Opinion Mining Project on Amazon Product Review Dataset

Import all the libraries and function from SparkSQL and SparkMLlib

In [0]:
```
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
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")
```

In [0]:
```
#Create UnionAll function to merge dataframs
from functools import reduce  # For Python 3.x
from pyspark.sql import DataFrame

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

In [0]:
```
# File location and type for Clothing , Shoes and Jewelry data
file_location = "/FileStore/tables/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 [0]:
```
# File location and type for beauty data
file_location = "/FileStore/tables/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 [0]:
```
#Execute union to merge bt and clsj dataframes
df  = unionAll(bt,clsj)
```

In [0]:
```
#Drop the columns that are not required part 1
df = df.drop("image")\
    .drop("verified")\
    .drop("vote")
df.columns
```

```
Out[49]: ['asin',
 'helpful',
 'overall',
 'reviewText',
 'reviewTime',
 'reviewerID',
 'reviewerName',
 'summary',
 'unixReviewTime']
```

In [0]:
```
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 [0]:
```
# Drop and columns and concatenate summary and reviewtext as Text column
df = df.withColumn("text",concat(F.col("summary"), lit(" "),F.col("reviewText")))\
 .drop("helpful")\
 .drop("reviewerID")\
 .drop("reviewerName")\
 .drop("reviewTime")
```

In [0]:
```
#Print the Count of number of instances
df.count()
```

```
Out[53]: 477179
```

In [0]:
```
#Describe the Distribution of the ratings column 'overall'
df.describe("overall").show()
```

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

In [0]:
```
#Data Cleaning : Finding Null Values in each column
for col in df.columns:
    print(col,":",df[df[col].isNull()].count())
```

```
asin : 0
overall : 0
reviewText : 0
summary : 0
unixReviewTime : 0
text : 0
```

In [0]:
```
#Filter out the  rows with neutral overall ratings
df1 = df.filter("overall !=3")

#Create a split such that from -inf to 4 will be placed in bucket 0 and 4 to inf will be placed in bucket 1
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()
```

```
+-------+-----+------+
|overall|label| count|
+-------+-----+------+
|    2.0|  0.0| 26919|
|    5.0|  1.0|277771|
|    1.0|  0.0| 21718|
|    4.0|  1.0| 98098|
+-------+-----+------+
```

In [0]:
```
#Diplay the dataframe after droping the columns and creating a new target variable/column 'label' based on overall ratings
df2.show()
```

```
+----------+-------+--------------------+--------------------+--------------+--------------------+-----+
|      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
```

In [0]:
```
#Sample withought replacement to resolve the imbalance in the target column
fractions = {1.0 : .1, 0.0 : 1.0}
df3 = df2.stat.sampleBy("label", fractions, 36)
df3.groupBy("label").count().show()
```

```
+-----+-----+
|label|count|
+-----+-----+
|  0.0|48637|
|  1.0|37519|
+-----+-----+
```

In [0]:
```
#Split data as 80-20% Train and Test dataset
splitSeed = 5043
trainingData, testData = df3.randomSplit([.80, 0.20], splitSeed)
```

NLP Text Preprocessing, Pipelines and Model Build

In [0]:
```
#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)

#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 [0]:
```
#Call the Binary Classification Evaluator function
evaluator = BinaryClassificationEvaluator()
areaUnderROC = evaluator.evaluate(lr_predictions)
print('Test Area Under ROC for Linear Regression model with Count Vectorizer is ' , areaUnderROC)
```

```
Test Area Under ROC for Linear Regression model with Count Vectorizer is  0.9398334608052311
```

Build a Vocabulary of all the relevant tokens and display the co-efficients

In [0]:
```
#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)
```

In [0]:
```
#Shows the top 10 positive sentiment tokens
cdf.orderBy(desc("weight")).show(10)
```

