In [1]:

```python
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
from pyspark.sql import SparkSession
from pyspark.sql.types import *
#import wget
from pyspark.ml.feature import Bucketizer,RegexTokenizer,StopWordsRemover,CountVectorizer,IDF, NGram, HashingTF, StringIndexer, Word2Vec
from pyspark.sql.functions import *
from pyspark.ml.classification import LogisticRegression, RandomForestClassificationModel, RandomForestClassifier, GBTClassificationModel, GBTClass
from pyspark.ml import Pipeline,PipelineModel
from pyspark.ml.evaluation import BinaryClassificationEvaluator


spark = SparkSession.builder.appName(
    "Opinion mining on Amazon Fashion product reviews"
).getOrCreate()

spark.sparkContext.setLogLevel("WARN")
```

Starting Spark application

ID	YARN Application ID	Kind	State	Spark UI	Driver log	User	Current session?
1	application_1671250343768_0002	pyspark	idle	Link	Link	None	✔

```
SparkSession available as 'spark'.
```

In [2]:

```python
from functools import reduce  # For Python 3.x
from pyspark.sql import DataFrame

def unionAll(*dfs):
    return reduce(DataFrame.unionAll, dfs)
```

In [3]:

```python
# File location and type
file_location = "s3://projectdatabia678/reviews_Clothing_Shoes_and_Jewelry_5.json.gz"
file_type = "gz"

# The applied options are for CSV files. For other file types, these will be ignored.
clsj = spark.read.json(file_location)
```

In [4]:

```python
#s3://projectdatabia678/reviews_Beauty_5.json.gz


# File location and type
file_location = "s3://projectdatabia678/reviews_Beauty_5.json.gz"
file_type = "gz"

# The applied options are for CSV files. For other file types, these will be ignored.
bt = spark.read.json(file_location)
```

In [5]:

```python
df  = unionAll(bt,clsj)
```

In [6]:

```python
df = df.drop("image")\
    .drop("verified")\
    .drop("vote")
df.columns
```

```
['asin', 'helpful', 'overall', 'reviewText', 'reviewTime', 'reviewerID', 'reviewerName', 'summary', 'unixReviewTime']
```

In [7]:

```python
df.printSchema()
```

```
root
 |-- asin: string (nullable = true)
 |-- helpful: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- overall: double (nullable = true)
 |-- reviewText: string (nullable = true)
 |-- reviewTime: string (nullable = true)
 |-- reviewerID: string (nullable = true)
 |-- reviewerName: string (nullable = true)
 |-- summary: string (nullable = true)
 |-- unixReviewTime: long (nullable = true)
```

In [8]:

```python
df = df.withColumn("text",concat(col("summary"), lit(" "),col("reviewText")))\
 .drop("helpful")\
 .drop("reviewerID")\
 .drop("reviewerName")\
 .drop("reviewTime")

# COMMAND ----------

#Print the Count of number of instances
df.count()
```

```
477179
```

