LARGE SCALE TEXT ANALYSIS
WITH APACHE SPARK

Ron Daniel Jr.
Director, Elsevier Labs
2015-03-31
Abstract

Elsevier Labs has developed an internal text analysis system, which runs a variety of standard Natural Language Processing steps over our archive of XML documents. We are now extending that basic system by using Spark and other parts of the Berkeley Data Analytics Stack for additional analyses, including ones at interactive speeds. We expect this to let us prototype new features more easily, and/or look at making current features more efficient. This talk will cover the architecture of the system and our experiences with Spark and other BDAS components.
Agenda

- A Brief Introduction to Spark, BDAS, and Databricks
- Demo: Word Count in Theory and in Practice
- External Libraries
- Demo: Exploring Content with a Concordancer
- Large-Scale Issues and Architectural Changes
- Demo: Converting XML to Plain Text for NLP Processing
- Demo: Abbreviation Handling
- Conclusion
Spark

• Modern successor to Hadoop

• Key Idea: Resilient Distributed Datasets (RDDs)
  • List of objects, partitioned and distributed to multiple processors
  • When possible, RDDs remain memory-resident. Will spill to disk if needed.

• Easier to program than Hadoop's Map-Reduce
  • Can use Scala anonymous functions or Python lambdas to provide functions inline that
    will be executed over all objects in an RDD:
    • someRDD = anRDD.map(lambda obj: func(obj)).map(lambda obj: func2(obj))
  • Many additional operations (sort, filter, reduce, union, distinct, join, pipe, count, ...)
  • Libraries for SQL, Machine Learning, and Graph Processing.
  • Support for Scala, Python, Java, SQL, and R (soon)
For More on Spark:

- Some very good books being published
- https://spark.apache.org/
  - Provided documentation is good
- https://databricks.com/spark/developer-resources
  - Databricks has a lot of additional Spark documentation
Berkeley Data Analytics Stack (BDAS)

- BDAS is:
  - Spark Core +
  - Major libraries built on Spark +
  - Special I/O and Cluster Management
- The libraries read and write RDDs which typically remain in memory between calls
  - Don't have to save intermediate results to disk, transform formats, fire up a new environment to use separate tool
  - Much faster end-to-end!
- BDAS runs on top of well-established standards and/or cluster management systems
  - (e.g. HDFS, S3, HBase, YARN, Mesos)
- Spark Documentation also covers BDAS libraries
Databricks Workspace

• Notebook interface
  • Like Mathematica or IPython Notebooks

• Interactive development
  • Attack small parts of problems and leave an audit trail
  • Caching of results

• Documentation and reports
Word Count - the "Hello World" of Big Data

# Make RDD of lines from text file
rdd = sc.textFile("s3://.../example.txt")

# Make 'tall' RDD with words of each line
wordsRDD = rdd.flatMap(lambda s: s.split(" "))

# Put a count of 1 with each word, then sum the counts for each word
result = wordsRDD.map(lambda w: (w, 1)).reduceByKey(lambda x, y: x + y)
"Demo": Word Count

```python
rdd = sc.textFile("s3://...")
words = rdd.flatMap(lambda s: s.split(" "))
result = words.map(lambda w: (w, 1))
                    .reduceByKey(lambda x, y: x + y)
result.take(20)
result.takeSample(False, 20, 1)

# More useful when sorted
srted_results = result.map(lambda x: (x[1], x[0]))
                    .sortByKey(False)
srted_results.take(20)
```

Note: Indentation is for legibility. Won't run as-is.
Real World Results

• Data: Customer email feedback
• Top 20 words are stop words or punctuation
• Tokenization issues
  • Trailing punctuation: Youngs, Utilization
• Stemming/lemmatization and case folding would be good
  • utilisation, Utilization:
• Random sampling over-emphasizes the tail

[(867015, u'the'),
 (618065, u'to'),
 (524285, u'you'),
 (417616, u'and'),
 (368275, u'of'),
 (364962, u'in'),
 (333093, u'your'),
 (329289, u'for'),
 (275007, u'this'),
 (245704, u'is'),
 (228733, u'I'),
 (212472, u'of'),
 (212406, u'the'),
 (197095, u'Customer'),
 (192925, u'Youngs'),
 (184271, u'table'),
 (184011, u'neuroregeneration?'),
 (183961, u'beach-ridge'),
 (183944, u'-'),
 (175925, u'www.clch.nhs.uk'),
 (175599, u'61.237.229.255'),
 (175364, u'40203'),
 (175357, u'Duisberg-Essen'),
 (1, u'journal/article.')]

