Case Study: Spam Detection
Domain: Telecom

There is a telecom operator forum in which cell phone users make public claims about SMS spam messages. The dataset contains a public set of SMS labeled messages that have been collected for mobile phone spam research. The sample collection is composed by 5,574 English, real and non-encoded messages, tagged according to being legitimate (ham) or spam.

Below is the sample dataset:

- Ham : What you doing? how are you?
- Spam : Sunshine Quiz! Win a super Sony DVD recorder if you can name the capital of Australia? Text MQUIZ to 82277

Tasks:

As a big data consultant , you are provided the sample dataset to generate the word cloud using Spark MLlib

You have to load this dataset in the HDFS and perform:

1. **Extract words from the SMS message**

   smsDF = spark.read.csv("/user/edureka_524533/Datasets/SMSSpamCollection",inferSchema=True,header=False,sep='\t').withColumnRenamed("_c0","message_type").withColumnRenamed("_c1","message_content")

   smsDF.printSchema()

   smsDF = smsDF.withColumnRenamed("_c0","message_type").withColumnRenamed("_c1","message_content")

```
In [21]: smsDF.printSchema()
```

```
root
 |-- message_type: string (nullable = true)
 |-- message_content: string (nullable = true)
```

```
In [22]: smsDF.show(5)
```

```
+------------+--------------------+
|message_type|     message_content|
+------------+--------------------+
|         ham|Go until jurong p...|
|         ham|Ok lar... Joking ...|
|        spam|Free entry in 2 a...|
|         ham|U dun say so earl...|
|         ham|Nah I don't think...|
+------------+--------------------+
only showing top 5 rows
```

2. **Removed stop words.**

   Convert message_type into equivalent Interger values which will help in later analysis

   def numeric_messType(messType):

```
    if messType == 'ham':
        return 1
    else:
        return 0
```

Register UDF

```
udf_convertToNumeric = udf(numeric_messType,IntegerType())
```

#Replace all 'ham' with 1 and 'spam' with 0, so we have numeric fields instead of string

```
smsDFStat =
smsDF.select('*',udf_convertToNumeric(smsDF['message_type']).alias('message_status'))
```

smsDFStat.show(5)

[10]:
```
smsDFStat.show(5)
```

```
+------------+--------------------+--------------+
|message_type|     message_content|message_status|
+------------+--------------------+--------------+
|         ham|Go until jurong p...|             1|
|         ham|Ok lar... Joking ...|             1|
|        spam|Free entry in 2 a...|             0|
|         ham|U dun say so earl...|             1|
|         ham|Nah I don't think...|             1|
+------------+--------------------+--------------+
only showing top 5 rows
```

UDF to remove Punctuations:

```
def remove_punctuations(message):
    messageEdit = [char for char in message if char not in string.punctuation]
    message = ''.join(messageEdit)
    return message
```

Register the UDF

```
udf_puncEdit = udf(remove_punctuations)
```

Execute to create a column without the punctuations:

```
smsDF1 = (smsDFStat.select('*',
udf_puncEdit(smsDFStat['message_content']).alias('message_punc')))
```

```
+------------+--------------------+--------------------+
|message_type|     message_content|        message_punc|
+------------+--------------------+--------------------+
|         ham|Go until jurong p...|Go until jurong p...|
|         ham|Ok lar... Joking ...|Ok lar Joking wif...|
|        spam|Free entry in 2 a...|Free entry in 2 a...|
|         ham|U dun say so earl...|U dun say so earl...|
|         ham|Nah I don't think...|Nah I dont think ...|
+------------+--------------------+--------------------+
only showing top 5 rows
```

UDF to remove StopWords:

```
from pyspark.ml.feature import StopWordsRemover
stopWords = StopWordsRemover.loadDefaultStopWords('english')
def remove_stopWords(message):
    wordList = message.split(' ')
    messageEdit = [word for word in wordList if word not in stopWords]
    message = ' '.join(messageEdit)
    return message
```

Register the UDF

```
udf_stopWEdit = udf(remove_stopWords)
```

Execute to create a column without the StopWords

```
smsDF2 = (smsDF1.select('*',
udf_stopWEdit(smsDF1['message_punc']).alias('message_stopW')))
```

```
In [59]: smsDF2.show(5)
```

```
+------------+--------------------+--------------------+--------------------+
|message_type|     message_content|        message_punc|       message_stopW|
+------------+--------------------+--------------------+--------------------+
|         ham|Go until jurong p...|Go until jurong p...|Go jurong point c...|
|         ham|Ok lar... Joking ...|Ok lar Joking wif...|Ok lar Joking wif...|
|        spam|Free entry in 2 a...|Free entry in 2 a...|Free entry 2 wkly...|
|         ham|U dun say so earl...|U dun say so earl...|U dun say early h...|
|         ham|Nah I don't think...|Nah I dont think ...|Nah I dont think ...|
+------------+--------------------+--------------------+--------------------+
only showing top 5 rows
```

