



BIG DATA IN MACHINE LEARNING

Bài 11: NLP

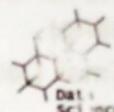
Phòng LT & Mang

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1000000/>

1990-01-01 00:00:00 1990-01-01 00:00:00

trung vào mảng số trắc

2020



Nội dung



- ## 1. Giới thiệu

Giới thiệu

❑ **Xử lý ngôn ngữ tự nhiên (NLP) là thành phần chính trong nhiều hệ thống khoa học dữ liệu (phải hiểu hoặc suy luận về một văn bản).**

• Ví dụ một số ứng dụng:

- Sentiment analysis
- Clustering News Articles (phân cụm tin tức)
- Suggesting similar books (đề xuất các sách tương tự)
- Grouping Legal Documents (phân nhóm tài liệu)
- Analyzing Consumer Feedback (phân tích phản hồi của khách hàng)
- Spam Email Detection (phát hiện mail rác)

Giới thiệu

• NLP rất cần thiết, số lượng ứng dụng AI có NLP ngày càng tăng. Trích xuất thông tin chính xác từ văn bản (free text) là điều bắt buộc nếu chúng ta xây dựng chatbot, tìm kiếm thông qua cơ sở dữ liệu, phân loại dịch vụ khách hàng hoặc các cuộc gọi bán hàng, trích xuất sự kiện từ báo cáo tài chính...

- NLP là một lĩnh vực của Machine Learning tập trung vào việc tạo ra các model từ nguồn dữ liệu văn bản.
- Bản thân NLP có nhiều thách thức nhưng cũng có thuận lợi vì có nhiều thuật toán và tính năng độc đáo riêng của nó (vì vậy, chúng ta sẽ chỉ tập trung vào một số tool)

- Một số quy trình NLP cơ bản
 - Compile all documents (Corpus, tổng hợp tất cả tài liệu)
 - Featurize the words to numerics (chuyển đổi từ thành số)
 - Compare features of documents (so sánh thuộc tính của các tài liệu)

Nội dung

1. Giới thiệu

2. Các công cụ cho NLP

dữ liệu (như hình ảnh, video, âm thanh và dữ liệu văn bản).

Ví dụ về các công cụ có thể nói là:

- Tokenization (tách từ)
- Stopwords (từ không có ý nghĩa)
- Stemming (giảm từ thành gốc)
- Lemmatization (giảm từ thành gốc)
- CountVectorizer (tạo ma trận tf-idf)



Các công cụ cho NLP

Spark có rất nhiều công cụ

pyspark.ml.feature để hỗ trợ toàn bộ quá trình này và làm cho công việc trở nên dễ dàng hơn, như:

- **Tokenizer** → chuyển văn bản thành tokens
- **StopWordsRemover** → bỏ stop word
- **NGram** → tạo các nhóm từ ghép
- **TF-IDF** → tạo vector cho các văn bản 1 cách công bằng
- **CountVectorizer** → tạo ma trận tf-idf
- ...



❑ Tokenizer

- Tokenization là quá trình phân định và (có thể) phân loại các phần của một chuỗi các ký tự đầu vào. Các token kết quả sau đó được chuyển sang một số hình thức khác nhau. Quá trình có thể được coi là một sub-task của việc phân tích cú pháp đầu vào.



https://en.wikipedia.org/wiki/Lexical_analysis#Tokenization

Big Data in Machine Learning

9

Các công cụ cho NLP

- Hiểu đơn giản, Tokenization là quá trình lấy văn bản (ví dụ như một câu) và chia nó thành các term riêng lẻ (thường là các từ). Một lớp Tokenizer cung cấp chức năng này.



Các công cụ cho NLP

- RegexTokenizer cho phép tokenization nâng cao hơn dựa trên việc kết hợp biểu thức chính quy (regex). Mặc định, tham số "pattern" (regex, mặc định: "\s+") được sử dụng làm dấu phân cách (delimiter) để phân tách văn bản đầu vào. Ngoài ra, người dùng có thể đặt tham số "gaps = False", chỉ ra regex "pattern" biểu thị "tokens", thay vì phân tách các khoảng trắng và tìm tất cả các lần xuất hiện phù hợp làm kết quả tokenization.

