DECEMBER 6, 2017

PROJECT 2: NAÏVE BAYES

ANALYSIS USING SPARK ML SCALA & R

HARSHITA JASTHI
N16124341
Hj997

CONTENTS:

DESCRIPTION OF NAÏVE BAYES ALGORITHM:

Naive Bayes is a classification algorithm based on probability calculation and it is called *naive Bayes* or *idiot Bayes* because the calculation of the probabilities for each hypothesis are simplified to make their calculation tractable. Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Though independent features are unlikely on real data , the classifier performs better than most of the other sophisticated classifiers.

FORMULA FOR NAÏVE BAYES:

Bayes' Theorem is stated as:

$P(c|x) = (P(x|c) * P(c)) / P(x)$

Where

- **P(c|x)** is the probability of class c given the data x. This is called the posterior probability.
- *P*(*c*) is the prior probability of *class*.
- *P*(*x*|*c*) is the likelihood which is the probability of *predictor* given *class*.
- *P*(*x*) is the prior probability of *predictor*.

GENERATIVE CLASSFIER:

Naïve Bayes is a generative classifier because we calculate joint probability distribution P(X,Y) and learn conditional probabilities for the complete model, whereas discriminative models learn P(X|Y) therefore relying on the relationship between Y and features X directly from data and build soft/hard boundaries.

RESTRICTIONS:

Naïve Bayes works for both binary and multiclass variable.

If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency".

When feeding into the model it requires data to be numeric (double preferred).

DATASET:

The dataset description available below,

TITANIC DATASET:

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	F or M
Age	Age	in years
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Dataset Refinement: The name & ticket column were dropped, and converted categorial string variables to integer type, removed nulls .

Dataset was preprocess using R and export to csv as mydata.csv.

SCALA CODE SNIPPET:

```
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.mllib._
import org.apache.spark.mllib.classification.{NaiveBayes, NaiveBayesModel}
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.rdd.RDD



object NaiveBayesExample {

 def main(args:Array[String]) : Unit = {
  val sc = SparkContext.getOrCreate()

  val datapre = sc.textFile("Mydata.csv")
  val parseddatapre = datapre.map { line =>
   val parts = line.split(',')
   LabeledPoint(parts(1).toDouble, Vectors.dense(parts(0).toDouble, parts(2).toDouble, parts(3).toDouble, parts(4).toDouble,
parts(5).toDouble, parts(6).toDouble, parts(7).toDouble, parts(8).toDouble))
  }


  val splits = parseddatapre.randomSplit(Array(0.6, 0.4), seed = 11L)
  val training = splits(0)
  val test = splits(1)
  val model = NaiveBayes.train(training)
  val predictionAndLabel = test.map(p => (model.predict(p.features), p.label))
  val accuracy = 1.0 * predictionAndLabel.filter(x => x._1 == x._2).count() / test.count()

  println("the accuracy is " + accuracy)

  val metrics = new BinaryClassificationMetrics(predictionAndLabel)
  val roc1 = metrics.roc

  println("the ROC is " + roc1.collect())

  val auc1 = metrics.areaUnderROC

  println("the AUC is " + auc1)

  val metrics2 = new MulticlassMetrics(predictionAndLabel)
  val confmatrix = metrics2.confusionMatrix
```

```
println("************************************************OUTPUT***************************************
***")
   println("the accuracy is " + accuracy)
   println("the ROC is " + roc1.collect())
   println("the AUC is " + auc1)
   println("the confusionmatrix is  " + confmatrix)

println("************************************************OUTPUT***************************************
***")

 }

}
```

ANALYSIS PERFORMED:

1) Data Analysis was performed using Naïve Bayes Algorithm from SPARK ML using Scala in IBM CLOUD
2) .csv file was loaded as text so we have an RDD and not dataset type, as Naïve Bayes accepts only RDD.
3) Labeled point was created to label our target and features supporting it (Vector dense was used on Features)
4) All of the lines were mapped as double features and stored in parseddatapre
5) We split the data in 60% training and 40% test
6) The model is trained, and we test using our test data to get predictionandlabels, which gives us accuracy (67%)
7) We then run BinaryClassifier to get performance metrics such as AUC , ROC
8) MulticlassMetrics is used to get the Confusion Matrix

   AUC

```
hj997@F4Linux1:~/spark/spark-2.2.0-bin-custom-spark/bin

scala> rocpre
res39: org.apache.spark.rdd.RDD[(Double, Double)] = UnionRDD[195] at UnionRDD at BinaryClassificationMetrics.scala:90

scala> println(rocpre)
UnionRDD[195] at UnionRDD at BinaryClassificationMetrics.scala:90

scala> println(metrics.areaUnderROC)
0.6201485917054779

scala> val area = metrics.areaUnderROC
area: Double = 0.6201485917054779
```

CONFUSION MATRIX:

```
scala> metrics2.confusionMatrix
res48: org.apache.spark.mllib.linalg.Matrix =
136.0  31.0
62.0   46.0

scala> [hj997@F4Linux1 bin]$ ^C
[hj997@F4Linux1 bin]$ ^C
[hj997@F4Linux1 bin]$ ^C
[hj997@F4Linux1 bin]$
```

