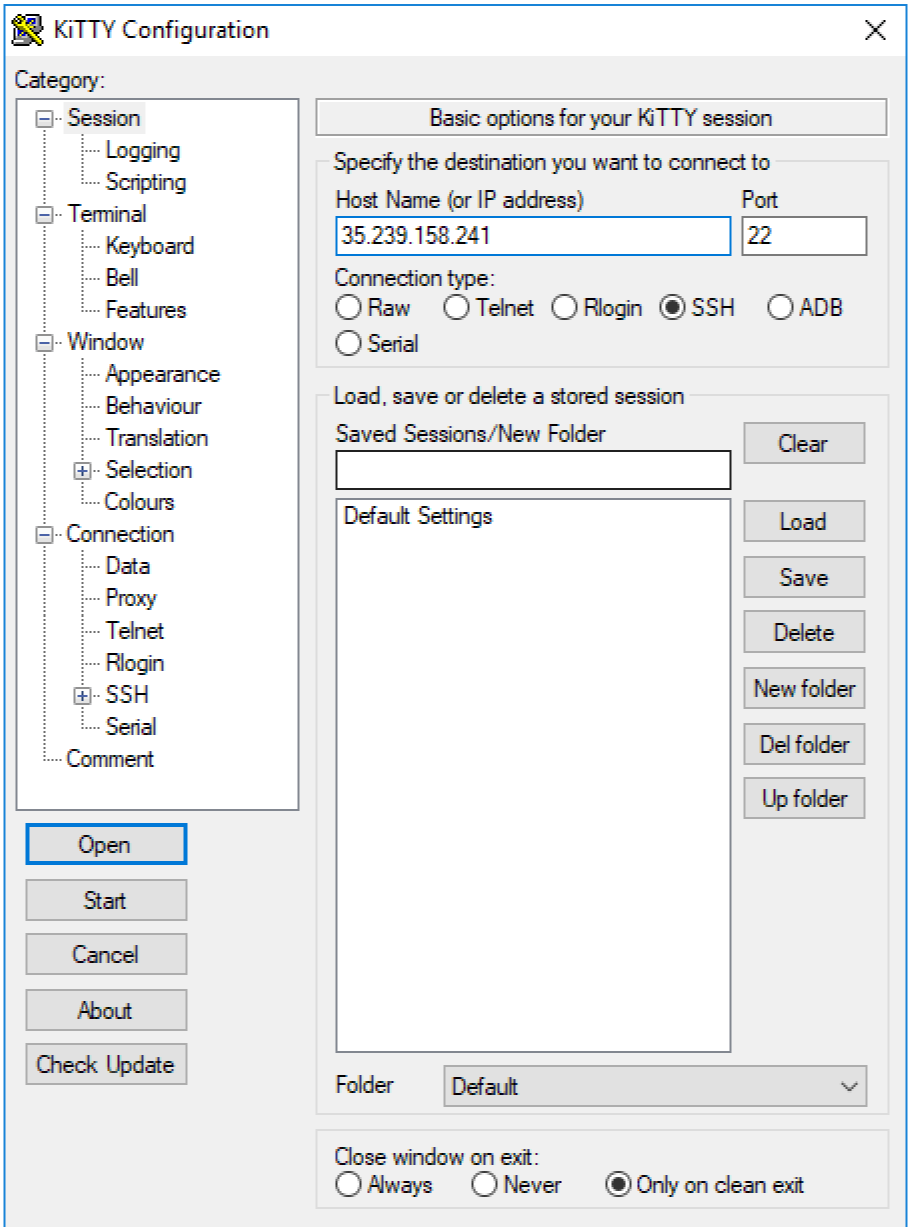
This document was a guideline of sentiment analysis using Spark. We’re using the 35.239.158.241:22 ssh through kitty, because of kitty stored all the session if our connection are not stable. Kitty ui was present at the below.