Tức là dùng cái móc
định nghĩa dữ liệu tự định
nghiêm

- Gán 2 loại:

→ Tokenizer truyền thống:

→ RegexTokenizer (default regex = " ")

Có có hai loại phổ biến
+) Tokenizer cho dữ liệu phổ biến

Big Data in Machine Learning

11

Các công cụ cho NLP



• Ví dụ

```
from pyspark.ml.feature import Tokenizer, RegexTokenizer
from pyspark.sql.functions import col, udf
from pyspark.sql.types import IntegerType

sentenceDataFrame = spark.createDataFrame([
    (0, "Hi I heard about Spark"),
    (1, "I know Spark can work well with NLP"),
    (2, "Logistic,regression,models,are,supervised")
], ["id", "sentence"])
```

sentenceDataFrame.show()

+-----+	id	----- sentence	+-----+
	0	Hi I heard about ...	
	1	I know Spark can ...	
	2	Logistic,regressi...	

Big Data in Machine Learning

12

Các công cụ cho NLP



```
tokenizer = Tokenizer(inputCol="sentence", outputCol="words")  
regexTokenizer = RegexTokenizer(inputCol="sentence", outputCol="words", pattern="\\W")  
# alternatively, pattern="[^w]", gaps=False)  
countTokens = udf(lambda words: len(words), IntegerType())  
tokenized = tokenizer.transform(sentenceDataFrame)  
tokenized.select("sentence", "words")\n    .withColumn("tokens", countTokens(col("words"))).show(truncate=False)
```

sentence	words	tokens
Hi I heard about Spark	[hi, i, heard, about, spark]	5
I know Spark can work well with NLP	[i, know, spark, can, work, well, with, nlp] 8	8
Logistic,regression,models,are,supervised	[logistic, regression, models, are, supervised]	1

phân cách nhau bởi
trong thứ tự phải word

để dán câu trên



Các công cụ cho NLP



```
regexTokenized = regexTokenizer.transform(sentenceDataFrame)\nregexTokenized.select("sentence", "words") \\\n    .withColumn("tokens", countTokens(col("words"))).show(truncate=False)
```

sentence	words	tokens
Hi I heard about Spark	[hi, i, heard, about, spark]	5
I know Spark can work well with NLP	[i, know, spark, can, work, well, with, nlp] 8	8
Logistic,regression,models,are,supervised	[logistic, regression, models, are, supervised] 5	5



Các công cụ cho NLP

□ StopWordsRemover

- Stop word là các từ được lọc ra trước khi xử lý dữ liệu ngôn ngữ tự nhiên (text). Stop word nói chung là những từ phổ biến nhất trong một ngôn ngữ; không có danh sách chung các stop word được sử dụng bởi tất cả các công cụ xử lý ngôn ngữ tự nhiên và thực sự không phải tất cả các công cụ đều sử dụng danh sách đó. Một số công cụ tránh loại bỏ các stop word để hỗ trợ tìm kiếm cụm từ.



Các công cụ cho NLP

- Bất kỳ tập hợp các từ có thể được chọn làm stop word cho một mục đích nhất định. Đối với một số công cụ tìm kiếm, đây là một số từ phổ biến nhất, ít có chức năng, ví dụ như, "is", "at", "which", "on"...
- Tóm lại, stop word là những từ nên được loại bỏ từ input, bởi vì các từ này xuất hiện thường xuyên và không mang nhiều ý nghĩa.



Các công cụ cho NLP

- StopemmeRemover lấy input là một loạt các chuỗi (ví dụ như: output của Tokenizer) và loại bỏ tất cả các stop word khỏi input. Danh sách các stop word được chỉ định bởi tham số stopWords. Các stop word mặc định cho một số ngôn ngữ có thể truy cập được bằng cách gọi `StopemmeRemover.loadDefaultStopemme(language)`, với các ngôn ngữ tùy chọn có sẵn là "danish", "dutch", "english", "finnish", "french", "german", "hungarian", "italian", "norwegian", "portuguese", "russian", "spanish", "swedish" and "turkish". Tham số boolean `caseSensitive` cho biết các kết quả so khớp có phân biệt chữ hoa chữ thường hay không (mặc định là False).

