Sentiment Analysis in Pyspark

In [1]:
```
#Import modules
from pyspark.sql.types import *
from pyspark.sql.functions import *
import pandas as pd
import numpy as np
from pyspark.ml.feature import Tokenizer, HashingTF, StopWordsRemover

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Spark context
sc
```

Out[2]:

SparkContext

Spark UI (http://10.0.2.15:4041)

Version

v3.2.3

Master

local[*]

AppName

PySparkShell

In [3]:
```
#sc master-running locally
sc.master
```

Out[3]:
```
'local[*]'
```

In [4]:
```
#Define Schema
customSchema = StructType([
    StructField("target", StringType()),
    StructField("id", StringType()),
    StructField("date", StringType()),
    StructField("flag", StringType()),
    StructField("user", StringType()),
    StructField("text", StringType())
])
```

In [5]:
```
#Read the input file from Hadoop Distributed File System
#http://help.sentiment140.com/for-students
df = spark.read.load('hdfs://localhost:9000/user1/training.1600000.processed.noemoticon.csv',
                     format="csv",
                     sep=',',
                     schema=customSchema).toDF('target', 'id', 'date','flag','user','text')
```

In [6]:
```
#Display the first three records of our dataframe
df.show(3);
```

```
[Stage 0:>                                                          (0 + 1) / 1]

+------+----------+--------------------+--------+---------------+--------------------+
|target|        id|                date|    flag|           user|                text|
+------+----------+--------------------+--------+---------------+--------------------+
|     0|1467810369|Mon Apr 06 22:19:...|NO_QUERY|_TheSpecialOne_|@switchfoot http:...| (http:...|)
|     0|1467810672|Mon Apr 06 22:19:...|NO_QUERY|  scotthamilton|is upset that he ...|
|     0|1467810917|Mon Apr 06 22:19:...|NO_QUERY|       mattycus|@Kenichan I dived...|
+------+----------+--------------------+--------+---------------+--------------------+
only showing top 3 rows
```

The table has 6 columns: **target**: contains the label of the sentiment. **id**: unique number for the tweet. **date**: tweet date **flag**: will not use, **user** twitter user's, **text** the actual tweet

In [7]:
```
#Display the number of rows in the dataframe
df.count()
```

Out[7]: 1600000

Exploratory data analysis (EDA)

In [8]:
```
#Display the dataframe schema(in tree format)
df.printSchema()
```

```
root
 |-- target: string (nullable = true)
 |-- id: string (nullable = true)
 |-- date: string (nullable = true)
 |-- flag: string (nullable = true)
 |-- user: string (nullable = true)
 |-- text: string (nullable = true)
```

In [9]:
```
#Rename Columns
# rename the text --> tweet
df = df.withColumnRenamed("text", "tweet")

#rename the target --> Sentiment
df = df.withColumnRenamed("target", "Sentiment")
```

In [10]:
```
df.show(2)
```

```
+---------+----------+--------------------+--------+--------------+--------------------+
|Sentiment|        id|                date|    flag|          user|               tweet|
+---------+----------+--------------------+--------+--------------+--------------------+
|        0|1467810369|Mon Apr 06 22:19:...|NO_QUERY|_TheSpecialOne_|@switchfoot http:...| (http:...|)
|        0|1467810672|Mon Apr 06 22:19:...|NO_QUERY|  scotthamilton|is upset that he ...|
+---------+----------+--------------------+--------+--------------+--------------------+
only showing top 2 rows
```

In [11]:
```
#Display the count of distinct values in the target column
df.select('Sentiment').distinct().count()
```

Out[11]: 2

In [12]:
```
#Display the unique distinct values in the target column
df.select('Sentiment').distinct().show()
```

```
[Stage 11:==============================>                          (1 + 1) / 2]

+---------+
|Sentiment|
+---------+
|        0|
|        4|
+---------+
```

The 'Sentiment' column has two values of the sentiment (4) for positive tweet and (0) for negative tweet.

Null Values

In [13]:
```
col_null_cnt_df =  df.select([
    count(when(col(c).isNull(),c)).alias(c) for c in df.columns])

```

In [14]:
```
1 col_null_cnt_df.show()
```

```
[Stage 14:>                                                 (0 + 2) / 2]

+---------+---+----+----+----+-----+
|Sentiment| id|date|flag|user|tweet|
+---------+---+----+----+----+-----+
|        0|  0|   0|   0|   0|    0|
+---------+---+----+----+----+-----+
```

Indexing Dataframe

We cannot access a Spark dataframe by [row,column] as we can a pandas dataframe since Spark dataframes are dispersed across clusters. A different approach to accomplishing that is by adding a new column with "incremental ID". Using the ".filter()" function on the "incremental ID" column, we can then retrieve data by row.