```
+-----------+----------+
|       word|    weight|
+-----------+----------+
|      great|  0.500755|
|       love| 0.4556638|
|    perfect|0.30859488|
|       nice|0.28766462|
|      loves|0.28668725|
|comfortable|0.24460426|
|  excellent|0.22886382|
|     great!|0.22694735|
|compliments|0.21397798|
|     highly|0.20946273|
+-----------+----------+
only showing top 10 rows
```

In [0]:
```
#Shows the top 10 negative sentiment tokens
cdf.orderBy("weight").show(10)
```

```
+--------------+-----------+
|          word|     weight|
+--------------+-----------+
|  disappointed| -0.4081317|
|      returned|-0.31633556|
|         waste|-0.29761943|
| disappointing|-0.26205418|
|          poor|-0.23398674|
|     returning|-0.22233732|
| unfortunately|-0.22159778|
|        return|-0.22129418|
|disappointment|-0.20847303|
|         cheap|-0.19569387|
+--------------+-----------+
only showing top 10 rows
```

In [0]:
```
#model evaluation and metrics for Logistic Regression  vanila version
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)

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       )
```

```
counttotal   : 17244
correct      : 14993
wrong        : 2251
ratioWrong   : 0.1305381581999536
ratioCorrect : 0.8694618418000464
truen        : 0.518905126420784
truep        : 0.35055671537926236
falsen       : 0.050916260728369286
falsep       : 0.07962189747158432
precision    : 0.8149096791588029
recall       : 0.8731763686263181
accuracy     : 0.8694618418000464
```

In [0]:
```
# Display the columns of predictions dataframe after transformation
predictions.printSchema()
display(predictions)
```

```
root
 |-- asin: string (nullable = true)
 |-- overall: double (nullable = true)
 |-- reviewText: string (nullable = true)
 |-- summary: string (nullable = true)
 |-- unixReviewTime: long (nullable = true)
 |-- text: string (nullable = true)
 |-- label: double (nullable = true)
 |-- reviewTokensUf: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- reviewTokens: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- cv: vector (nullable = true)
 |-- features: vector (nullable = true)
 |-- rawPrediction: vector (nullable = true)
 |-- probability: vector (nullable = true)
 |-- prediction: double (nullable = false)
```

asin	overall	reviewText	summary	unixReviewTime	
9759091062	1.0	Did nothing for my skin. Used as suggested and no signs of brightness. Wish it worked for me, but it didn't.	Nothing	1392681600	Nothing Did nothing for my skin. Used as suggested and no s Wish it worked f

In [0]:
```
#Model Logistic Regrssion with Bigrams and Hashing TF with IDF vectorization

#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)

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

In [0]:
```
evaluator = BinaryClassificationEvaluator()
areaUnderROC = evaluator.evaluate(bi_predictions)
print('Test Area Under ROC for Bigrams linear regression', areaUnderROC)
```

```
Test Area Under ROC for Bigrams linear regression 0.9467960427315577
```

In [0]:
```
#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       )
```

```
counttotal   : 17244
correct      : 15314
wrong        : 1930
ratioWrong   : 0.11192298770586871
ratioCorrect : 0.8880770122941313
truen        : 0.5028995592669914
truep        : 0.38517745302713985
falsen       : 0.06692182788216192
falsep       : 0.045001159823706796
precision    : 0.8953895928821785
recall       : 0.8519753719856337
accuracy     : 0.8880770122941313
```

In [0]:
```
# BiGRams created after the transformation
bi_predictions.select(F.col("bigrams")).show(5)
```

```
+--------------------+
|             bigrams|
+--------------------+
|[nothing did, did...|
|[industrial stren...|
|[dries out, out m...|
|[not impressed, i...|
|[love this, this ...|
+--------------------+
only showing top 5 rows
```

d RandomForest Implementation

In [0]:
```
#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, \
                            leafCol="leafId")

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)

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

In [0]:
```
rf_predictions.printSchema()
rf_predictions.show(5)
```