→ h^o phân biệt chữ hoa vs chữ thường

Big Data in Machine Learning

17

Các công cụ cho NLP

• Ví dụ

```
from pyspark.ml.feature import StopWordsRemover

sentenceData = spark.createDataFrame([
    (0, ["I", "go", "to", "school", "by", "bus"]),
    (1, ["Minh", "has", "lots", "of", "pencils"])
], ["id", "raw"])
remover = StopWordsRemover(inputCol="raw", outputCol="filtered")
remover.transform(sentenceData).show(truncate=False)
```

id raw	filtered
[0 [I, go, to, school, by, bus]	[go, school, bus]
[1 [Minh, has, lots, of, pencils]]	[Minh, lots, pencils]

Các công cụ cho NLP

❑ Ngram

- Một n-gram là một chuỗi các nn token (thường là các từ) cho số nn nguyên. Lớp NGram có thể được sử dụng để chuyển đổi các input thành nn-grams.
- NGram lấy input là một loạt các chuỗi (ví dụ như: output của Tokenizer). Tham số n được sử dụng để xác định số lượng term trong mỗi n-gram. Output sẽ bao gồm một chuỗi n-gram trong đó mỗi n-gram được biểu diễn bằng một chuỗi phân tách bằng dấu cách của n từ liên tiếp (n consecutive words). Nếu input chứa ít hơn n chuỗi, không có output nào được tạo ra.



Các công cụ cho NLP

• Ví dụ

```

from pyspark.ml.feature import NGram

wordDataFrame = spark.createDataFrame([
    (0, ["Hi", "I", "heard", "about", "Spark"]),
    (1, ["I", "know", "Spark", "can", "work", "well", "with", "NLP"]),
    (2, ["Logistic", "regression", "models", "are", "supervised"])
], ["id", "words"])

ngram = NGram(n=2, inputCol="words", outputCol="ngrams")

ngramDataFrame = ngram.transform(wordDataFrame)
ngramDataFrame.select("ngrams").show(truncate=False)

+-----+
| ngrams
+-----+
|[Hi I, I heard, heard about, about Spark]
|[I know, know Spark, Spark can, can work, work well, well with, with NLP]
|[Logistic regression, regression models, models are, are supervised]
+-----+
    
```

❑ TF-IDF (Term Frequency - Inverse Document Frequency)

- (Tần suất xuất hiện của từ-nghịch đảo tần suất của văn bản)
- Mặc dù số lần xuất hiện của các từ có thể có ích khi xây dựng các mô hình, các từ xuất hiện nhiều lần có thể làm sai lệch kết quả một cách không mong muốn.



Các công cụ cho NLP

- Để hạn chế những từ phổ biến này từ việc áp đảo mô hình, một hình thức chuẩn hóa có thể được sử dụng. TF-IDF có tác dụng làm giảm giá trị của các từ phổ biến, đồng thời tăng trọng số của các từ không xảy ra trong nhiều tài liệu.



Các công cụ cho NLP

- TF-IDF là trọng số của một từ trong văn bản thu được qua thống kê, nó thể hiện mức độ quan trọng của từ này trong một văn bản, với bản thân văn bản đang xét nằm trong một tập hợp các văn bản.
- IF-IDF thường được sử dụng vì: trong ngôn ngữ luôn có những từ xảy ra thường xuyên với các từ khác.

<https://en.wikipedia.org/wiki/TF-IDF>

Big Data in Machine Learning

23

Các công cụ cho NLP

- TF- Term Frequency : dùng để ước lượng tần xuất xuất hiện của từ trong văn bản. Tuy nhiên với mỗi văn bản thì có độ dài khác nhau, vì thế số lần xuất hiện của từ có thể nhiều hơn . Vì vậy số lần xuất hiện của từ sẽ được chia cho độ dài của văn bản (tổng số từ trong văn bản đó)

Các công cụ cho NLP

- IDF- Inverse Document Frequency (Nghịch đảo tần suất của văn bản), giúp đánh giá tầm quan trọng của một từ . Khi tính toán TF, tất cả các từ được coi như có độ quan trọng bằng nhau. Nhưng một số từ như “is”, “of” và “that” thường xuất hiện rất nhiều lần nhưng độ quan trọng là không cao. Như thế chúng ta cần giảm độ quan trọng của những từ này xuống.