KiTTY Configuration

Basic options for your KiTTY session

Specify the destination you want to connect to

Host Name (or IP address): 35.239.158.241

Port: 22

Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ ADB ○ Serial

Load, save or delete a stored session

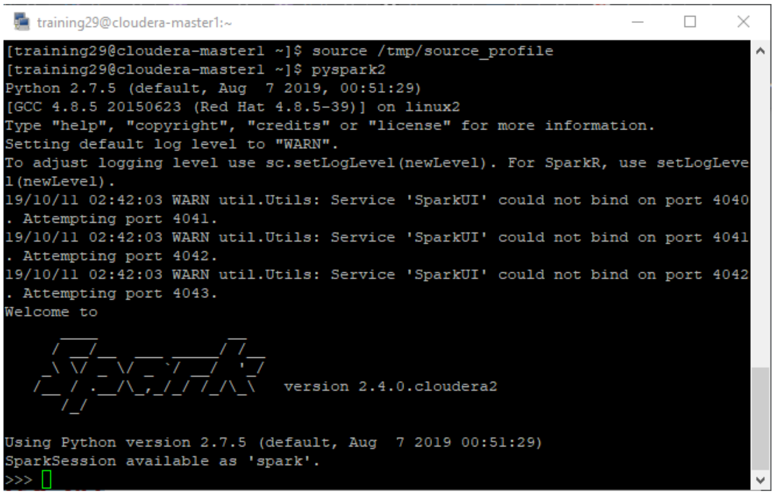
Saved Sessions/New Folder:

Default Settings

Folder: Default

Close window on exit: ○ Always ○ Never ● Only on clean exit

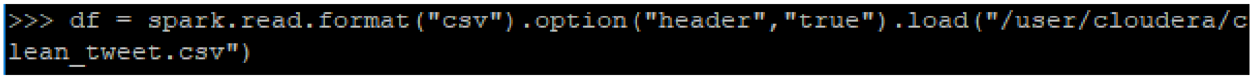
Login with training29 and password Cl0ud3r4* to start the spark from server, use command ‘source /tmp/source_profile’ to declare the source of spark and type pyspark2 to start the spark.



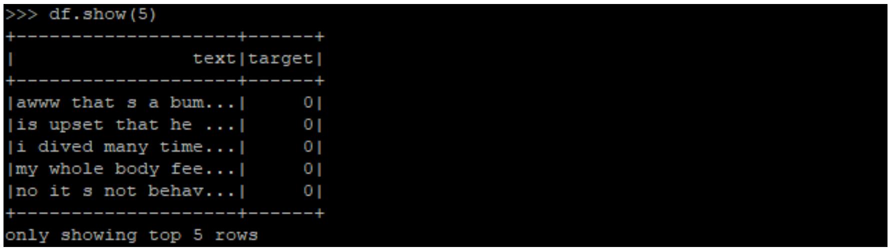
```
[training29@cloudera-master1 ~]$ source /tmp/source_profile
[training29@cloudera-master1 ~]$ pyspark2
Python 2.7.5 (default, Aug  7 2019, 00:51:29)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
19/10/11 02:42:03 WARN util.Utils: Service 'SparkUI' could not bind on port 4040
. Attempting port 4041.
19/10/11 02:42:03 WARN util.Utils: Service 'SparkUI' could not bind on port 4041
. Attempting port 4042.
19/10/11 02:42:03 WARN util.Utils: Service 'SparkUI' could not bind on port 4042
. Attempting port 4043.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.0.cloudera2
      /_/

Using Python version 2.7.5 (default, Aug  7 2019 00:51:29)
SparkSession available as 'spark'.
>>> 
```

1. **Import and Data Preparation**

Import the data.

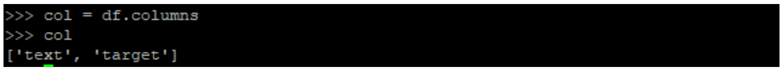


```
>>> df = spark.read.format("csv").option("header","true").load("/user/cloudera/c
lean_tweet.csv")
```



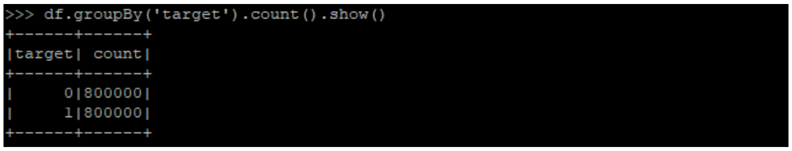
```
>>> df.show(5)
+--------------------+------+
|                text|target|
+--------------------+------+
|awww that s a bum...|     0|
|is upset that he ...|     0|
|i dived many time...|     0|
|my whole body fee...|     0|
|no it s not behav...|     0|
+--------------------+------+
only showing top 5 rows
```

Get the columns, and store into cols variable, there was a columns named as text and target. The text column stored a tweet data and the target stored a sentiment, 1 means positive, 0 means negative.