```
root
 |-- asin: string (nullable = true)
 |-- overall: double (nullable = true)
 |-- reviewText: string (nullable = true)
 |-- summary: string (nullable = true)
 |-- unixReviewTime: long (nullable = true)
 |-- text: string (nullable = true)
 |-- label: double (nullable = true)
 |-- reviewTokensUf: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- reviewTokens: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- bigrams: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- h_features: vector (nullable = true)
 |-- features: vector (nullable = true)
 |-- rawPrediction: vector (nullable = true)
 |-- probability: vector (nullable = true)
 |-- prediction: double (nullable = false)

+----------+-------+--------------------+--------------------+--------------+--------------------+-----+--------------------+--------------------+-
-------------------+--------------------+--------------------+--------------------+--------------------+----------+
|      asin|overall|          reviewText|             summary|unixReviewTime|                text|label|      reviewTokensUf|        reviewTokens|
bigrams|          h_features|            features|       rawPrediction|         probability|prediction|
+----------+-------+--------------------+--------------------+--------------+--------------------+-----+--------------------+--------------------+-
-------------------+--------------------+--------------------+--------------------+--------------------+----------+
|9759091062|    1.0|Did nothing for m...|             Nothing|    1392681600|Nothing Did nothi...|  0.0|[nothing, did, no...|[nothing, nothing...|
[nothing did, did...|(262144,[5175,105...|(262144,[5175,105...|[11.6976762579305...|[0.99999168695484...|       0.0|
|B00004TMFE|    2.0|I was so very dis...|Industrial streng...|    1401840000|Industrial streng...|  0.0|[industrial, stre...|[industrial, stre...|
[industrial stren...|(262144,[827,957,...|(262144,[827,957,...|[-33.729943770600...|[2.24528536462979...|       1.0|
|B00004TMFE|    2.0|Product dried out...|   Dries out my hair|    1341792000|Dries out my hair...|  0.0|[dries, out, my, ...|[dries, hair, pro...|
[dries out, out m...|(262144,[1047,103...|(262144,[1047,103...|[21.4810559052377...|[0.99999999953129...|       0.0|
|B00004TMFE|    2.0|We were pretty di...|       Not impressed|    1365724800|Not impressed We ...|  0.0|[not, impressed, ...|[impressed, prett...|
[not impressed, i...|(262144,[5663,692...|(262144,[5663,692...|[11.7933173945776...|[0.99999244518122...|       0.0|
|B00004U9UY|    5.0|This is one of my...|Love this scent s...|    1386115200|Love this scent s...|  1.0|[love, this, scen...|[love, scent, muc...|
[love this, this ...|(262144,[1996,427...|(262144,[1996,427...|[-22.243118661483...|[2.18744078296392...|       1.0|
+----------+-------+--------------------+--------------------+--------------+--------------------+-----+--------------------+--------------------+-
-------------------+--------------------+--------------------+--------------------+--------------------+----------+
only showing top 5 rows
```

In [0]:
```
evaluator = BinaryClassificationEvaluator()
areaUnderROC = evaluator.evaluate(rf_predictions)
print('Test Area Under ROC for Random Forests Classification is ', areaUnderROC)
```

```
Test Area Under ROC for Random Forests Classification is  0.8941074905567711
```

In [0]:
```
y_pred=rf_predictions.select("prediction").collect()
y_orig=rf_predictions.select("label").collect()

cf = classification_report(y_pred, y_orig)
print(cf)
```

```
              precision    recall  f1-score   support

         0.0       0.88      0.92      0.90      9448
         1.0       0.90      0.85      0.87      7796

    accuracy                           0.89     17244
   macro avg       0.89      0.88      0.89     17244
weighted avg       0.89      0.89      0.89     17244
```

In [0]:
```
#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       )
```

```
counttotal   : 17244
correct      : 10028
wrong        : 7216
ratioWrong   : 0.4184643934122013
ratioCorrect : 0.5815356065877987
truen        : 0.5695314312224542
truep        : 0.012004175365344467
falsen       : 0.00028995592669914174
falsep       : 0.4181744374855022
precision    : 0.027905095713130223
recall       : 0.9764150943396226
accuracy     : 0.5815356065877987
```

Naive Bayes

In [0]:
```
nb = NaiveBayes(smoothing=1)
steps =  [regexTokenizer, stopwords_remover_rf, countVectors, label_stringIdx, nb]