Các công cụ cho NLP

- Công thức:

$$w_{x,y} = tf_{x,y} \times \log \left(\frac{N}{df_x} \right)$$

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

Term x within document y

Các công cụ cho NLP

● Ví dụ

```

sentenceData.show(truncate=False)

+---+-----+
|label|sentence    |
+---+-----+
|0.0 |a b c      |
|0.0 |a b c a    |
|1.0 |a b d d a c c|
+---+-----+


tokenizer = Tokenizer(inputCol="sentence", outputCol="words")
wordsData = tokenizer.transform(sentenceData)
wordsData.show(truncate=False)

+---+-----+-----+
|label|sentence    |words          |
+---+-----+-----+
|0.0 |a b c      |[a, b, c]      |
|0.0 |a b c a    |[a, b, c, a]   |
|1.0 |a b d d a c c|[a, b, d, d, a, c, c]|
+---+-----+-----+

```

Big Data in Machine Learning

27



Các công cụ cho NLP

● Ví dụ

này là
tính TF }
 hashingTF = HashingTF(inputCol="words", outputCol="rawFeatures", numFeatures=10)
 featurizedData = hashingTF.transform(wordsData)
 featurizedData.show(truncate=False)

label	sentence	words	rawFeatures
0.0	a b c	[a, b, c]	[(10,[0,1,2],[1.0,1.0,1.0])]
0.0	a b c a	[a, b, c, a]	[(10,[0,1,2],[2.0,1.0,1.0])]
1.0	a b d d a c c	[a, b, d, d, a, c, c]	[(10,[0,1,2,4],[2.0,1.0,2.0,2.0])]

lưu trữ
tên

đo feature
raw

kết quả

lấy 10 feature có
tần suất (TF) cao nhất

hỗ trợ 4 thí nghiệm và
nó sắp xếp theo alphabet

abc xuất hiện nhiều lần

IDF là 1.38

rescaledData.select("label", "features").show(truncate=False)

label	features
0.0	[(10,[0,1,2],[0.0,0.0,0.0])]
0.0	[(10,[0,1,2],[0.0,0.0,0.0])]
1.0	[(10,[0,1,2,4],[0.0,0.0,0.0,1.3862943611198906])]



28

Các công cụ cho NLP

CountVectorizer

- Lớp CountVectorizer và CountVectorizerModel giúp chuyển đổi một bộ sưu tập văn bản thành một vector đếm. Kết quả khi chuyển đổi biến phân loại thành một vector đếm là one-hot encoded vector. Kích thước của vector sẽ bằng với số loại khác nhau mà chúng ta có.
- Khi không có a-priori dictionary, CountVectorizer có thể được sử dụng như Estimator để trích xuất từ vựng và tạo ra CountVectorizerModel.

Big Data in Machine Learning

29

Các công cụ cho NLP

- Cái này là nhữnghết phải làm, cái TF-IDF đã làm xong, chung là phải làm
corpus = ['The sky is blue and beautiful',
'The king is old and the queen is beautiful',
'Love this beautiful blue sky',
'The beautiful queen and the old king']

['and', 'beautiful', 'blue', 'is', 'king', 'love', 'old', 'queen', 'sky', 'the', 'this']

and beautiful blue is king love old queen sky the this

0	1	1	1	1	0	0	0	0	1	1	0
1	1	1	0	2	1	0	1	1	0	2	0
2	0	1	1	0	0	1	0	0	1	0	1
3	1	1	0	0	1	0	1	1	0	2	0

