Xi'an Jiaotong-Liverpool University
西交利物浦大学

DTS303TC Big Data Security and Analytics

Laboratory Session 5 – Sentiment Analysis with PySpark

(Duration: 2 hours)

Overview:

Twitter and other micro-blogging websites have long been a source of information about consumer sentiment, especially since people tend to discuss various issues as well as post their feelings, complaints, or praise of various products and services on Twitter.

In this activity, you will be able to perform the following in PySpark:

- Build different sentiment classifiers
- Measure classification performance using metrics

PART 1: Sentiment Analysis with PySpark

Task Instructions

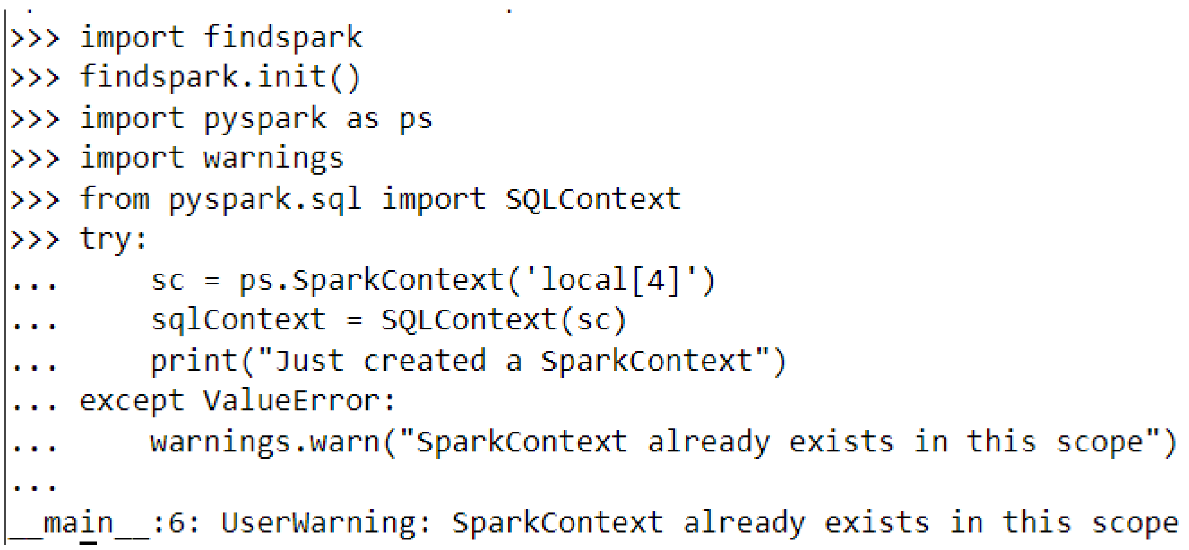
In this part, you will be programming in Pyspark.

Data Preparation

The dataset to be used is annotated Tweets from "Sentiment140". It originated from a Stanford research project. We will use the processed dataset in this lab.
Dataset download link: https://www.heyibo.net/usr/uploads/2022/09/3691850101.zip

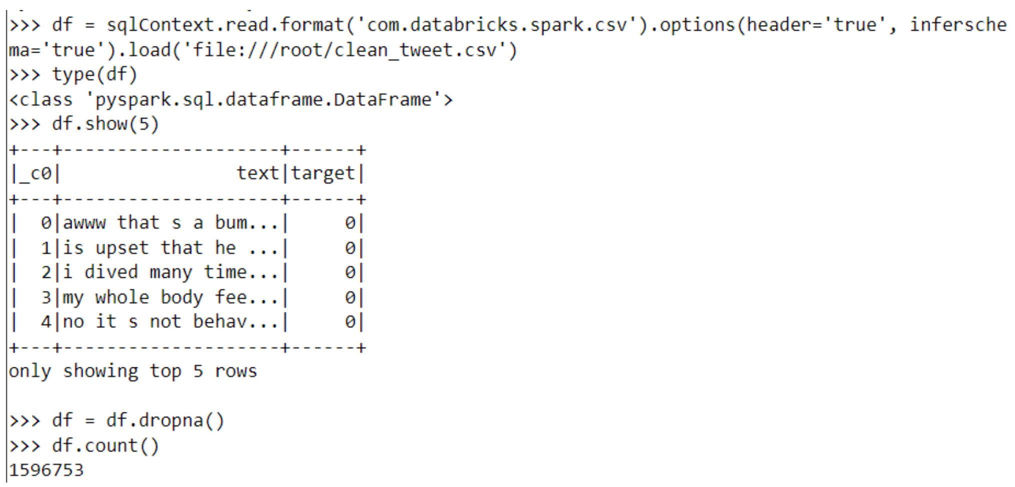
Using PySpark

Step 1: The first step in any Apache programming is to create a SparkContext. SparkContext is needed when we want to execute operations in a cluster. SparkContext tells Spark how and where to access a cluster. It is the first step to connect with Apache Cluster.



```
>>> import findspark
>>> findspark.init()
>>> import pyspark as ps
>>> import warnings
>>> from pyspark.sql import SQLContext
>>> try:
...     sc = ps.SparkContext('local[4]')
...     sqlContext = SQLContext(sc)
...     print("Just created a SparkContext")
... except ValueError:
...     warnings.warn("SparkContext already exists in this scope")
...
__main__:6: UserWarning: SparkContext already exists in this scope
```

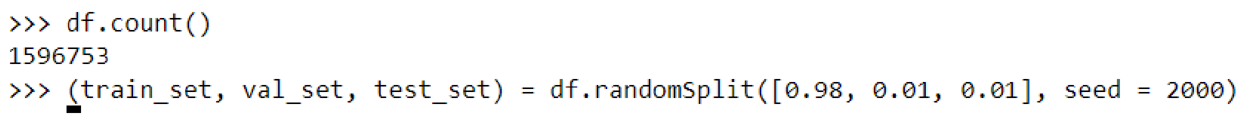
Step 2: Import dataset.



```
>>> df = sqlContext.read.format('com.databricks.spark.csv').options(header='true', inferschema='true').load('file:///root/clean_tweet.csv')
>>> type(df)
<class 'pyspark.sql.dataframe.DataFrame'>
>>> df.show(5)
+---+--------------------+------+
|_c0|                text|target|
+---+--------------------+------+
|  0|awww that s a bum...|     0|
|  1|is upset that he ...|     0|
|  2|i dived many time...|     0|
|  3|my whole body fee...|     0|
|  4|no it s not behav...|     0|
+---+--------------------+------+
only showing top 5 rows

>>> df = df.dropna()
>>> df.count()
1596753
```

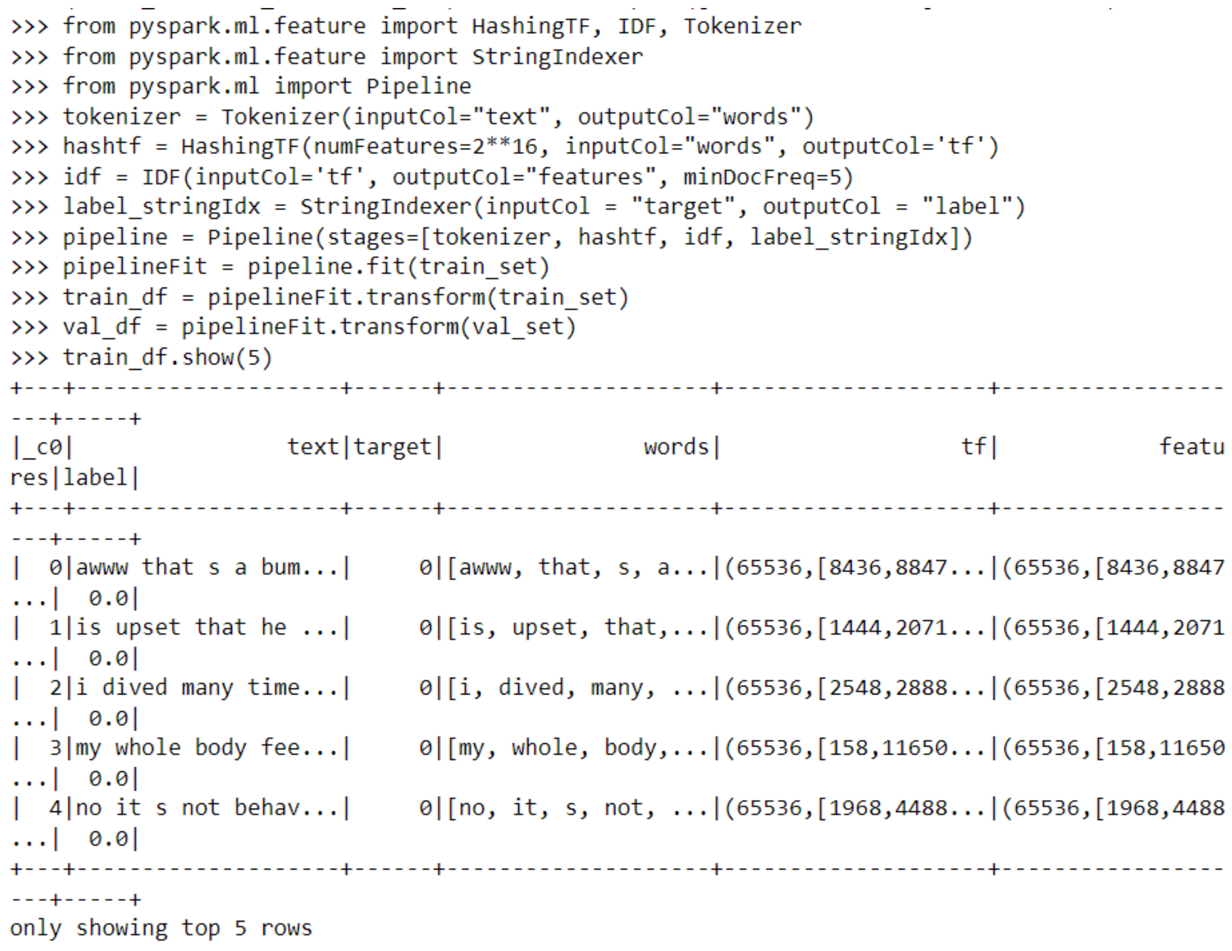
After successfully loading the data as Spark Dataframe, we can take a peek at the data by calling .show(), which is equivalent to Pandas .head(). After dropping NA, we have a bit less than 1.6 million Tweets. We will split this into three parts; training, validation, test. Since there are around 1.6 million entries, 1% each for validation and test set will be enough to test the models.



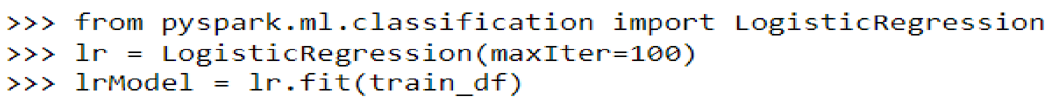
```
>>> df.count()
1596753
>>> (train_set, val_set, test_set) = df.randomSplit([0.98, 0.01, 0.01], seed = 2000)
```

Step 3: HashingTF + IDF + Logistic Regression.

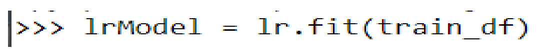
TF-IDF with Logistic Regression is quite a strong combination, and showed robust performance, as high as Word2Vec + Convolutional Neural Network model. In this lab, we will try to implement TF-IDF + Logistic Regression model with PySpark.



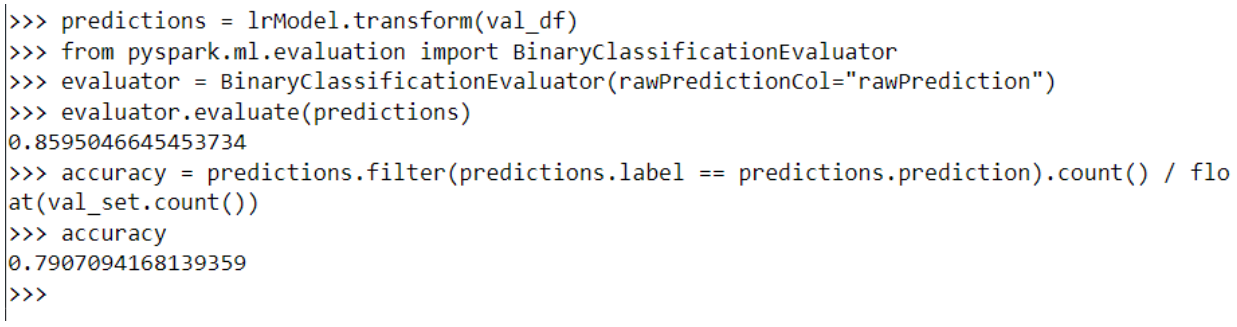
```
>>> from pyspark.ml.feature import HashingTF, IDF, Tokenizer
>>> from pyspark.ml.feature import StringIndexer
>>> from pyspark.ml import Pipeline
>>> tokenizer = Tokenizer(inputCol="text", outputCol="words")
>>> hashtf = HashingTF(numFeatures=2**16, inputCol="words", outputCol='tf')
>>> idf = IDF(inputCol='tf', outputCol="features", minDocFreq=5)
>>> 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|            featu
res|label|
+---+--------------------+------+--------------------+--------------------+----------------
---+-----+
|  0|awww that s a bum...|     0|[awww, that, s, a...|(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|i dived many time...|     0|[i, dived, many, ...|(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 s not behav...|     0|[no, it, s, not, ...|(65536,[1968,4488...|(65536,[1968,4488
...|  0.0|
+---+--------------------+------+--------------------+--------------------+----------------
---+-----+
only showing top 5 rows
```



```
>>> from pyspark.ml.classification import LogisticRegression
>>> lr = LogisticRegression(maxIter=100)
>>> lrModel = lr.fit(train_df)
```