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

In [0]:
```
evaluator = BinaryClassificationEvaluator()
areaUnderROC = evaluator.evaluate(nb_predictions)
print('Test Area Under ROC for Naive Bayes Classification is ', areaUnderROC)
```

```
Test Area Under ROC for Naive Beyes Classification is  0.5399622753791408
```

In [0]:
```
#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       )
```

```
counttotal   : 17244
correct      : 15240
wrong        : 2004
ratioWrong   : 0.116214335421016
ratioCorrect : 0.883785664578984
truen        : 0.5045813036418464
truep        : 0.37920436093713755
falsen       : 0.0652400835073069
falsep       : 0.050974251913709114
precision    : 0.881504448638447
recall       : 0.8532098121085595
accuracy     : 0.883785664578984
```

In [0]:
```
#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)
```

In [0]:
```
train_tr.printSchema()
```

```
root
 |-- asin: string (nullable = true)
 |-- overall: double (nullable = true)
 |-- reviewText: string (nullable = true)
 |-- summary: string (nullable = true)
 |-- unixReviewTime: long (nullable = true)
 |-- text: string (nullable = true)
 |-- label: double (nullable = true)
 |-- reviewTokensUf: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- reviewTokens: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- cv: vector (nullable = true)
```

In [0]:
```
train_tr.show(5)
```

```
+----------+-------+--------------------+--------------------+--------------+--------------------+-----+--------------------+--------------------+-
-------------------+
|      asin|overall|          reviewText|             summary|unixReviewTime|                text|label|      reviewTokensUf|        reviewTokens|
cv|
+----------+-------+--------------------+--------------------+--------------+--------------------+-----+--------------------+--------------------+-
-------------------+
|7806397051|    1.0|PLEASE DONT DO IT...|              smh!!!|    1376611200|smh!!! PLEASE DON...|  0.0|[smh!!!, please, ...|[smh!!!, please, ...|
(87097,[0,4,5,9,4...|
|7806397051|    1.0|Very oily and cre...|Don't waste your ...|    1391040000|Don't waste your ...|  0.0|[don't, waste, yo...|[waste, money, oi...|
(87097,[43,55,67,...|
|7806397051|    2.0|Chalky,Not Pigmen...|Chalky, Not Pigme...|    1378252800|Chalky, Not Pigme...|  0.0|[chalky, not, pig...|[chalky, pigmente...|
(87097,[46,103,11...|
|7806397051|    2.0|I really can't te...|Do not work on my...|    1386460800|Do not work on my...|  0.0|[do, not, work, o...|[work, face, real...|
(87097,[7,21,25,4...|
|9759091062|    1.0|This gell did not...|        Does nothing|    1389398400|Does nothing This...|  0.0|[does, nothing, t...|[nothing, gell, n...|
(87097,[19,57,122...|
+----------+-------+--------------------+--------------------+--------------+--------------------+-----+--------------------+--------------------+-
-------------------+
only showing top 5 rows
```

In [0]:
```
# training
gbtc = GBTClassifier(featuresCol="cv", labelCol="label", maxIter=20)
gbtc = gbtc.fit(train_tr)

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