3. **Modify the stop words to include your custom words such as '-'**

Already taken care by the string.punctuation

Modified the dataset to include only the columns to be used in further analysis

```
df = smsDF2.select('message_status','message_stopW')
df.printSchema()
```

```
df.printSchema()
```

```
root
 |-- message_status: integer (nullable = true)
 |-- message_stopW: string (nullable = true)
```

Convert message_stopW into a column of ArrayType which is required for CountVectorizer to work
from pyspark.sql.functions import col, split
df = df.withColumn("Message_Array", split(col("message_stopW")," "))

Again removed the unnecessary columns
df1 = df.select('message_status','Message_Array')

In [30]:
```
df1.printSchema()
```

```
root
 |-- message_status: integer (nullable = true)
 |-- Message_Array: array (nullable = true)
 |    |-- element: string (containsNull = true)
```

In [31]:
```
df1.show(5)
```

```
+--------------+--------------------+
|message_status|       Message_Array|
+--------------+--------------------+
|             1|[Go, jurong, poin...|
|             1|[Ok, lar, Joking,...|
|             0|[Free, entry, 2, ...|
|             1|[U, dun, say, ear...|
|             1|[Nah, I, dont, th...|
+--------------+--------------------+
only showing top 5 rows
```

4. **Create the features from SMS message using CountVectorizer**
   from pyspark.ml.feature import CountVectorizer
   cv = CountVectorizer(inputCol="Message_Array", outputCol="cv", vocabSize=4, minDF=1.0)

5. **Split the data into train and test - decide on a strategy**
   trainData,testData = df1.randomSplit([0.7,0.3])

6. Use logistic regression and check the accuracy
   from pyspark.ml.feature import IDF,StringIndexer
   from pyspark.ml.classification import LogisticRegression

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
idf = IDF(inputCol='cv', outputCol="features", minDocFreq=5)
label_stringIdx = StringIndexer(inputCol = "message_status", outputCol = "label")
lr = LogisticRegression(maxIter=100)
```

Create the Pipeline

```
pipeline = Pipeline(stages=[cv, idf, label_stringIdx, lr])
```

Create Model on Training Data

```
pipelineFit = pipeline.fit(trainData)
```

Make Predictions on Test Data

```
predictions = pipelineFit.transform(testData)
```

```
In [45]: predictions.show(5)
```

```
+--------------+--------------------+--------------------+--------------------+-----+--------------------+----------
---------+----------+
|message_status|       Message_Array|                  cv|            features|label|       rawPrediction|         pr
obability|prediction|
+--------------+--------------------+--------------------+--------------------+-----+--------------------+----------
---------+----------+
|             0|[, FREE, POLYPHON...|       (4,[0],[1.0])|(4,[0],[1.6503338...|  1.0|[1.67997187410784...|[0.84290080
674692...|       0.0|
|             0|[0ANETWORKS, allo...|           (4,[],[])|           (4,[],[])|  1.0|[1.69852572011228...|[0.84534208
738786...|       0.0|
|             0|[1000s, flirting,...|       (4,[0],[2.0])|(4,[0],[3.3006677...|  1.0|[1.66141802810340...|[0.84042826
462966...|       0.0|
|             0|[1000s, girls, ma...|(4,[0,2,3],[1.0,2...|(4,[0,2,3],[1.650...|  1.0|[-0.5158515842156...|[0.37382278
257869...|       1.0|
|             0|     [22, 146tf150p]|           (4,[],[])|           (4,[],[])|  1.0|[1.69852572011228...|[0.84534208
738786...|       0.0|
+--------------+--------------------+--------------------+--------------------+-----+--------------------+----------
---------+----------+
only showing top 5 rows
```

```
In [46]: predictions.select('message_status','prediction').show(100)
```

```
+--------------+----------+
|message_status|prediction|
+--------------+----------+
|             0|       0.0|
|             0|       0.0|
|             0|       0.0|
|             0|       1.0|
|             0|       0.0|
|             0|       0.0|
|             0|       0.0|
|             0|       0.0|
|             0|       0.0|
|             0|       0.0|
|             0|       0.0|
|             0|       0.0|
|             0|       1.0|
|             0|       0.0|
|             0|       0.0|
|             0|       0.0|
|             0|       0.0|
|             0|       0.0|
```

Check Accuracy

```
accuracy = predictions.filter(predictions.label == predictions.prediction).count() /
float(testData.count())
```

```
In [48]: accuracy
Out[48]: 0.8792892156862745
```

evaluate1 =
BinaryClassificationEvaluator(rawPredictionCol='prediction',labelCol='message_status')
roc_auc = evaluate1.evaluate(predictions)

```
In [51]: roc_auc
Out[51]: 0.48149712652669563
```

7. Try to use a Random Forest classifier and see if it increases the accuracy.
from pyspark.ml.classification import RandomForestClassifier

rf = RandomForestClassifier(labelCol="message_status", featuresCol="features")
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
Define the Pipeline
pipeline = Pipeline(stages=[cv, idf, label_stringIdx, rf])
# Train a RandomForest model.
pipelineFit = pipeline.fit(trainData)
Use Test data for predictions
predictions = pipelineFit.transform(testData)