In [10]:

```python
#COMMAND ----------

#Describe the Distribution of the ratings column 'overall'
df.describe("overall").show()

# COMMAND ----------

#Data Cleaning : Finding Null Values
for col in df.columns:
    print(col,":",df[df[col].isNull()].count())


# COMMAND ----------

#Filter out the  rows with neutral overall ratings
df1 = df.filter("overall !=3")

splits = [-float("inf"), 4.0, float("inf")]

#Bucketize data and create labels 0 if overall rating is in (1.0,2.0), #otherwise 1
bucketizer = Bucketizer(splits=splits,\
                        inputCol="overall", outputCol="label")

df2= bucketizer.transform(df1)

df2.groupBy("overall","label").count().show()

# COMMAND ----------

df2.show()

# COMMAND ----------

#take sample to create train and test dataset
fractions = {1.0 : .1, 0.0 : 1.0}
df3 = df2.stat.sampleBy("label", fractions, 36)
df3.groupBy("label").count().show()

# COMMAND ----------

#Split data as 80-20% Train and Test dataset
splitSeed = 5043
trainingData, testData = df3.randomSplit([0.80, 0.20], splitSeed)
```

```
+-------+------------------+
|summary|           overall|
+-------+------------------+
|  count|            477179|
|   mean|4.2223610007984425|
| stddev|1.1306299814841152|
|    min|               1.0|
|    max|               5.0|
+-------+------------------+

asin : 0
overall : 0
reviewText : 0
summary : 0
unixReviewTime : 0
text : 0
+-------+-----+------+
|overall|label| count|
+-------+-----+------+
|    5.0|  1.0|277771|
|    1.0|  0.0| 21718|
|    4.0|  1.0| 98098|
|    2.0|  0.0| 26919|
+-------+-----+------+

+----------+-------+--------------------+--------------------+--------------+--------------------+-----+
|      asin|overall|          reviewText|             summary|unixReviewTime|                text|label|
+----------+-------+--------------------+--------------------+--------------+--------------------+-----+
|7806397051|    1.0|Very oily and cre...|Don't waste your ...|    1391040000|Don't waste your ...|  0.0|
|7806397051|    4.0|The texture of th...|       great quality|    1378425600|great quality The...|  1.0|
|7806397051|    2.0|I really can't te...|Do not work on my...|    1386460800|Do not work on my...|  0.0|
|7806397051|    5.0|I was very happy ...|  Very nice palette!|    1365984000|Very nice palette...|  1.0|
|7806397051|    1.0|PLEASE DONT DO IT...|              smh!!!|    1376611200|smh!!! PLEASE DON...|  0.0|
|7806397051|    2.0|Chalky,Not Pigmen...|Chalky, Not Pigme...|    1378252800|Chalky, Not Pigme...|  0.0|
|9759091062|    2.0|Did nothing for m...|no Lightening, no...|    1405209600|no Lightening, no...|  0.0|
|9759091062|    1.0|Did nothing for m...|             Nothing|    1392681600|Nothing Did nothi...|  0.0|
|9759091062|    5.0|I bought this pro...|          This works|    1390435200|This works I boug...|  1.0|
|9759091062|    1.0|This gell did not...|        Does nothing|    1389398400|Does nothing This...|  0.0|
|9759091062|    5.0|i got this to get...|            it works|    1392681600|it works i got th...|  1.0|
|9759091062|    2.0|I used it for ana...|               burns|    1396742400|burns I used it f...|  0.0|
|9759091062|    5.0|I order this crea...|     Did work for me|    1379116800|Did work for me I...|  1.0|
|9759091062|    4.0|Good product. Use...|           excellent|    1382054400|excellent Good pr...|  1.0|
|9788072216|    5.0|I haven't been a ...|Love the smell of...|    1316390400|Love the smell of...|  1.0|
|9788072216|    5.0|We gave these as ...|               Happy|    1376092800|Happy We gave the...|  1.0|
|9788072216|    5.0|This is the first...|           Very good|    1322438400|Very good This is...|  1.0|
|9788072216|    5.0|So I got this abo...|    Lurrrrrrrrv.....|    1338076800|Lurrrrrrrrv..... ...|  1.0|
|9788072216|    5.0|This product has ...|         Great Scent|    1359763200|Great Scent This ...|  1.0|
|9790790961|    5.0|I'm very picky wh...|Spring Garden in ...|    1394496000|Spring Garden in ...|  1.0|
+----------+-------+--------------------+--------------------+--------------+--------------------+-----+
only showing top 20 rows

+-----+-----+
|label|count|
+-----+-----+
|  1.0|37633|
|  0.0|48637|
+-----+-----+
```

In [11]:

```python
#Tokenize the sentence based on the regex pattern
tokenizer = RegexTokenizer(inputCol="text",outputCol="reviewTokensUf",pattern="\\s+|[,.()\"]")

#Remove Stop Words that do not contribute in any way to our analysis
stopwords_remover = StopWordsRemover(stopWords=StopWordsRemover.loadDefaultStopWords("english"),inputCol="reviewTokensUf",outputCol="reviewTokens")

#converts word documents to vectors of token counts
cv = CountVectorizer(inputCol="reviewTokens",outputCol="cv",vocabSize=296337)

#IDF model
idf = IDF(inputCol="cv",outputCol="features")

#Logistic Boosted Classifier
lr = LogisticRegression(maxIter=100,regParam=0.02,elasticNetParam=0.3)
```