In [15]:
```
1 df = df.withColumn("index", monotonically_increasing_id())
2 df.show(2)
```

```
+---------+----------+--------------------+--------+---------------+--------------------+-----+
|Sentiment|        id|                date|    flag|           user|               tweet|index|
+---------+----------+--------------------+--------+---------------+--------------------+-----+
|        0|1467810369|Mon Apr 06 22:19:...|NO_QUERY|_TheSpecialOne_|@switchfoot http:...| (http:...|)    0|
|        0|1467810672|Mon Apr 06 22:19:...|NO_QUERY|  scotthamilton|is upset that he ...|    1|
+---------+----------+--------------------+--------+---------------+--------------------+-----+
only showing top 2 rows
```

Create a new dataframe

In [16]:
```
1 df_new =  df.select(df["index"],
2                      df["tweet"],df["Sentiment"].cast("Int"))
```

In [17]:
```
df_new.show(2)
```

```
+-----+--------------------+---------+
|index|               tweet|Sentiment|
+-----+--------------------+---------+
|    0|@switchfoot http:...| (http:...|)        0|
|    1|is upset that he ...|        0|
+-----+--------------------+---------+
only showing top 2 rows
```

In [18]:
```
#print the schema
df_new.printSchema()
```

```
root
 |-- index: long (nullable = false)
 |-- tweet: string (nullable = true)
 |-- Sentiment: integer (nullable = true)
```

In [ ]:
```

```

Split the data into train & test

In [19]:
```
# Split the data, 80% for training, 20% for testing
splitdata = df_new.randomSplit([0.8, 0.2])
data_train = splitdata[0]          #index 0 = data training
data_test  = splitdata[1]          #index 1 = data testing
train_rows = data_train.count()
test_rows = data_test.count()
print ("Training data rows:", train_rows, "; Testing data rows:", test_rows)

```

```
[Stage 22:>                                                         (0 + 2) / 2]

Training data rows: 1280082 ; Testing data rows: 319918
```

In [ ]:
```
1
```

Training Data Preprocessing

Tokenizer

In [20]:
```
1 #Separate "SentimentText" into individual words using tokenizer
2
3 from pyspark.ml.feature import Tokenizer
4 # use PySparks build in tokenizer to tokenize tweets
5 tokenizer = Tokenizer(inputCol  = "tweet",
6                        outputCol = "SentimentWords")
7
8 tokenized_dfTrain = tokenizer.transform(data_train)
9 tokenized_dfTrain.show(3)
```

```
[Stage 25:>                                                         (0 + 1) / 1]

+-----+--------------------+---------+--------------------+
|index|               tweet|Sentiment|      SentimentWords|
+-----+--------------------+---------+--------------------+
|    1|is upset that he ...|        0|[is, upset, that,...|
|    2|@Kenichan I dived...|        0|[@kenichan, i, di...|
|    3|my whole body fee...|        0|[my, whole, body,...|
+-----+--------------------+---------+--------------------+
only showing top 3 rows
```

Remove Stop Words

In [21]:
```
swr = StopWordsRemover(inputCol=tokenizer.getOutputCol(),
                       outputCol="MeaningfulWords")
SwRemovedTrain = swr.transform(tokenized_dfTrain)
SwRemovedTrain.show(3)
```

```
[Stage 26:>                                                         (0 + 1) / 1]

+-----+--------------------+---------+--------------------+--------------------+
|index|               tweet|Sentiment|      SentimentWords|     MeaningfulWords|
+-----+--------------------+---------+--------------------+--------------------+
|    1|is upset that he ...|        0|[is, upset, that,...|[upset, update, f...|
|    2|@Kenichan I dived...|        0|[@kenichan, i, di...|[@kenichan, dived...|
|    3|my whole body fee...|        0|[my, whole, body,...|[whole, body, fee...|
+-----+--------------------+---------+--------------------+--------------------+
only showing top 3 rows
```

Converting words feature into numerical feature

In Spark 2.2.1,it is implemented in HashingTF funtion using Austin Appleby's MurmurHash 3 algorithm

In [22]:
```
hashTF = HashingTF(inputCol=swr.getOutputCol(), outputCol="features")
numeric_dfTrain = hashTF.transform(SwRemovedTrain).select(
    'Sentiment', 'MeaningfulWords', 'features')
numeric_dfTrain.show(3)
```

```
[Stage 27:>                                                         (0 + 1) / 1]

+---------+--------------------+--------------------+
|Sentiment|     MeaningfulWords|            features|
+---------+--------------------+--------------------+
|        0|[upset, update, f...|(262144,[59577,61...|
|        0|[@kenichan, dived...|(262144,[3924,283...|
|        0|[whole, body, fee...|(262144,[34121,80...|
+---------+--------------------+--------------------+
only showing top 3 rows
```