```
+----------+-------+--------------------+--------------------+--------------+--------------------+-----+--------------------+--------------------+-
-------------------+--------------------+--------------------+----------+
|      asin|overall|          reviewText|             summary|unixReviewTime|                text|label|      reviewTokensUf|        reviewTokens|
cv|       rawPrediction|         probability|prediction|
+----------+-------+--------------------+--------------------+--------------+--------------------+-----+--------------------+--------------------+-
-------------------+--------------------+--------------------+----------+
|9759091062|    1.0|Did nothing for m...|             Nothing|    1392681600|Nothing Did nothi...|  0.0|[nothing, did, no...|[nothing, nothing...|
(87097,[8,22,122,...|[0.59522853597419...|[0.76682280121619...|       0.0|
|B00004TMFE|    2.0|I was so very dis...|Industrial streng...|    1401840000|Industrial streng...|  0.0|[industrial, stre...|[industrial, stre...|
(87097,[0,1,2,4,6...|[0.78854553024981...|[0.82879214659798...|       0.0|
|B00004TMFE|    2.0|Product dried out...|   Dries out my hair|    1341792000|Dries out my hair...|  0.0|[dries, out, my, ...|[dries, hair, pro...|
(87097,[1,2,3,17,...|[0.58726461104543...|[0.76396271794041...|       0.0|
+----------+-------+--------------------+--------------------+--------------+--------------------+-----+--------------------+--------------------+-
-------------------+--------------------+--------------------+----------+
only showing top 3 rows
```

In [0]:
```
print(gbtc)
```

```
GBTClassificationModel: uid = GBTClassifier_b7bb4030176d, numTrees=20, numClasses=2, numFeatures=87097
```

In [0]:
```
evaluator = BinaryClassificationEvaluator()
areaUnderROC = evaluator.evaluate(pred)
print('Test Area Under ROC for Gradient Boost Classification is ', areaUnderROC)
```

```
Test Area Under ROC for Naive Bayes Classification is  0.8692959846983227
```

In [0]:
```
pred.show(5)
```

```
+----------+-------+--------------------+--------------------+--------------+--------------------+-----+--------------------+--------------------+-
-------------------+--------------------+--------------------+----------+
|      asin|overall|          reviewText|             summary|unixReviewTime|                text|label|      reviewTokensUf|        reviewTokens|
cv|       rawPrediction|         probability|prediction|
+----------+-------+--------------------+--------------------+--------------+--------------------+-----+--------------------+--------------------+-
-------------------+--------------------+--------------------+----------+
|9759091062|    1.0|Did nothing for m...|             Nothing|    1392681600|Nothing Did nothi...|  0.0|[nothing, did, no...|[nothing, nothing...|
(87097,[8,22,122,...|[0.59522853597419...|[0.76682280121619...|       0.0|
|B00004TMFE|    2.0|I was so very dis...|Industrial streng...|    1401840000|Industrial streng...|  0.0|[industrial, stre...|[industrial, stre...|
(87097,[0,1,2,4,6...|[0.78854553024981...|[0.82879214659798...|       0.0|
|B00004TMFE|    2.0|Product dried out...|   Dries out my hair|    1341792000|Dries out my hair...|  0.0|[dries, out, my, ...|[dries, hair, pro...|
(87097,[1,2,3,17,...|[0.58726461104543...|[0.76396271794041...|       0.0|
|B00004TMFE|    2.0|We were pretty di...|       Not impressed|    1365724800|Not impressed We ...|  0.0|[not, impressed, ...|[impressed, prett...|
(87097,[1,2,5,15,...|[0.96996381320242...|[0.87434419227186...|       0.0|
|B00004U9UY|    5.0|This is one of my...|Love this scent s...|    1386115200|Love this scent s...|  1.0|[love, this, scen...|[love, scent, muc...|
(87097,[3,4,7,10,...|[-0.8614088581098...|[0.15150857992698...|       1.0|
+----------+-------+--------------------+--------------------+--------------+--------------------+-----+--------------------+--------------------+-
-------------------+--------------------+--------------------+----------+
only showing top 5 rows
```

In [0]:
```
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)
```

```
Prediction Accuracy:  0.7811992577128276
Confusion Matrix:
[[8332 1494]
 [2279 5139]]
```

In [0]:
```
from sklearn.metrics import confusion_matrix, classification_report

cf = classification_report(y_orig, y_pred)
print(cf)
```

```
              precision    recall  f1-score   support

         0.0       0.79      0.85      0.82      9826
         1.0       0.77      0.69      0.73      7418

    accuracy                           0.78     17244
   macro avg       0.78      0.77      0.77     17244
weighted avg       0.78      0.78      0.78     17244
```

In [0]: