINISHED

```
%spark
```

```
// Predict
val labelsAndPreds_rf = parsedTestData.map { point =>
    val pred = model_rf.predict(point.features)
    (point.label, pred)
}

val m_rf = new Metrics(labelsAndPreds_rf)
println("precision = %.2f, recall = %.2f, F1 = %.2f, accuracy = %.2f"
        .format(m_rf.precision, m_rf.recall, m_rf.F1, m_rf.accuracy))
```

```
labelsAndPreds_rf: org.apache.spark.rdd.RDD[(Double, Double)] = MapPartitionsRDD[228] at map a
t <console>:81
m_rf: Metrics = Metrics@69986c10
precision = 0.49, recall = 0.16, F1 = 0.24, accuracy = 0.71
```

Took 8 sec. Last updated by anonymous at February 03 2017, 8:11:04 PM.

default

Zeppelin

Lab 2

ERROR

```
%spark

// import org.apache.spark.rdd._
 import org.apache.spark.rdd.RDD
import org.apache.spark.SparkContext._
import scala.collection.JavaConverters._
import au.com.bytecode.opencsv.CSVReader
import java.io._

// function to do a preprocessing step for a given file

def preprocess_spark(delay_file: String, weather_file: String): RDD[Array[Double]] = {
  // Read wether data
  val delayRecs = prepFlightDelays(delay_file).map{ rec =>
        val features = rec.gen_features
        (features._1, features._2)
  }

  // Read weather data into RDDs
  val station_inx = 0
  val date_inx = 1
  val metric_inx = 2
  val value_inx = 3

  def filterMap(wdata:RDD[Array[String]], metric:String):RDD[(String,Double)] = {
    wdata.filter(vals => vals(metric_inx) == metric).map(vals => (vals(date_inx), vals(value_
  }

  val wdata = sc.textFile(weather_file).map(line => line.split(","))
                    .filter(vals => vals(station_inx) == "USW00094846")
  val w_tmin = filterMap(wdata,"TMIN")
  val w_tmax = filterMap(wdata,"TMAX")
  val w_prcp = filterMap(wdata,"PRCP")
  val w_snow = filterMap(wdata,"SNOW")
  val w_awnd = filterMap(wdata,"AWND")

  delayRecs.join(w_tmin).map(vals => (vals._1, vals._2._1 ++ Array(vals._2._2)))
           .join(w_tmax).map(vals => (vals._1, vals._2._1 ++ Array(vals._2._2)))
           .join(w_prcp).map(vals => (vals._1, vals._2._1 ++ Array(vals._2._2)))
           .join(w_snow).map(vals => (vals._1, vals._2._1 ++ Array(vals._2._2)))
           .join(w_awnd).map(vals => vals._2._1 ++ Array(vals._2._2))
}
```

```
val data_2007 = preprocess_spark("/Users/datascienceadmin/Downloads/flights_2007.csv.bz2", "/l
2007.csv.gz")
val data_2008 = preprocess_spark("/Users/datascienceadmin/Downloads/flights_2008.csv.bz2", "/l
2008.csv.gz")

data_2007.take(5).map(x => x mkString ",").foreach(println)
```

```
 at scala.Option.getOrElse(Option.scala:121)
 at org.apache.spark.rdd.RDD.partitions(RDD.scala:246)
 at org.apache.spark.Partitioner$$anonfun$2.apply(Partitioner.scala:58)
 at org.apache.spark.Partitioner$$anonfun$2.apply(Partitioner.scala:58)
 at scala.math.Ordering$$anon$5.compare(Ordering.scala:122)
 at java.util.TimSort.countRunAndMakeAscending(TimSort.java:355)
 at java.util.TimSort.sort(TimSort.java:220)
 at java.util.Arrays.sort(Arrays.java:1438)
 at scala.collection.SeqLike$class.sorted(SeqLike.scala:648)
 at scala.collection.AbstractSeq.sorted(Seq.scala:41)
 at scala.collection.SeqLike$class.sortBy(SeqLike.scala:623)
 at scala.collection.AbstractSeq.sortBy(Seq.scala:41)
 at org.apache.spark.Partitioner$.defaultPartitioner(Partitioner.scala:58)
 at org.apache.spark.rdd.PairRDDFunctions$$anonfun$join$2.apply(PairRDDFunctions.scala:652)
 at org.apache.spark.rdd.PairRDDFunctions$$anonfun$join$2.apply(PairRDDFunctions.scala:652)
 at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151)
 at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:112)
 at org.apache.spark.rdd.RDD.withScope(RDD.scala:358)
```

Took 6 sec. Last updated by anonymous at February 03 2017, 8:55:04 PM.

ERROR

```
%spark

import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.feature.StandardScaler

def parseData(vals: Array[Double]): LabeledPoint = {
  LabeledPoint(if (vals(0)>=15) 1.0 else 0.0, Vectors.dense(vals.drop(1)))
}

// Prepare training set
val parsedTrainData = data_2007.map(parseData)
val scaler = new StandardScaler(withMean = true, withStd = true).fit(parsedTrainData.map(x =>
val scaledTrainData = parsedTrainData.map(x => LabeledPoint(x.label, scaler.transform(Vectors
parsedTrainData.cache
scaledTrainData.cache

// Prepare test/validation set
val parsedTestData = data_2008.map(parseData)
val scaledTestData = parsedTestData.map(x => LabeledPoint(x.label, scaler.transform(Vectors.de
parsedTestData.cache
scaledTestData.cache

scaledTrainData.take(5).map(x => (x.label, x.features)).foreach(println)
```

```
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.feature.StandardScaler
parseData: (vals: Array[Double])org.apache.spark.mllib.regression.LabeledPoint
parsedTrainData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] =
 MapPartitionsRDD[705] at map at <console>:210
org.apache.spark.SparkException: Job aborted due to stage failure: Task 1 in stage 166.0 fai
led 1 times, most recent failure: Lost task 1.0 in stage 166.0 (TID 867, localhost): java.la
ng.ArrayIndexOutOfBoundsException: -1590091655
        at org.apache.hadoop.io.compress.bzip2.CBZip2InputStream.getAndMoveToFrontDecode0(CB
Zip2InputStream.java:1014)
        at org.apache.hadoop.io.compress.bzip2.CBZip2InputStream.getAndMoveToFrontDecode(CBZ
ip2InputStream.java:829)
        at org.apache.hadoop.io.compress.bzip2.CBZip2InputStream.initBlock(CBZip2InputStrea
m.java:504)
        at org.apache.hadoop.io.compress.bzip2.CBZip2InputStream.changeStateToProcessABlock
(CBZip2InputStream.java:333)
        at org.apache.hadoop.io.compress.bzip2.CBZip2InputStream.read(CBZip2InputStream.jav
```

Took 1 min 7 sec. Last updated by anonymous at February 03 2017, 8:58:19 PM.

READY

```
%spark

import org.apache.spark.mllib.classification.LogisticRegressionWithSGD

// Build the Logistic Regression model
val model_lr = LogisticRegressionWithSGD.train(scaledTrainData, numIterations=100)

// Predict
val labelsAndPreds_lr = scaledTestData.map { point =>
    val pred = model_lr.predict(point.features)
    (point.label, pred)
}
val m_lr = new Metrics(labelsAndPreds_lr)
println("precision = %.2f, recall = %.2f, F1 = %.2f, accuracy = %.2f"
        .format(m_lr.precision, m_lr.recall, m_lr.F1, m_lr.accuracy))
```

READY

```
%spark

println(model_lr.weights)
```

READY

```
 1 %spark
 2
 3 import org.apache.spark.mllib.tree.DecisionTree
 4
 5 // Build the Decision Tree model
 6 val numClasses = 2
 7 val categoricalFeaturesInfo = Map[Int, Int]()
 8 val impurity = "gini"
 9 val maxDepth = 10
10 val maxBins = 100
11 val model_dt = DecisionTree.trainClassifier(parsedTrainData, numClasses, categoricalFeat
12
```

```
// Predict
val labelsAndPreds_dt = parsedTestData.map { point =>
    val pred = model_dt.predict(point.features)
    (point.label, pred)
}
val m_dt = new Metrics(labelsAndPreds_dt)
println("precision = %.2f, recall = %.2f, F1 = %.2f, accuracy = %.2f"
        .format(m_dt.precision, m_dt.recall, m_dt.F1, m_dt.accuracy))
```

READY

```
%spark

import org.apache.spark.mllib.tree.RandomForest
import org.apache.spark.mllib.tree.configuration.Strategy

val treeStrategy = Strategy.defaultStrategy("Classification")
val model_rf = RandomForest.trainClassifier(parsedTrainData,
                                            treeStrategy,
                                            numTrees = 20,
                                            featureSubsetStrategy = "auto", seed = 125)

// Predict
val labelsAndPreds_rf = parsedTestData.map { point =>
    val pred = model_rf.predict(point.features)
    (point.label, pred)
}
val m_rf = new Metrics(labelsAndPreds_rf)
println("precision = %.2f, recall = %.2f, F1 = %.2f, accuracy = %.2f"
        .format(m_rf.precision, m_rf.recall, m_rf.F1, m_rf.accuracy))
```

READY