Train Our model using LogisticRegression

Modelling

In [23]:
```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(labelCol="Sentiment", featuresCol="features",
                        maxIter=10, regParam=0.01)
model = lr.fit(numeric_dfTrain)
print ("Training is done!")
```

```
2023-05-15 15:42:15,031 WARN memory.MemoryStore: Not enough space to cache rdd_87_1 in memory! (computed 65.
0 MiB so far)
2023-05-15 15:42:15,040 WARN storage.BlockManager: Persisting block rdd_87_1 to disk instead.
2023-05-15 15:42:15,342 WARN memory.MemoryStore: Not enough space to cache rdd_87_0 in memory! (computed 65.
0 MiB so far)
2023-05-15 15:42:15,343 WARN storage.BlockManager: Persisting block rdd_87_0 to disk instead.
2023-05-15 15:42:27,588 WARN netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.Nativ
eSystemBLAS
2023-05-15 15:42:27,589 WARN netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.Nativ
eRefBLAS


Training is done!
```

Prepare testing data

In [24]:
```
tokenized_dfTest = tokenizer.transform(data_test)
SwRemovedTest = swr.transform(tokenized_dfTest)
numericTest = hashTF.transform(SwRemovedTest).select(
    'Sentiment', 'MeaningfulWords', 'features')
numericTest.show(2)
```

```
[Stage 40:>                                                         (0 + 1) / 1]

+---------+--------------------+--------------------+
|Sentiment|     MeaningfulWords|            features|
+---------+--------------------+--------------------+
|        0|[@switchfoot, htt...|(262144,[38640,52...|
|        0|[@twittera, que, ...|(262144,[133107,1...|
+---------+--------------------+--------------------+
only showing top 2 rows
```

Prediction

In [25]:
```
prediction = model.transform(numericTest)
predictionFinal = prediction.select(
    "MeaningfulWords", "prediction", "Sentiment")
predictionFinal.show(10)
```

```
2023-05-15 15:43:21,522 WARN scheduler.DAGScheduler: Broadcasting large task binary with size 10.1 MiB
[Stage 41:>                                                         (0 + 1) / 1]

+--------------------+----------+---------+
|     MeaningfulWords|prediction|Sentiment|
+--------------------+----------+---------+
|[@switchfoot, htt...|       0.0|        0|
|[@twittera, que, ...|       4.0|        0|
|[@lettya, ahh, iv...|       0.0|        0|
|[@angry_barista, ...|       0.0|        0|
|[week, going, hoped]|       0.0|        0|
|[thought, sleepin...|       0.0|        0|
|[@fleurylis, eith...|       0.0|        0|
|[really, feel, li...|       0.0|        0|
|[checked, user, t...|       0.0|        0|
|[broadband, plan,...|       0.0|        0|
+--------------------+----------+---------+
only showing top 10 rows
```

Model Evaluation

In [26]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [27]:
```
#Accuracy
correctPrediction = predictionFinal.filter(
    predictionFinal['prediction'] == predictionFinal['Sentiment']).count()
totalData = predictionFinal.count()
print("correct prediction:", correctPrediction, ", total data:", totalData,
      ", accuracy:", correctPrediction/totalData)
```

```
2023-05-15 15:43:25,427 WARN scheduler.DAGScheduler: Broadcasting large task binary with size 10.1 MiB
[Stage 45:>                                                         (0 + 2) / 2]

correct prediction: 233568 , total data: 319918 , accuracy: 0.730087084815484
```

In [28]:
```
#Classification Report
y_true = predictionFinal.select(['Sentiment']).collect()
y_pred = predictionFinal.select(['prediction']).collect()
print(classification_report(y_true, y_pred))
```

```
2023-05-15 15:43:44,839 WARN scheduler.DAGScheduler: Broadcasting large task binary with size 10.1 MiB


              precision    recall  f1-score   support

           0       0.73      0.73      0.73    160147
           4       0.73      0.73      0.73    159771

    accuracy                           0.73    319918
   macro avg       0.73      0.73      0.73    319918
weighted avg       0.73      0.73      0.73    319918
```

In [29]:
```
#Confusion Matrix
cm = print(confusion_matrix(y_true, y_pred))
```

```
[[116701  43446]
 [ 42904 116867]]
```

Reference:
https://github.com/ardianumam/compilations/blob/master/ApacheSparkVideoSeries/08%20Sentiment%20Analysis%20in%20Spark.ipyn
(https://github.com/ardianumam/compilations/blob/master/ApacheSparkVideoSeries/08%20Sentiment%20Analysis%20in%20Spark.ipyr

https://www.projectpro.io/recipes/get-null-count-of-each-column-of-dataframe-pyspark-databricks
(https://www.projectpro.io/recipes/get-null-count-of-each-column-of-dataframe-pyspark-databricks)

In [ ]:
```
1
```