```
In [59]: predictions.show(5)
+--------------+--------------------+--------------------+--------------------+-----+--------------------+----------
---------+----------+
|message_status|       Message_Array|                  cv|            features|label|       rawPrediction|         pr
obability|prediction|
+--------------+--------------------+--------------------+--------------------+-----+--------------------+----------
---------+----------+
|             0|[, FREE, POLYPHON...|       (4,[0],[1.0])|(4,[0],[1.6503338...|  1.0|[4.63444364712323...|[0.23172218
235616...|       1.0|
|             0|[0ANETWORKS, allo...|           (4,[],[])|           (4,[],[])|  1.0|[2.76821647667088...|[0.13841082
383354...|       1.0|
|             0|[1000s, flirting,...|       (4,[0],[2.0])|(4,[0],[3.3006677...|  1.0|[4.10887937323700...|[0.20544396
866185...|       1.0|
|             0|[1000s, girls, ma...|(4,[0,2,3],[1.0,2...|(4,[0,2,3],[1.650...|  1.0|[3.30555555555555...|[0.16527777
777777...|       1.0|
|             0|     [22, 146tf150p]|           (4,[],[])|           (4,[],[])|  1.0|[2.76821647667088...|[0.13841082
383354...|       1.0|
+--------------+--------------------+--------------------+--------------------+-----+--------------------+----------
---------+----------+
only showing top 5 rows
```

Check Accuracy
evaluator = MulticlassClassificationEvaluator(labelCol="message_status",
predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)

```
In [62]: accuracy
Out[62]: 0.8817401960784313
```

Check Test Error

```
In [63]: accuracy = evaluator.evaluate(predictions)
         print("Test Error = %g" % (1.0 - accuracy))

         Test Error = 0.11826
```

Check weighted Percision

```
evaluatorwp = MulticlassClassificationEvaluator(labelCol="message_status",
predictionCol="prediction", metricName="weightedPrecision")
wp = evaluatorwp.evaluate(predictions)
print("weightedPrecision = %g" % wp)
```

```
In [64]: evaluatorwp = MulticlassClassificationEvaluat
         wp = evaluatorwp.evaluate(predictions)
         print("weightedPrecision = %g" % wp)

         weightedPrecision = 0.866659
```

8. Introduce bi-gram and tri-gram and note the change in accuracy.

```
from pyspark.ml.feature import NGram, CountVectorizer, VectorAssembler
def build_ngrams(inputCol="Message_Array", n=3):

    ngrams = [
      NGram(n=i, inputCol="Message_Array", outputCol="{0}_grams".format(i))
      for i in range(1, n + 1)
    ]

    vectorizers = [
      CountVectorizer(inputCol="{0}_grams".format(i),
        outputCol="{0}_counts".format(i))
      for i in range(1, n + 1)
    ]

    assembler = [VectorAssembler(
      inputCols=["{0}_counts".format(i) for i in range(1, n + 1)],
      outputCol="features"
    )]

    return Pipeline(stages=ngrams + vectorizers + assembler)
```

Create Pipeline

Pipeline = build_ngrams()

Train the Model and make predictions

result = build_ngrams().fit(trainData).transform(testData)

```
In [57]: result.show(5)
```

```
+--------------+--------------------+--------------------+--------------------+--------------------+----------------
---+--------------------+--------------------+--------------------+
|message_status|       Message_Array|             1_grams|             2_grams|             3_grams|             1_cou
nts|            2_counts|            3_counts|            features|
+--------------+--------------------+--------------------+--------------------+--------------------+----------------
---+--------------------+--------------------+--------------------+
|             0|[08714712388, 10a...|[08714712388, 10a...|[08714712388 10am...|[08714712388 10am...|(9441,[649,826,8
6...|(28685,[1341,1507...|(29594,[28827],[1...|(67720,[649,826,8...|
|             0|[123, Congratulat...|[123, Congratulat...|[123 Congratulati...|[123 Congratulati...|(9441,[0,2,4,84,
1...|(28685,[67,2372,2...|(29594,[1098,2200...|(67720,[0,2,4,84,...|
|             0|[18, days, Euro20...|[18, days, Euro20...|[18 days, days Eu...|[18 days Euro2004...|(9441,[5,33,104,
1...|(28685,[1944,2246...|(29594,[1969,2873...|(67720,[5,33,104,...|
|             0|[1st, wk, FREE, G...|[1st, wk, FREE, G...|[1st wk, wk FREE,...|[1st wk FREE, wk ...|(9441,[2,3,66,9
0,...|(28685,[310,3212,...|       (29594,[],[])|(67720,[2,3,66,90...|
|             0|     [22, 146tf150p]|     [22, 146tf150p]|      [22 146tf150p]|                  []| (9441,[8081],[1.
0])|       (28685,[],[])|       (29594,[],[])|(67720,[8081],[1.0])|
+--------------+--------------------+--------------------+--------------------+--------------------+----------------
---+--------------------+--------------------+--------------------+
only showing top 5 rows
```

9. **Decide on a strategy and generate a data pipeline.**
   Covered in previous points