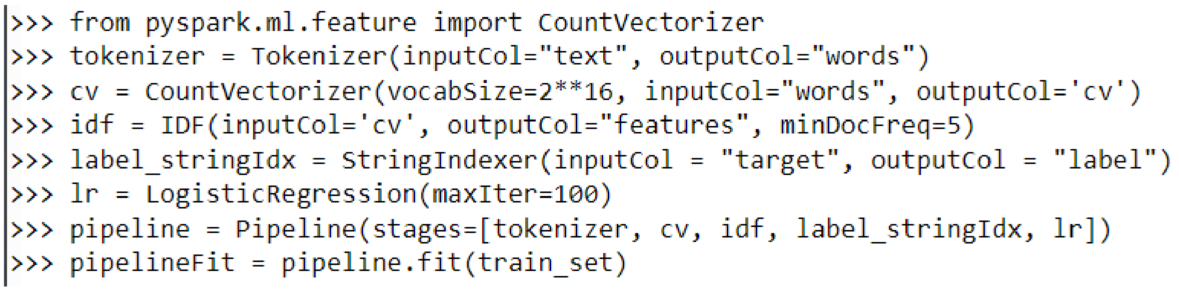
```
>>> lrModel = lr.fit(train_df)
```



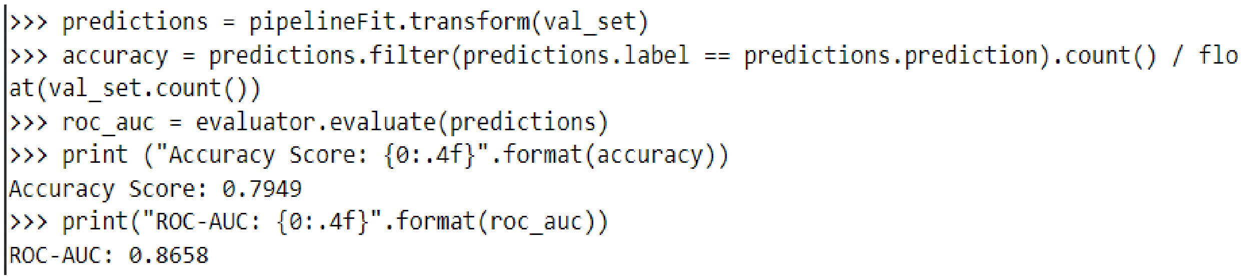
```
>>> predictions = lrModel.transform(val_df)
>>> from pyspark.ml.evaluation import BinaryClassificationEvaluator
>>> evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
>>> evaluator.evaluate(predictions)
0.8595046645453734
>>> accuracy = predictions.filter(predictions.label == predictions.prediction).count() / flo
at(val_set.count())
>>> accuracy
0.7907094168139359
>>>
```

Step 4: CountVectorizer + IDF + Logistic Regression.

There's another way that you can get term frequency for IDF (Inverse Document Frequency) calculation. It is CountVectorizer in SparkML. Apart from the reversibility of the features (vocabularies), there is an important difference in how each of them filters top features. In case of HashingTF it is dimensionality reduction with possible collisions. CountVectorizer discards infrequent tokens.



```
>>> 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)
>>> 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() / flo
at(val_set.count())
>>> roc_auc = evaluator.evaluate(predictions)
>>> print ("Accuracy Score: {0:.4f}".format(accuracy))
Accuracy Score: 0.7949
>>> print("ROC-AUC: {0:.4f}".format(roc_auc))
ROC-AUC: 0.8658
```

Step 5: Compare the results achieved by the two combinations.