Các công cụ cho NLP

• Ví dụ

```

from pyspark.ml.feature import CountVectorizer
# Input data: Each row is a bag of words with a ID.
df = spark.createDataFrame([
    (0, "a b c".split(" ")),
    (1, "a b b c a".split(" ")),
    (2, "a b d d a c c".split(" "))
], ["id", "words"])
# fit a CountVectorizerModel from the corpus.
cv = CountVectorizer(inputCol="words", outputCol="features", vocabSize=4, minDF=1)

model = cv.fit(df)
result = model.transform(df)
result.show(truncate=False)

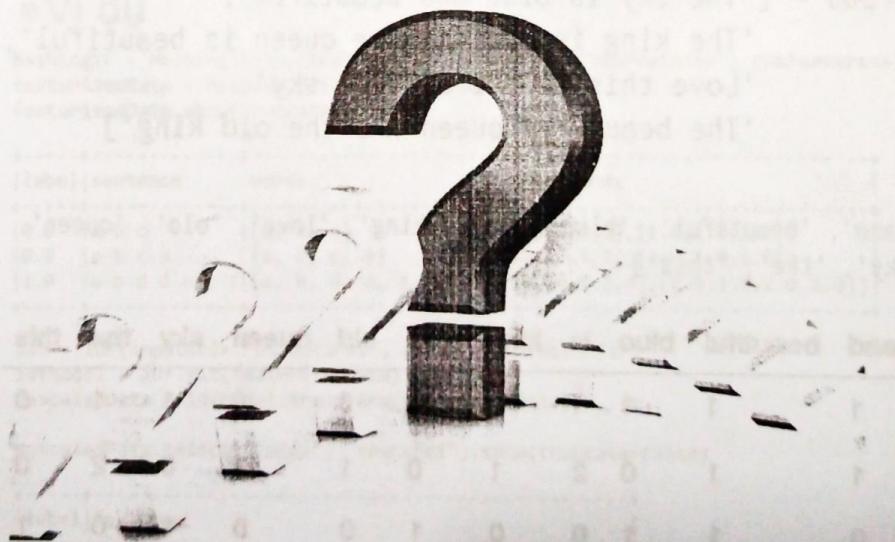
```

xuất hiện if nhái trang bao
nhéu tài liệu

gõ gõ câu num Features

slide 28 gõ chí
lấy 4 câu có fàn số
cao nhái

id words	features
0 [a, b, c]	[(4,[0,1,2],[1.0,1.0,1.0])]
1 [a, b, b, c, a]	[(4,[0,1,2],[2.0,2.0,1.0])]
2 [a, b, d, d, a, c, c]	[(4,[0,1,2,3],[2.0,1.0,2.0,2.0])]





Chapter 11: NLP

Ex1: Ham vs Spam

Requirement: Build a spam filter. Use the various NLP tools and a new classifier, Naive Bayes, to predict if one email is ham or spam.

- Dataset: UCI Repository SMS Spam Detection:

<https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection>
 (<https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection>)

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

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

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

```
In [4]: data = spark.read.csv("smsspamcollection/SMSpamCollection",
                           inferSchema=True,
                           sep='\t')
```

```
In [5]: data = data.withColumnRenamed('_c0','class').withColumnRenamed('_c1','text')
```

```
In [6]: data.show(5)
```

class	text
ham	Go until jurong p...
ham	Ok lar... Joking ...
spam	Free entry in 2 a...
ham	U dun say so earl...
ham	Nah I don't think...

only showing top 5 rows

Clean and Prepare the Data

** Create a new length feature: **

```
In [7]: from pyspark.sql.functions import length
```



```
In [8]: data = data.withColumn('length', length(data['text']))
```

```
In [9]: data.show(5)
```

class	text length
ham Go until jurong p...	111
ham Ok lar... Joking ...	29
spam Free entry in 2 a...	155
ham U dun say so earl...	49
ham Nah I don't think...	61

only showing top 5 rows

```
In [10]: # Pretty Clear Difference  
data.groupby('class').mean().show()
```

class	avg(length)
ham 71.45431945307645	
spam 138.6706827309237	

Feature Transformations

```
In [11]: from pyspark.ml.feature import Tokenizer, StopWordsRemover  
from pyspark.ml.feature import CountVectorizer, IDF, StringIndexer  
tokenizer = Tokenizer(inputCol="text", 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")  
ham_spam_to_num = StringIndexer(inputCol='class', outputCol='label')
```

```
In [12]: from pyspark.ml.feature import VectorAssembler  
from pyspark.ml.linalg import Vector
```

```
In [13]: 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 [14]: from pyspark.ml.classification import NaiveBayes
```



```
In [15]: # Use defaults
nb = NaiveBayes()
```

Pipeline

```
In [16]: from pyspark.ml import Pipeline
```

```
In [17]: data_prep_pipe = Pipeline(stages=[ham_spam_to_num,
                                         tokenizer,
                                         stopremoval,
                                         count_vec,
                                         idf,
                                         clean_up])
```

```
In [18]: cleaner = data_prep_pipe.fit(data)
```

```
In [19]: clean_data = cleaner.transform(data)
```