In [13]:

```python
#Create a pipeline by combining all the functions we defined above - tokenizer , stopwords_remover, cv, idf, gbtc
steps =  [tokenizer, stopwords_remover, cv, idf, lr]
lr_pipeline = Pipeline(stages=steps)

#fit the training dataset dataset into the pipeline
model = lr_pipeline.fit(trainingData)

#Obtain the predictions from the model
predictions = model.transform(testData)
```

In [12]:

```python
# COMMAND ----------

#Call the Binary Classification Evaluator function
evaluator = BinaryClassificationEvaluator()
areaUnderROC = evaluator.evaluate(predictions)
print('Test Area Under ROC for Linear Regression model with Count Vectorizer is ' , areaUnderROC)

# COMMAND ----------

#Building the vocabulary to explore the coefficients
vocabulary = model.stages[2].vocabulary
weights = model.stages[-1].coefficients.toArray()
weights = [float(weight) for weight in weights]

schema = StructType([StructField('word', StringType()),
                     StructField('weight', FloatType())
                     ])
cdf = spark.createDataFrame(zip(vocabulary, weights), schema)

# COMMAND ----------

cdf.orderBy(desc("weight")).show(10)

# COMMAND ----------

cdf.orderBy("weight").show(10)
```

```
Test Area Under ROC for Linear Regression model with Count Vectorizer is  0.9425358441407751
+-----------+----------+
|       word|    weight|
+-----------+----------+
|      great|0.50126475|
|       love| 0.4562465|
|    perfect|0.32605672|
|      loves|0.30220935|
|       nice|0.27957106|
|comfortable|0.24367884|
|compliments|0.23149578|
|  excellent|0.22868377|
|     great!|0.22442001|
|     highly|0.21256128|
+-----------+----------+
only showing top 10 rows

+-------------+-----------+
|         word|     weight|
+-------------+-----------+
| disappointed|-0.36692452|
|        waste| -0.2958275|
|     returned|-0.29348242|
|disappointing|-0.25194058|
|       return|-0.24936205|
|         poor|-0.24399708|
|    returning|-0.21882328|
|unfortunately|-0.21715495|
|        cheap|-0.20842884|
|     horrible|-0.19916554|
+-------------+-----------+
only showing top 10 rows
```

In [13]:

```python
# COMMAND ----------

lp = predictions.select("label", "prediction")
lp.printSchema()
lp.filter(~(F.col("label") == F.col("prediction"))).count()

# COMMAND ----------

#model evaluation
lp = predictions.select("label", "prediction")
counttotal = predictions.count()
correct = lp.filter(F.col("label")== F.col("prediction")).count()
wrong = lp.filter(~(F.col("label") == F.col("prediction"))).count()
ratioWrong = float(wrong) / float(counttotal)
ratioCorrect=correct/counttotal



trueneg =( lp.filter(F.col("label") == 0.0).filter(F.col("label") == F.col("prediction")).count()) /counttotal
truepos = (lp.filter(F.col("label") == 1.0).filter(F.col("label") == F.col("prediction")).count())/counttotal
falseneg = (lp.filter(F.col("label") == 0.0).filter(~(F.col("label") == F.col("prediction"))).count())/counttotal
falsepos = (lp.filter(F.col("label") == 1.0).filter(~(F.col("label") == F.col("prediction"))).count())/counttotal

precision = truepos / (truepos + falsepos)
recall  = truepos / (truepos + falseneg)
#fmeasure= 2  precision  recall / (precision + recall)
accuracy=(truepos + trueneg) / (truepos + trueneg + falsepos + falseneg)

# COMMAND ----------

print('counttotal   :', counttotal     )
print('correct      :', correct        )
print('wrong        :', wrong          )
print('ratioWrong   :', ratioWrong     )
print('ratioCorrect :', ratioCorrect   )
print('truen        :', trueneg          )
print('truep        :', truepos          )
print('falsen       :', falseneg         )
print('falsep       :', falsepos         )
print('precision    :', precision      )
print('recall       :', recall         )
print('accuracy     :', accuracy       )

# COMMAND ----------
```

```
root
 |-- label: double (nullable = true)
 |-- prediction: double (nullable = false)

counttotal   : 17364
correct      : 15179
wrong        : 2185
ratioWrong   : 0.12583506104584197
ratioCorrect : 0.874164938954158
truen        : 0.519638332181525
truep        : 0.35452660677263304
falsen       : 0.04756968440451509
falsep       : 0.07826537664132688
precision    : 0.8191616766467066
recall       : 0.881695789172157
accuracy     : 0.874164938954158
```

