Sentiment Analysis with PySpark

Satya Katragadda

March 22, 2018

Goal

- Perform Sentiment Analysis with spark
  - TF-IDF
    - N-gram
    - Count Vectorizer
  - Logistic Regression

Import the Data

```
import findspark
findspark.init()
import pyspark as ps
import warnings
from pyspark.sql import SQLContext
```

```
df = sqlContext.read.format('com.databricks.spark.csv').options(header='true', inferschema='true')
.load('project-capstone/Twitter_sentiment_analysis/clean_tweet.csv')
```

Import the Data -2

```
df.show(5)
```

```
+---+--------------------+------+
|_c0|                text|target|
+---+--------------------+------+
|  0|awww that bummer ...|     0|
|  1|is upset that he ...|     0|
|  2|dived many times ...|     0|
|  3|my whole body fee...|     0|
|  4|no it not behavin...|     0|
+---+--------------------+------+
```

Training and Testing Sets

```
(train_set, val_set, test_set) = df.randomSplit([0.98, 0.01, 0.01], seed = 2000)
```

Hashing TF and IDF

```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer
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: remove 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)
```

```
+---+--------------------+------+--------------------+--------------------+--------------------+--
---+
|_c0|                text|target|               words|                  tf|            features|la
bel|
+---+--------------------+------+--------------------+--------------------+--------------------+--
---+
|  0|awww that bummer ...|     0|[awww, that, bumm...|(65536,[8436,8847...|(65536,[8436,8847...|
0.0|
|  1|is upset that he ...|     0|[is, upset, that,...|(65536,[1444,2071...|(65536,[1444,2071...|
0.0|
|  2|dived many times ...|     0|[dived, many, tim...|(65536,[2548,2888...|(65536,[2548,2888...|
0.0|
|  3|my whole body fee...|     0|[my, whole, body,...|(65536,[158,11650...|(65536,[158,11650...|
0.0|
|  4|no it not behavin...|     0|[no, it, not, beh...|(65536,[1968,4488...|(65536,[1968,4488...|
0.0|
+---+--------------------+------+--------------------+--------------------+--------------------+--
---+
only showing top 5 rows
```

Sentiment Analysis

```
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)
```

```
0.8612433722998375
```

Another way to compute TF

```
%%time
from pyspark.ml.feature import CountVectorizer

tokenizer = Tokenizer(inputCol="text", outputCol="words")
cv = CountVectorizer(vocabSize=2**16, inputCol="words", outputCol='cv')
idf = IDF(inputCol='cv', outputCol="features", minDocFreq=5) #minDocFreq: remove sparse terms
label_stringIdx = StringIndexer(inputCol = "target", outputCol = "label")
lr = LogisticRegression(maxIter=100)
pipeline = Pipeline(stages=[tokenizer, cv, idf, label_stringIdx, lr])

pipelineFit = pipeline.fit(train_set)
predictions = pipelineFit.transform(val_set)
accuracy = predictions.filter(predictions.label == predictions.prediction).count() / float(val_set
.count())
roc_auc = evaluator.evaluate(predictions)

print "Accuracy Score: {0:.4f}".format(accuracy)
print "ROC-AUC: {0:.4f}".format(roc_auc)
```

```
Accuracy Score: 0.7982
ROC-AUC: 0.8681
CPU times: user 45.7 ms, sys: 15.1 ms, total: 60.8 ms
Wall time: 1min 15s
```

N-Gram Implementation

```
from pyspark.ml.feature import NGram, VectorAssembler
from pyspark.ml.feature import ChiSqSelector

def build_trigrams(inputCol=["text","target"], n=3):
    tokenizer = [Tokenizer(inputCol="text", outputCol="words")]
    ngrams = [
        NGram(n=i, inputCol="words", outputCol="{0}_grams".format(i))
        for i in range(1, n + 1)
    ]

    cv = [
        CountVectorizer(vocabSize=2**14,inputCol="{0}_grams".format(i),
            outputCol="{0}_tf".format(i))
        for i in range(1, n + 1)
    ]
    idf = [IDF(inputCol="{0}_tf".format(i), outputCol="{0}_tfidf".format(i), minDocFreq=5) for i i
n range(1, n + 1)]

    assembler = [VectorAssembler(
        inputCols=["{0}_tfidf".format(i) for i in range(1, n + 1)],
        outputCol="rawFeatures"
    )]
    label_stringIdx = [StringIndexer(inputCol = "target", outputCol = "label")]
    selector = [ChiSqSelector(numTopFeatures=2**14,featuresCol='rawFeatures', outputCol="features"
)]
    lr = [LogisticRegression(maxIter=100)]
    return Pipeline(stages=tokenizer + ngrams + cv + idf+ assembler + label_stringIdx+selector+lr)
```

N-Gram Implementation

```
%%time
trigram_pipelineFit = build_trigrams().fit(train_set)
predictions = trigram_pipelineFit.transform(val_set)
accuracy = predictions.filter(predictions.label == predictions.prediction).count() / float(dev_set
.count())
roc_auc = evaluator.evaluate(predictions)

# print accuracy, roc_auc
print "Accuracy Score: {0:.4f}".format(accuracy)
print "ROC-AUC: {0:.4f}".format(roc_auc)
```

```
Accuracy Score: 0.8129
ROC-AUC: 0.8884
CPU times: user 2.11 s, sys: 935 ms, total: 3.04 s
Wall time: 4h 1min 9s
```

Questions: satya@Louisiana.edu

Acknowledgements:

https://github.com/tthustla/setiment_analysis_pyspark/blob/master/Sentiment%20Analysis%20with%20PySpark.ipynb