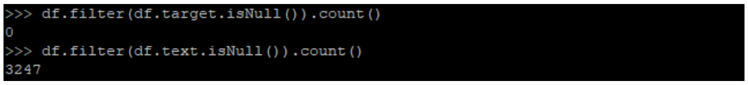
```
>>> col = df.columns
>>> col
['text', 'target']
```

Count the value of target data



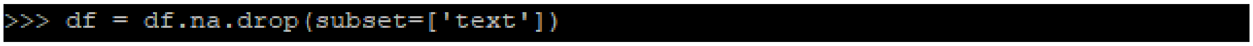
```
>>> df.groupBy('target').count().show()
+------+------+
|target| count|
+------+------+
|     0|800000|
|     1|800000|
+------+------+
```

Checking the null value from the data.



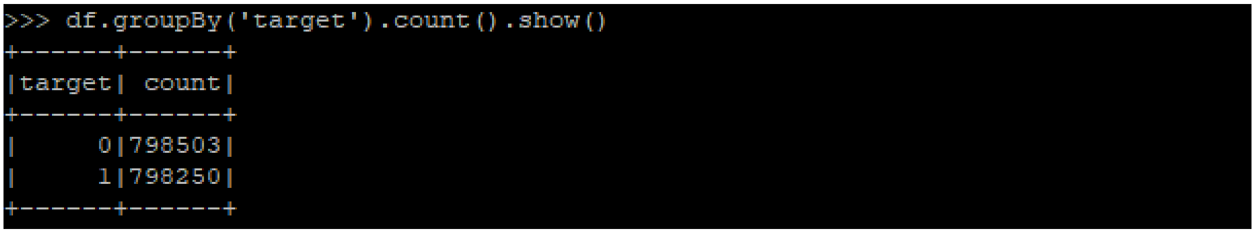
```
>>> df.filter(df.target.isNull()).count()
0
>>> df.filter(df.text.isNull()).count()
3247
```

There’s null value in the text data, so we must drop that data to improve our classification.



```
>>> df = df.na.drop(subset=['text'])
```

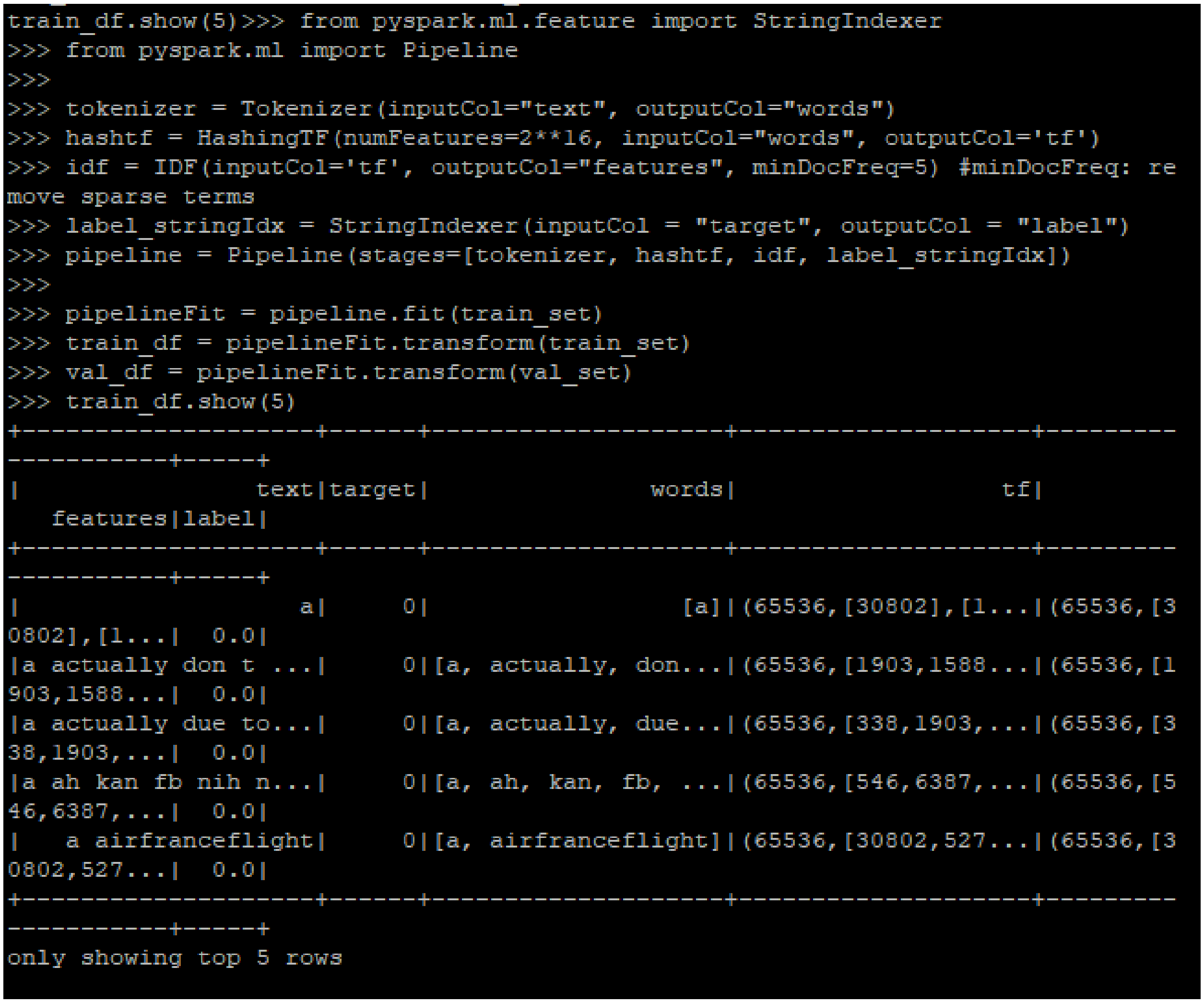
And then, we must check the count of data, to make sure that we done that we do.



```
>>> df.groupBy('target').count().show()
+------+------+
|target| count|
+------+------+
|     0|798503|
|     1|798250|
+------+------+
```