Training and Evaluation!

```
In [20]: clean_data = clean_data.select(['label','features'])
```

```
In [21]: clean_data.show(10)
```

label	features
0.0	(13424,[7,11,31,6...]
0.0	(13424,[0,24,297,...]
1.0	(13424,[2,13,19,3...]
0.0	(13424,[0,70,80,1...]
0.0	(13424,[36,134,31...]
1.0	(13424,[10,60,139...]
0.0	(13424,[10,53,103...]
0.0	(13424,[125,184,4...]
1.0	(13424,[1,47,118,...]
1.0	(13424,[0,1,13,27...]

only showing top 10 rows

```
In [22]: (training,testing) = clean_data.randomSplit([0.7,0.3])
```

```
In [23]: spam_predictor = nb.fit(training)
```



In [24]: `data.printSchema()`

```
root
 |-- class: string (nullable = true)
 |-- text: string (nullable = true)
 |-- length: integer (nullable = true)
```

In [25]: `test_results = spam_predictor.transform(testing)`

In [26]: `test_results.show(10)`

label	features	rawPrediction	probability	prediction
0	[0,1,2,7,8...]	[-805.18281628272...]	[1.0, 1.5026888754...]	0.
0	[0,1,3,9,1...]	[-571.08015050035...]	[0.99999999999989...]	0.
0	[0,1,7,8,1...]	[-1151.0845440365...]	[1.0, 1.7344215452...]	0.
0	[0,1,9,14,...]	[-560.99061513047...]	[1.0, 1.1693266509...]	0.
0	[0,1,9,14,...]	[-560.99061513047...]	[1.0, 1.1693266509...]	0.
0	[0,1,12,33...]	[-443.53614103103...]	[1.0, 4.2387365649...]	0.
0	[0,1,18,20...]	[-830.74555602901...]	[1.0, 4.1260268479...]	0.
0	[0,1,21,27...]	[-757.45725664880...]	[1.0, 4.2310444011...]	0.
0	[0,1,21,27...]	[-1025.2055750518...]	[1.0, 9.2583979062...]	0.
0	[0,1,24,31...]	[-343.57478257252...]	[1.0, 7.9081642069...]	0.

only showing top 10 rows

In [27]: `test_results.groupBy("label", "prediction").count().show()`

label	prediction	count
1.0	1.0	217
0.0	1.0	152
1.0	0.0	8
0.0	0.0	1323

16/2020

Ex1_NLP_ham_spam - Jupyter Notebook



```
In [28]: from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
In [29]: acc_eval = MulticlassClassificationEvaluator()
acc = acc_eval.evaluate(test_results)
print("Accuracy of model at predicting spam was: {}".format(acc))
```

```
Accuracy of model at predicting spam was: 0.9148755595026191
```

Not bad considering we're using straight math on text data! Try switching out the classification models! Or even try to come up with other engineered features!



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 = 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
sandra tu "Yea...			good	1393545600	
1384719342 [13, 14]		5.0	The product does ...	03 16, 2013	A14VAT5EAX3D95
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 windsreen 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)`

<code> label </code>	<code>features </code>	<code>rawPrediction </code>	<code>probability predictio</code>
0	[0,1,2,3,4...]	[-8037.9167293525...]	2.
0	[0,1,2,3,4...]	[-11977.791289462...]	0.
0	[0,1,2,3,4...]	[-4701.5867096432...]	0.
0	[0,1,2,3,4...]	[-9502.4876633284...]	1.
0	[0,1,2,3,4...]	[-5436.0778379955...]	0.
0	[0,1,2,3,4...]	[-22250.822977014...]	1.
0	[0,1,2,3,4...]	[-12613.007139582...]	1.
0	[0,1,2,3,4...]	[-3678.0900065707...]	0.
0	[0,1,2,3,4...]	[-7685.3563363984...]	1.
0	[0,1,2,3,4...]	[-5670.9474482192...]	0.

only showing top 10 rows

In [32]: `# Create a confusion matrix
test_results.groupBy('label', 'prediction').count().show()`

<code> label </code>	<code>prediction </code>	<code>count </code>
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



```
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 Bayes

```
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???
```