In [14]:

```python
#Create a pipeline by combining all the functions we defined above - tokenizer , stopwords_remover, cv, idf, gbtc

#Tokenize the sentence based on the regex pattern
tokenizer = RegexTokenizer(inputCol="text",outputCol="reviewTokensUf",pattern="\\s+|[,.()\"]")


bigram = NGram(inputCol = "reviewTokensUf", outputCol = "bigrams", n = 2)


tfs  = HashingTF(inputCol="bigrams", outputCol="h_features")

#IDF model
idf = IDF(inputCol="h_features",outputCol="features")

lr = LogisticRegression(maxIter=20)

steps =  [tokenizer, stopwords_remover, bigram, tfs, idf, lr]
bigrams_pipeline = Pipeline(stages=steps)
```

In [15]:

```python
# COMMAND ----------

model = bigrams_pipeline.fit(trainingData)
bi_predictions = model.transform(testData)
```

In [18]:

```python
# COMMAND ----------

evaluator = BinaryClassificationEvaluator()
areaUnderROC = evaluator.evaluate(bi_predictions)
print('Test Area Under ROC for Bigrams linear regression', areaUnderROC)

# COMMAND ----------

#model evaluation
lp = bi_predictions.select("label", "prediction")
counttotal = bi_predictions.count()
correct = lp.filter(F.col("label")== F.col("prediction")).count()
wrong = lp.filter(~(F.col("label") == F.col("prediction"))).count()
ratioWrong = float(wrong) / float(counttotal)
ratioCorrect=correct/counttotal



trueneg =( lp.filter(F.col("label") == 0.0).filter(F.col("label") == F.col("prediction")).count()) /counttotal
truepos = (lp.filter(F.col("label") == 1.0).filter(F.col("label") == F.col("prediction")).count())/counttotal
falseneg = (lp.filter(F.col("label") == 0.0).filter(~(F.col("label") == F.col("prediction"))).count())/counttotal
falsepos = (lp.filter(F.col("label") == 1.0).filter(~(F.col("label") == F.col("prediction"))).count())/counttotal

precision = truepos / (truepos + falsepos)
recall  = truepos / (truepos + falseneg)
#fmeasure= 2  precision  recall / (precision + recall)
accuracy=(truepos + trueneg) / (truepos + trueneg + falsepos + falseneg)

print('counttotal   :', counttotal     )
print('correct      :', correct        )
print('wrong        :', wrong          )
print('ratioWrong   :', ratioWrong     )
print('ratioCorrect :', ratioCorrect   )
print('truen        :', trueneg          )
print('truep        :', truepos          )
print('falsen       :', falseneg         )
print('falsep       :', falsepos         )
print('precision    :', precision      )
print('recall       :', recall         )
#print('fmeasure     :', fmeasure       )
print('accuracy     :', accuracy       )

# COMMAND ----------

bi_predictions.select(F.col("bigrams")).show(5)

# COMMAND ----------
```

```
Test Area Under ROC for Bigrams linear regression 0.9464655891452727
counttotal   : 17364
correct      : 15529
wrong        : 1835
ratioWrong   : 0.1056784151117254
ratioCorrect : 0.8943215848882746
truen        : 0.5260884588804423
truep        : 0.3682331260078323
falsen       : 0.04111955770559779
falsep       : 0.06455885740612761
precision    : 0.8508316699933467
recall       : 0.8995498030388295
accuracy     : 0.8943215848882746
+--------------------+
|             bigrams|
+--------------------+
|[no lightening, l...|
|[a favorite, favo...|
|[really covers, c...|
|[great little, li...|
|[must have, have ...|
+--------------------+
only showing top 5 rows
```