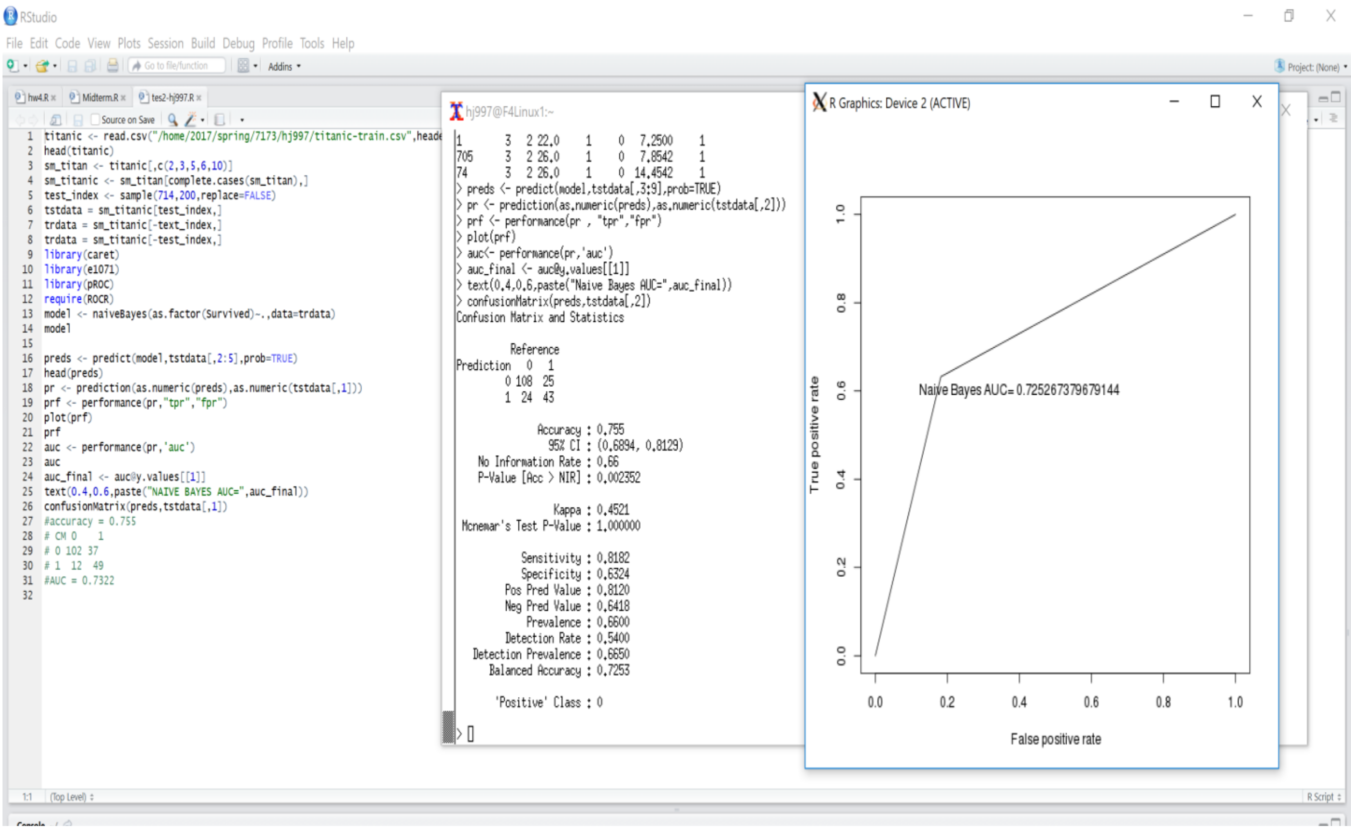
R CODE SNIPPET:

RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

hw4.R | Midterm.R | tes2-hj997.R

```r
library(caret)
library(e1071)
library(pROC)
require(ROCR)
titanic <- read.csv("/home/2017/spring/7173/hj997/titanic-train.csv",header=T)
head(titanic)
titanic$Ticket <- NULL
titanic$Name <- NULL
titanic[is.na(titanic)]<-0
titanic$Cabin <- as.numeric(titanic$Cabin)
titanic$Sex <- as.numeric(titanic$Sex)
titanic$Embarked <- as.numeric(titanic$Embarked)
write.table(titanic, file="/home/2017/spring/7173/hj997/spark/spark-2.2.0-bin-custom-spark/bin/Mydata.csv" ,col.names=FALSE , row.names = FALSE)
set.seed(11L)
test_index <- sample(714,200,replace=FALSE)
tstdata = titanic[test_index,]
trdata = titanic[-test_index,]
write.table(tstdata, file="/home/2017/spring/7173/hj997/spark/spark-2.2.0-bin-custom-spark/bin/test.csv" ,col.names=FALSE , row.names = FALSE)
write.table(trdata, file="/home/2017/spring/7173/hj997/spark/spark-2.2.0-bin-custom-spark/bin/train.csv" ,col.names=FALSE , row.names = FALSE)
model <- naiveBayes(as.factor(Survived)~.,data=trdata)
model
preds <- predict(model,tstdata[,3:9],prob=TRUE)
head(preds)
pr <- prediction(as.numeric(preds),as.numeric(tstdata[,2]))
prf <- performance(pr,"tpr","fpr")
plot(prf)
prf
auc <- performance(pr,'auc')
auc
auc_final <- auc@y.values[[1]]
text(0.4,0.6,paste("NAIVE BAYES AUC=",auc_final))
confusionMatrix(preds,tstdata[,2])
# CM 0    1
# 0 102 37
# 1  12  49
#AUC = 0.725
```



COMPARE RESULTS FROM R & SCALA:

R had slightly higher accuracy and lesser False Positives and False negatives than Scala Spark Apache. This is also an example of how an Algorithms performs slightly different based on tools beginning used. Over all for the ease of usage and features engineering as well model training R was easier. Scala had much more complex methods, but Scala & spark lets us gain complete understanding of internal schema as well is much faster and useful for parallel processing.

R – AUC 72%   SCALA AUC – 62%

REFERENCES:

https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html

https://spark.apache.org/docs/2.2.0/mllib-naive-bayes.html

https://spark.apache.org/docs/2.2.0/api/scala/index.html#org.apache.spark.package