Chapter 11: NLP

Ex2: Musical Instruments

Requirement: Build a reviewer filter. Use the various NLP tools and a new classifier, Naive Bayes, to predict if one reviewText is like (overall >= 4)/don't like (overall <=2) /neutral (2<overall<4)

- Dataset: Musical_Instruments_5.json

In [1]:
```
import findspark
findspark.init()
```

In [2]:
```
from pyspark.sql import SparkSession
```

In [3]:
```
spark = SparkSession.builder.appName('nlp_musical').getOrCreate()
```

In [4]:
```
data = data = spark.read.json("Musical_Instruments_5.json")
```

In [5]:
```
data.show(5)
```

```
+----------+--------+-------+--------------------+-----------+--------------+--
------------------+--------------------+--------------+
|      asin| helpful|overall|          reviewText| reviewTime|    reviewerID|
reviewerName|             summary|unixReviewTime|
+----------+--------+-------+--------------------+-----------+--------------+--
------------------+--------------------+--------------+
|1384719342|  [0, 0]|    5.0|Not much to write...|02 28, 2014|A2IBPI20UZIR0U|ca
ssandra tu "Yea...|                good|    1393545600|
|1384719342|[13, 14]|    5.0|The product does ...|03 16, 2013|A14VAT5EAX3D9S|
Jake|                Jake|    1363392000|
|1384719342|  [1, 1]|    5.0|The primary job o...|08 28, 2013|A195EZSQDW3E21|Ri
ck Bennette "Ri...|It Does The Job Well|    1377648000|
|1384719342|  [0, 0]|    5.0|Nice windscreen p...|02 14, 2014|A2C00NNG1ZQQG2|Ru
styBill "Sunday...|GOOD WINDSCREEN F...|    1392336000|
|1384719342|  [0, 0]|    5.0|This pop filter i...|02 21, 2014| A94QU4C90B1AX|
SEAN MASLANKA|No more pops when...|    1392940800|
+----------+--------+-------+--------------------+-----------+--------------+--
------------------+--------------------+--------------+
only showing top 5 rows
```

In [6]:
```
from pyspark.sql.functions import *
```

In [7]:
```
data = data.withColumn('class', when(data.overall >=4, "like")
                                .when(data.overall <= 2, "not_like")
                                .otherwise("neutral"))
```

```
In [8]: data = data.select("reviewText", "overall", "class")
```

Clean and Prepare the Data

** Create a new length feature: **

```
In [9]: from pyspark.sql.functions import length
```

```
In [10]: data = data.withColumn('length',length(data['reviewText']))
```

```
In [11]: data.show(10)
```

```
+--------------------+-------+------+------+
|          reviewText|overall| class|length|
+--------------------+-------+------+------+
|Not much to write...|    5.0|  like|   268|
|The product does ...|    5.0|  like|   544|
|The primary job o...|    5.0|  like|   436|
|Nice windscreen p...|    5.0|  like|   206|
|This pop filter i...|    5.0|  like|   159|
|So good that I bo...|    5.0|  like|   234|
|I have used monst...|    5.0|  like|   191|
|I now use this ca...|    3.0|neutral|   845|
|Perfect for my Ep...|    5.0|  like|   201|
|Monster makes the...|    5.0|  like|   217|
+--------------------+-------+------+------+
only showing top 10 rows
```

```
In [12]: # Pretty Clear Difference
         data.groupby('class').mean().show()
```

```
+--------+------------------+-----------------+
|   class|      avg(overall)|      avg(length)|
+--------+------------------+-----------------+
|not_like|1.5353319057815846|579.2055674518201|
| neutral|               3.0|579.2111398963731|
|    like|4.7690090888938155|473.1188206606074|
+--------+------------------+-----------------+
```

```
In [13]: data.groupby('class').count().show()
```

```
+--------+-----+
|   class|count|
+--------+-----+
|not_like|  467|
| neutral|  772|
|    like| 9022|
+--------+-----+
```

Feature Transformations

In [14]:

```
from pyspark.ml.feature import Tokenizer,StopWordsRemover
from pyspark.ml.feature import CountVectorizer,IDF,StringIndexer
tokenizer = Tokenizer(inputCol="reviewText", outputCol="token_text")
stopremove = StopWordsRemover(inputCol='token_text',outputCol='stop_tokens')
count_vec = CountVectorizer(inputCol='stop_tokens',outputCol='c_vec')
idf = IDF(inputCol="c_vec", outputCol="tf_idf")
class_to_num = StringIndexer(inputCol='class',outputCol='label')
```