In [16]:

```python
#RandomForest Implementation
regexTokenizer = RegexTokenizer(inputCol="text", outputCol="words", pattern="\\W")

#Remove Stop Words that do not contribute in any way to our analysis
stopwords_remover_rf = \
StopWordsRemover(stopWords=StopWordsRemover.loadDefaultStopWords("english"),inputCol="words",outputCol="filtered")

# bag of words count
countVectors = CountVectorizer(inputCol="filtered", outputCol="features", vocabSize=10000, minDF=5)


label_stringIdx = StringIndexer(inputCol = "label", outputCol = "new_label")

rf = RandomForestClassifier(numTrees=3, maxDepth=2, labelCol="new_label", seed=42)

rf = RandomForestClassifier(labelCol="new_label", \
                            featuresCol="features", \
                            numTrees = 100, \
                            maxDepth = 4, \
                            maxBins = 32)

steps =  [regexTokenizer, stopwords_remover_rf, countVectors, label_stringIdx, rf]
randomforests_pipeline = Pipeline(stages=steps)

# COMMAND ----------

model = randomforests_pipeline.fit(trainingData)
rf_predictions = model.transform(testData)
```

In [20]:

```python
# COMMAND ----------

rf_predictions.show(5)

# COMMAND ----------

evaluator = BinaryClassificationEvaluator()
areaUnderROC = evaluator.evaluate(rf_predictions)
print('Test Area Under ROC for Random Forests Classification is ', areaUnderROC)

# COMMAND ----------

#model evaluation
lp = rf_predictions.select("label", "prediction")
counttotal = rf_predictions.count()
correct = lp.filter(F.col("label")== F.col("prediction")).count()
wrong = lp.filter(~(F.col("label") == F.col("prediction"))).count()
ratioWrong = float(wrong) / float(counttotal)
ratioCorrect=correct/counttotal



trueneg =( lp.filter(F.col("label") == 0.0).filter(F.col("label") == F.col("prediction")).count()) /counttotal
truepos = (lp.filter(F.col("label") == 1.0).filter(F.col("label") == F.col("prediction")).count())/counttotal
falseneg = (lp.filter(F.col("label") == 0.0).filter(~(F.col("label") == F.col("prediction"))).count())/counttotal
falsepos = (lp.filter(F.col("label") == 1.0).filter(~(F.col("label") == F.col("prediction"))).count())/counttotal

precision = truepos / (truepos + falsepos)
recall  = truepos / (truepos + falseneg)
#fmeasure= 2  precision  recall / (precision + recall)
accuracy=(truepos + trueneg) / (truepos + trueneg + falsepos + falseneg)

print('counttotal   :', counttotal     )
print('correct      :', correct        )
print('wrong        :', wrong          )
print('ratioWrong   :', ratioWrong     )
print('ratioCorrect :', ratioCorrect   )
print('truen        :', trueneg          )
print('truep        :', truepos          )
print('falsen       :', falseneg         )
print('falsep       :', falsepos         )
print('precision    :', precision      )
print('recall       :', recall         )
#print('fmeasure     :', fmeasure       )
print('accuracy     :', accuracy       )

# COMMAND ----------
```

```
+----------+-------+--------------------+--------------------+--------------+--------------------+-----+--------------------+--------------------+-
-------------------+---------+--------------------+--------------------+----------+
|      asin|overall|          reviewText|             summary|unixReviewTime|                text|label|               words|            filtered|
features|new_label|       rawPrediction|         probability|prediction|
+----------+-------+--------------------+--------------------+--------------+--------------------+-----+--------------------+--------------------+-
-------------------+---------+--------------------+--------------------+----------+
|9759091062|    2.0|Did nothing for m...|no Lightening, no...|    1405209600|no Lightening, no...|  0.0|[no, lightening, ...|[lightening, brig...|
(10000,[8,22,24,2...|      0.0|[58.6037740872404...|[0.58603774087240...|       0.0|
|B00004U9V2|    5.0|My son used some ...|          A favorite|    1399248000|A favorite My son...|  1.0|[a, favorite, my,...|[favorite, son, u...|
(10000,[5,24,26,2...|      1.0|[54.9123411787099...|[0.54912341178709...|       0.0|
|B000052WYD|    5.0|I have tried lots...|      Really covers.|    1365033600|Really covers. I ...|  1.0|[really, covers, ...|[really, covers, ...|
(10000,[7,36,55,6...|      1.0|[57.0794088913992...|[0.57079408891399...|       0.0|
|B000052WYD|    5.0|I think this prod...|Great little prod...|    1309910400|Great little prod...|  1.0|[great, little, p...|[great, little, p...|
(10000,[2,3,8,15,...|      1.0|[54.4750026032045...|[0.54475002603204...|       0.0|
|B000052WYD|    5.0|My must have. Gre...|           MUST HAVE|    1391817600|MUST HAVE My must...|  1.0|[must, have, my, ...|[must, must, grea...|
(10000,[3,30,81,8...|      1.0|[55.4514658163727...|[0.55451465816372...|       0.0|
+----------+-------+--------------------+--------------------+--------------+--------------------+-----+--------------------+--------------------+-
-------------------+---------+--------------------+--------------------+----------+
only showing top 5 rows

Test Area Under ROC for Random Forests Classification is  0.8956797407993092
counttotal   : 17364
correct      : 9903
wrong        : 7461
ratioWrong   : 0.4296821008984105
ratioCorrect : 0.5703178991015895
truen        : 0.56720801658604
truep        : 0.0031098825155494126
falsen       : 0.0
falsep       : 0.4296821008984105
precision    : 0.00718562874251497
recall       : 1.0
accuracy     : 0.5703178991015895
```