2. Feature Extraction

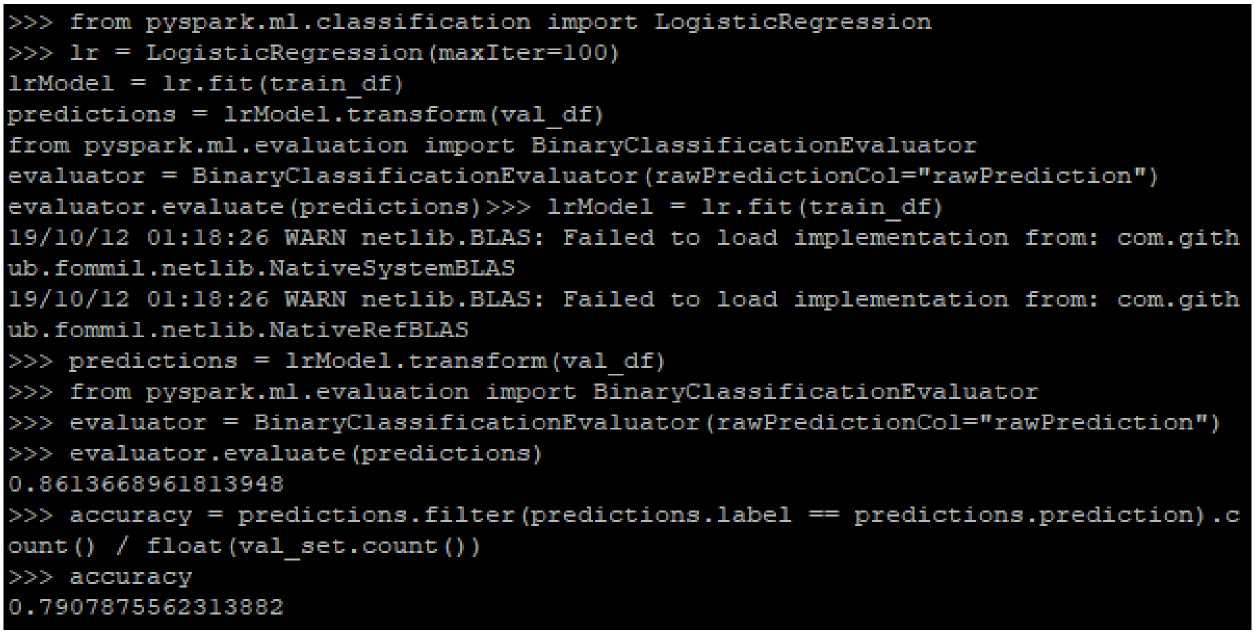
After data cleansing, we gonna be make a token with tokenizer library, and then extract the feature with hashing convectorizer and idf vectorizer.



```
train_df.show(5)>>> from pyspark.ml.feature import StringIndexer
>>> from pyspark.ml import Pipeline
>>>
>>> tokenizer = Tokenizer(inputCol="text", outputCol="words")
>>> hashtf = HashingTF(numFeatures=2**16, inputCol="words", outputCol='tf')
>>> idf = IDF(inputCol='tf', outputCol="features", minDocFreq=5) #minDocFreq: re
move sparse terms
>>> label_stringIdx = StringIndexer(inputCol = "target", outputCol = "label")
>>> pipeline = Pipeline(stages=[tokenizer, hashtf, idf, label_stringIdx])
>>>
>>> pipelineFit = pipeline.fit(train_set)
>>> train_df = pipelineFit.transform(train_set)
>>> val_df = pipelineFit.transform(val_set)
>>> train_df.show(5)
+--------------------+------+--------------------+--------------------+---------
-----------+-----+
|                text|target|               words|                  tf|
   features|label|
+--------------------+------+--------------------+--------------------+---------
-----------+-----+
|                   a|     0|                 [a]|(65536,[30802],[1...|(65536,[3
0802],[1...|  0.0|
|a actually don t ...|     0|[a, actually, don...|(65536,[1903,1588...|(65536,[1
903,1588...|  0.0|
|a actually due to...|     0|[a, actually, due...|(65536,[338,1903,...|(65536,[3
38,1903,...|  0.0|
|a ah kan fb nih n...|     0|[a, ah, kan, fb, ...|(65536,[546,6387,...|(65536,[5
46,6387,...|  0.0|
|   a airfranceflight|     0|[a, airfranceflight]|(65536,[30802,527...|(65536,[3
0802,527...|  0.0|
+--------------------+------+--------------------+--------------------+---------
-----------+-----+
only showing top 5 rows
```

3. Classification and evaluation

After the feature extracted, we import the machine learning library from pyspark.ml.classification, in this implementation we use logistic regression. From the result we get, the accuracy from model was 86 % and the accuracy from data test was 79%.



```
>>> from pyspark.ml.classification import LogisticRegression
>>> lr = LogisticRegression(maxIter=100)
lrModel = lr.fit(train_df)
predictions = lrModel.transform(val_df)
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
evaluator.evaluate(predictions)>>> lrModel = lr.fit(train_df)
19/10/12 01:18:26 WARN netlib.BLAS: Failed to load implementation from: com.gith
ub.fommil.netlib.NativeSystemBLAS
19/10/12 01:18:26 WARN netlib.BLAS: Failed to load implementation from: com.gith
ub.fommil.netlib.NativeRefBLAS
>>> predictions = lrModel.transform(val_df)
>>> from pyspark.ml.evaluation import BinaryClassificationEvaluator
>>> evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
>>> evaluator.evaluate(predictions)
0.8613668961813948
>>> accuracy = predictions.filter(predictions.label == predictions.prediction).c
ount() / float(val_set.count())
>>> accuracy
0.7907875562313882
```