In [15]:

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.linalg import Vector
```

In [16]:

```
clean_up = VectorAssembler(inputCols=['tf_idf','length'],outputCol='features')
```

The Model

We'll use Naive Bayes, but feel free to play around with this choice!

In [17]:

```
from pyspark.ml.classification import NaiveBayes
```

In [18]:

```
# Use defaults
nb = NaiveBayes()
```

Pipeline

In [19]:

```
from pyspark.ml import Pipeline
```

In [20]:

```
data_prep_pipe = Pipeline(stages=[class_to_num,tokenizer,
                                  stopremove,count_vec,
                                  idf,clean_up])
```

In [21]:

```
cleaner = data_prep_pipe.fit(data)
```

In [22]:

```
clean_data = cleaner.transform(data)
```

Training and Evaluation!

In [23]:

```
clean_data = clean_data.select(['label','features'])
```

In [24]:
```
clean_data.show(10) # 0: like, 1: neutral, 2: not_like
```

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  0.0|(51949,[3,12,14,3...|
|  0.0|(51949,[2,3,12,16...|
|  0.0|(51949,[11,19,44,...|
|  0.0|(51949,[18,37,57,...|
|  0.0|(51949,[2,122,132...|
|  0.0|(51949,[0,5,15,21...|
|  0.0|(51949,[5,16,29,1...|
|  1.0|(51949,[1,3,4,8,1...|
|  0.0|(51949,[0,3,12,33...|
|  0.0|(51949,[1,6,15,52...|
+-----+--------------------+
only showing top 10 rows
```

In [25]:
```
(training,testing) = clean_data.randomSplit([0.7,0.3])
```

In [26]:
```
training.groupBy("label").count().show()
```

```
+-----+-----+
|label|count|
+-----+-----+
|  0.0| 6273|
|  1.0|  538|
|  2.0|  321|
+-----+-----+
```

In [27]:
```
testing.groupBy("label").count().show()
```

```
+-----+-----+
|label|count|
+-----+-----+
|  0.0| 2749|
|  1.0|  234|
|  2.0|  146|
+-----+-----+
```

In [28]:
```
predictor = nb.fit(training)
```

In [29]:
```
data.printSchema()
```

```
root
 |-- reviewText: string (nullable = true)
 |-- overall: double (nullable = true)
 |-- class: string (nullable = false)
 |-- length: integer (nullable = true)
```

In [30]:

```
test_results = predictor.transform(testing)
```

In [31]:

```
test_results.show(10)
```

```
+-----+--------------------+--------------------+--------------------+---------
-+
|label|            features|       rawPrediction|         probability|predictio
n|
+-----+--------------------+--------------------+--------------------+---------
-+
|  0.0|(51949,[0,1,2,3,4...|[-8037.9167293525...|[3.74336796460930...|       2.
0|
|  0.0|(51949,[0,1,2,3,4...|[-11977.791289462...|[1.0,1.0209759137...|       0.
0|
|  0.0|(51949,[0,1,2,3,4...|[-4701.5867096432...|[0.99999999999999...|       0.
0|
|  0.0|(51949,[0,1,2,3,4...|[-9502.4876633284...|[4.53700693583550...|       1.
0|
|  0.0|(51949,[0,1,2,3,4...|[-5436.0778379955...|[1.0,1.1410818254...|       0.
0|
|  0.0|(51949,[0,1,2,3,4...|[-22250.822977014...|[1.82927389637432...|       1.
0|
|  0.0|(51949,[0,1,2,3,4...|[-12613.007139582...|[8.30105759458827...|       1.
0|
|  0.0|(51949,[0,1,2,3,4...|[-3678.0900065707...|[1.0,5.2918918521...|       0.
0|
|  0.0|(51949,[0,1,2,3,4...|[-7685.3563363984...|[5.36237204440014...|       1.
0|
|  0.0|(51949,[0,1,2,3,4...|[-5670.9474482192...|[0.97550306223960...|       0.
0|
+-----+--------------------+--------------------+--------------------+---------
-+
only showing top 10 rows
```

In [32]:

```
# Create a confusion matrix
test_results.groupBy('label', 'prediction').count().show()
```

```
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  2.0|       0.0|   59|
|  1.0|       1.0|   70|
|  0.0|       1.0|  495|
|  1.0|       0.0|  141|
|  2.0|       2.0|   43|
|  2.0|       1.0|   44|
|  1.0|       2.0|   23|
|  0.0|       0.0| 2056|
|  0.0|       2.0|  198|
+-----+----------+-----+
```

In [33]:

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

In [34]:

```
acc_eval = MulticlassClassificationEvaluator()
acc = acc_eval.evaluate(test_results)
print("Accuracy of model at predicting: {}".format(acc))
```

```
Accuracy of model at predicting: 0.7440091662279948
```

- Not very good result! (~74%)
- Solution: Try switching out the classification models! Or even try to come up with other engineered features!...

Use LogisticRegression/Random Forest

Logistic Regression

In [35]:

```
from pyspark.ml.classification import RandomForestClassifier, LogisticRegression
```

In [36]:

```
lg = LogisticRegression(maxIter=20, regParam=0.3, elasticNetParam=0)
```

In [37]:

```
predictor_1 = lg.fit(training)
```

In [38]:

```
test_results_1 = predictor_1.transform(testing)
```

In [39]:

```
# Create a confusion matrix
test_results_1.groupBy('label', 'prediction').count().show()
```

```
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  2.0|       0.0|  144|
|  1.0|       1.0|    1|
|  0.0|       1.0|    4|
|  1.0|       0.0|  233|
|  2.0|       2.0|    1|
|  2.0|       1.0|    1|
|  0.0|       0.0| 2744|
|  0.0|       2.0|    1|
+-----+----------+-----+
```

In [40]:

```
acc_eval = MulticlassClassificationEvaluator()
acc_1 = acc_eval.evaluate(test_results_1)
print("Accuracy of model at predicting: {}".format(acc_1))
```

```
Accuracy of model at predicting: 0.8226357404655656
```

In [41]:

```
## Higher accuracy but not better result!!!
```

Random forest

Random forest

In [42]:
```
rf = RandomForestClassifier(labelCol="label", \
                            featuresCol="features", \
                            numTrees = 500, \
                            maxDepth = 5, \
                            maxBins = 64)
```

In [43]:
```
predictor_2 = rf.fit(training)
```

In [44]:
```
test_results_2 = predictor_2.transform(testing)
```

In [45]:
```
# Create a confusion matrix
test_results_2.groupBy('label', 'prediction').count().show()
```

```
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  2.0|       0.0|  146|
|  1.0|       0.0|  234|
|  0.0|       0.0| 2749|
+-----+----------+-----+
```

In [46]:
```
test_results_2.groupBy('prediction').count().show()
```

```
+----------+-----+
|prediction|count|
+----------+-----+
|       0.0| 3129|
+----------+-----+
```

In [47]:
```
acc_eval = MulticlassClassificationEvaluator()
acc_2 = acc_eval.evaluate(test_results_2)
print("Accuracy of model at predicting: {}".format(acc_2))
```

```
Accuracy of model at predicting: 0.8217587374516523
```

In [48]:
```
## Higher accuracy but too bad result!!!
```

Need to resample data

In [49]:

```
like_df = training.filter(col("label") == 0)
neutral_df = training.filter(col("label") == 1)
not_like_df = training.filter(col("label") == 2)
ratio_1 = int(like_df.count()/neutral_df.count())
ratio_2 = int(like_df.count()/not_like_df.count())
print("ratio like/neutral: {}".format(ratio_1))
print("ratio like/not_like: {}".format(ratio_2))
```

```
ratio like/neutral: 11
ratio like/not_like: 19
```

In [50]:

```
# resample neutral
a1 = range(ratio_1)
# duplicate the minority rows
oversampled_neutral_df = neutral_df.withColumn("dummy",
                                               explode(array([lit(x) for x in a1
                                               .drop('dummy')
# combine both oversampled minority rows and previous majority rows
combined_df = like_df.unionAll(oversampled_neutral_df)
combined_df.show(10)
```

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  0.0|(51949,[0],[1.025...|
|  0.0|(51949,[0],[1.025...|
|  0.0|(51949,[0,1,2,3,4...|
|  0.0|(51949,[0,1,2,3,4...|
|  0.0|(51949,[0,1,2,3,4...|
|  0.0|(51949,[0,1,2,3,4...|
|  0.0|(51949,[0,1,2,3,4...|
|  0.0|(51949,[0,1,2,3,4...|
|  0.0|(51949,[0,1,2,3,4...|
|  0.0|(51949,[0,1,2,3,4...|
+-----+--------------------+
only showing top 10 rows
```

In [51]:

```
combined_df.groupBy("label").count().show()
```

```
+-----+-----+
|label|count|
+-----+-----+
|  0.0| 6273|
|  1.0| 5918|
+-----+-----+
```

In [52]:
```
# resample not_like
a2 = range(ratio_2)
# duplicate the minority rows
oversampled_notlike_df = not_like_df.withColumn("dummy",
                                                explode(array([lit(x) for x in a
                                                .drop('dummy')
# combine both oversampled minority rows and previous majority rows
combined_df = combined_df.unionAll(oversampled_notlike_df)
combined_df.show(10)
```

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  0.0|(51949,[0],[1.025...|
|  0.0|(51949,[0],[1.025...|
|  0.0|(51949,[0,1,2,3,4...|
|  0.0|(51949,[0,1,2,3,4...|
|  0.0|(51949,[0,1,2,3,4...|
|  0.0|(51949,[0,1,2,3,4...|
|  0.0|(51949,[0,1,2,3,4...|
|  0.0|(51949,[0,1,2,3,4...|
|  0.0|(51949,[0,1,2,3,4...|
|  0.0|(51949,[0,1,2,3,4...|
+-----+--------------------+
only showing top 10 rows
```

In [53]:
```
combined_df.groupBy("label").count().show()
```

```
+-----+-----+
|label|count|
+-----+-----+
|  0.0| 6273|
|  1.0| 5918|
|  2.0| 6099|
+-----+-----+
```

Naive Bayer

In [54]:
```
predictor_4 = nb.fit(combined_df)
```

In [55]:
```
test_results_4 = predictor_4.transform(testing)
```

In [56]:
```
test_results_4.groupBy('label', 'prediction').count().show()
```

```
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  2.0|       0.0|  121|
|  1.0|       1.0|   30|
|  0.0|       1.0|  125|
|  1.0|       0.0|  193|
|  2.0|       2.0|   14|
|  2.0|       1.0|   11|
|  1.0|       2.0|   11|
|  0.0|       0.0| 2566|
|  0.0|       2.0|   58|
+-----+----------+-----+
```

In [57]:
```
acc_eval = MulticlassClassificationEvaluator()
acc_4 = acc_eval.evaluate(test_results_4)
print("Accuracy of model at predicting: {}".format(acc_4))
```

```
Accuracy of model at predicting: 0.8179081830591779
```

Logistic Regression

In [58]:
```
predictor_5 = lg.fit(combined_df)
```

In [59]:
```
test_results_5 = predictor_5.transform(testing)
```

In [60]:
```
test_results_5.groupBy('label', 'prediction').count().show()
```

```
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  2.0|       0.0|  120|
|  1.0|       1.0|   25|
|  0.0|       1.0|   57|
|  1.0|       0.0|  205|
|  2.0|       2.0|   17|
|  2.0|       1.0|    9|
|  1.0|       2.0|    4|
|  0.0|       0.0| 2678|
|  0.0|       2.0|   14|
+-----+----------+-----+
```

In [61]:
```
acc_eval = MulticlassClassificationEvaluator()
acc_5 = acc_eval.evaluate(test_results_5)
print("Accuracy of model at predicting: {}".format(acc_5))
```

```
Accuracy of model at predicting: 0.8383409421434534
```

Random Forest

```
In [62]: predictor_3 = rf.fit(combined_df)
```

```
In [63]: test_results_3 = predictor_3.transform(testing)
```

```
In [64]: test_results_3.groupBy('label').count().show()
```

```
+-----+-----+
|label|count|
+-----+-----+
|  0.0| 2749|
|  1.0|  234|
|  2.0|  146|
+-----+-----+
```

```
In [65]: # Create a confusion matrix
         test_results_3.groupBy('label', 'prediction').count().show()
```

```
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|  2.0|       0.0|   95|
|  1.0|       1.0|    9|
|  0.0|       1.0|   23|
|  1.0|       0.0|  211|
|  2.0|       2.0|   45|
|  2.0|       1.0|    6|
|  1.0|       2.0|   14|
|  0.0|       0.0| 2645|
|  0.0|       2.0|   81|
+-----+----------+-----+
```

```
In [66]: acc_eval = MulticlassClassificationEvaluator()
         acc_3 = acc_eval.evaluate(test_results_3)
         print("Accuracy of model at predicting: {}".format(acc_3))
```

```
Accuracy of model at predicting: 0.8349933724328554
```

```
In [67]: ## Higher accuracy and better result. But not very good!
         ## Do you have another solution???
```