In [17]:

```python
nb = NaiveBayes(smoothing=1)
steps =  [regexTokenizer, stopwords_remover_rf, countVectors, label_stringIdx, nb]

nb_pipeline = Pipeline(stages=steps)

# COMMAND ----------

model = nb_pipeline.fit(trainingData)
nb_predictions = model.transform(testData)
```

In [21]:

```python
# COMMAND ----------

#Create a pipeline by combining all the functions we defined above - tokenizer , stopwords_remover, cv, idf, gbtc
#gbtc = GBTClassifier(maxIter=20)
#steps =  [tokenizer, stopwords_remover, cv, idf, gbtc]
#pipeline = Pipeline(stages=steps)

# COMMAND ----------

#model = pipeline.fit(trainingData)

# COMMAND ----------

#predictions = model.transform(testData)


# COMMAND ----------

#evaluator = BinaryClassificationEvaluator()
#areaUnderROC = evaluator.evaluate(predictions)
#print('Test Area Under ROC', areaUnderROC)

# COMMAND ----------

#NaiveBayes Implementation



# COMMAND ----------

evaluator = BinaryClassificationEvaluator()
areaUnderROC = evaluator.evaluate(nb_predictions)
print('Test Area Under ROC for Naive Bayes Classification is ', areaUnderROC)

# COMMAND ----------

#model evaluation
lp = nb_predictions.select("label", "prediction")
counttotal = nb_predictions.count()
correct = lp.filter(F.col("label")== F.col("prediction")).count()
wrong = lp.filter(~(F.col("label") == F.col("prediction"))).count()
ratioWrong = float(wrong) / float(counttotal)
ratioCorrect=correct/counttotal



trueneg =( lp.filter(F.col("label") == 0.0).filter(F.col("label") == F.col("prediction")).count()) /counttotal
truepos = (lp.filter(F.col("label") == 1.0).filter(F.col("label") == F.col("prediction")).count())/counttotal
falseneg = (lp.filter(F.col("label") == 0.0).filter(~(F.col("label") == F.col("prediction"))).count())/counttotal
falsepos = (lp.filter(F.col("label") == 1.0).filter(~(F.col("label") == F.col("prediction"))).count())/counttotal

precision = truepos / (truepos + falsepos)
recall  = truepos / (truepos + falseneg)
#fmeasure= 2  precision  recall / (precision + recall)
accuracy=(truepos + trueneg) / (truepos + trueneg + falsepos + falseneg)

print('counttotal   :', counttotal     )
print('correct      :', correct        )
print('wrong        :', wrong          )
print('ratioWrong   :', ratioWrong     )
print('ratioCorrect :', ratioCorrect   )
print('truen        :', trueneg          )
print('truep        :', truepos          )
print('falsen       :', falseneg         )
print('falsep       :', falsepos         )
print('precision    :', precision      )
print('recall       :', recall         )
#print('fmeasure     :', fmeasure       )
print('accuracy     :', accuracy       )
```

```
Test Area Under ROC for Naive Bayes Classification is  0.5466967847903211
counttotal   : 17364
correct      : 15447
wrong        : 1917
ratioWrong   : 0.11040082930200415
ratioCorrect : 0.8895991706979959
truen        : 0.506046993780235
truep        : 0.3835521769177609
falsen       : 0.06116102280580511
falsep       : 0.04923980649619903
precision    : 0.8862275449101796
recall       : 0.8624708624708625
accuracy     : 0.8895991706979959
```

In [25]:

```python
#Using XGBoost

#Create a pipeline by combining all the functions we defined above - tokenizer , stopwords_remover, cv, idf, gbtc

#Tokenize the sentence based on the regex pattern
tokenizer = RegexTokenizer(inputCol="text",outputCol="reviewTokensUf",pattern="\\s+|[,.()\"]")
word2Vec = Word2Vec(vectorSize=5, seed=42, inputCol="reviewTokensUf", outputCol="features")


steps =  [tokenizer,word2Vec]
xgboost_prepro = Pipeline(stages=steps)
xg_boost_model = xgboost_prepro.fit(trainingData)
train_df = xg_boost_model.transform(trainingData)
test_df = xg_boost_model.transform(testData)
```

In [18]:

```python
#Create a pipeline by combining all the functions we defined above - tokenizer , stopwords_remover, cv , gbtc

#Tokenize the sentence based on the regex pattern
tokenizer = RegexTokenizer(inputCol="text",outputCol="reviewTokensUf",pattern="\\s+|[,.()\"]")


#Remove Stop Words that do not contribute in any way to our analysis
stopwords_remover = StopWordsRemover(stopWords=StopWordsRemover.loadDefaultStopWords("english"),inputCol="reviewTokensUf",outputCol="reviewTokens")

#converts word documents to vectors of token counts
cv = CountVectorizer(inputCol="reviewTokens",outputCol="cv",vocabSize=296337)



steps =  [tokenizer, stopwords_remover, cv]
Pipeline_mo = Pipeline(stages=steps)
transformed_model = Pipeline_mo.fit(trainingData)
train_tr = transformed_model.transform(trainingData)
test_tr = transformed_model.transform(testData)

gbt = GBTClassifier(maxIter=5, maxDepth=2, labelCol="label", seed=42, featuresCol="cv")


gbt.fit(train_tr)
gbt.transform(test_tr)
```

```
Interrupted by user
```

In [ ]:

```python
#GBTClassifier : Gradiennt Boosted Trees

#Tokenize the sentence based on the regex pattern
tokenizer = RegexTokenizer(inputCol="text",outputCol="reviewTokensUf",pattern="\\s+|[,.()\"]")


#Remove Stop Words that do not contribute in any way to our analysis
stopwords_remover = StopWordsRemover(stopWords=StopWordsRemover.loadDefaultStopWords("english"),inputCol="reviewTokensUf",outputCol="reviewTokens")

#converts word documents to vectors of token counts
cv = CountVectorizer(inputCol="reviewTokens",outputCol="cv",vocabSize=296337)



steps =  [tokenizer, stopwords_remover, cv]
Pipeline_mo = Pipeline(stages=steps)
transformed_model = Pipeline_mo.fit(trainingData)
train_tr = transformed_model.transform(trainingData)
test_tr = transformed_model.transform(testData)

# training
gbtc = GBTClassifier(featuresCol="cv", labelCol="label", maxIter=20)
gbtc = gbtc.fit(train_tr)

# prediction
pred = gbtc.transform(test_tr)
pred.show(3)
```

In [ ]:

```python
evaluator = BinaryClassificationEvaluator()
areaUnderROC = evaluator.evaluate(pred)
print('Test Area Under ROC for Gradient Boost Classification is ', areaUnderROC)
```

In [ ]:

```python
from sklearn.metrics import confusion_matrix, classification_report

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator=MulticlassClassificationEvaluator(metricName="accuracy")
#e
acc = evaluator.evaluate(pred)

print("Prediction Accuracy: ", acc)

y_pred=pred.select("prediction").collect()
y_orig=pred.select("label").collect()

cm = confusion_matrix(y_orig, y_pred)
print("Confusion Matrix:")
print(cm)
```