(251, u'utilisation'),
 (119, u'Michal'),
 (87, u'Utilization:'),
 (22, u'table,'),
 (6, u'Youngs'),
 (2, u'Minus'),
 (1, u'neuroregeneration?'),
 (1, u'beach-ridge'),
 (1, u'www.clch.nhs.uk'),
 (1, u'61.237.229.255'),
 (1, u'40203'),
 (1, u'Duisberg-Essen'),
 (1, u'journal/article.')
A More Realistic Word Count

- Do wordcount with:
  - Smarter tokenizer
  - Junk token elimination
  - Lowercasing
  - Stopword removal
  - Porter stemmer?
  - Minimum count threshold

- Conclusions:
  - A custom stopword list is important
  - Wordcount not the right tool for determining problems
import nltk
from nltk.tokenize import TreebankWordTokenizer
from nltk.stem.porter import *

stemmer = PorterStemmer()
words = sentsRDD.flatMap(lambda s:
    TreebankWordTokenizer().tokenize(s.lower()))
stems = words.map(lambda w: stemmer.stem(w))
stopped = stems.filter(lambda w: w not in stopwords)
Useful Libraries

- NLTK, Numpy, scikit.learn, and other Python libraries
  - NLTK easy to install in Databricks, just give name of PyPI package
  - Numpy & scikit.learn already installed.
- Stanford CoreNLP and other Java libraries in .jar files
  - Provides more advanced capabilities than NLTK (dependencies, constituents, co-ref)
  - More difficult to install and use
    - Must cut unneeded model files from .jar
    - Must work around non-serializability of StanfordCoreNLP class.
- LXML (Python) or Spark-XML-Utils+Saxon(Scala calling Java)
  - Can read XML content as strings in RDDs
  - Very efficient for I/O
  - XPath and other tools to extract data from the XML for new RDDs
  - *Plug* Spark-XML-Utils developed by coworker, available from GitHub
Better Exploration with a Concordancer

```python
concordancer(KVSchemaRDD, r"trouble with ([^\./,;:\-\s]+)", 20, 70, 25)
```
import re

def makeTableRows(pii, str, pattern, pre, post):
    iterator = pattern.finditer(str)
    for match in iterator:
        (lo, hi) = (match.span())
        start = max(0, lo-pre)
        end = min(len(str), hi+post)
        tmpStr1 = "<tr><td>"+pii+"</td><td style='white-space: nowrap'>" \ 
        + str[start:lo].replace("&", "&amp;").replace("","&lt;") \ 
        + "<span style='color:red'>" + str[lo:hi].replace("&", "&amp;").replace("","&lt;") \ 
        + "</span>" + str[hi:end].replace("&", "&amp;").replace("","&lt;") + "</td></tr>"
        yield tmpStr1

def concordancer(rdd, regex, pre, post, nrows):
    pattern = re.compile(regex)
    rdd1 = rdd.flatMap(lambda row: makeTableRows(row.pii, row.text, pattern, pre, post))
    lis1 = rdd1.take(nrows)
    tmpStr = "\n".join(lis1)
    displayHTML("<table border='1' style='font-family: monospace'">" + tmpStr + "</table>")
Large-Scale Issues

• Avoid shuffles!
  • Hash partition the wordcount example for faster reduceByKey

• I/O: Many small files gives very poor I/O utilization
  • 45 min for 13M text files, 6 min for Hadoop sequence file

• Weekly 'cron' job to update the Hadoop Sequence Files
  • Immutability of RDDs makes HSF maintenance harder
  • Splittable compression not so important if HSFs heavily partitioned.
  • Easy to work with subsets - numeric suffixes handled with wildcards

• Unreliable writes to S3
  • Save a copy to HDFS first, then write to S3. Don't have to re-compute from scratch if write fails
Content Analysis Toolbench ver. 2 & 3

Architecture of CAT 2

EC2
- Initial NLP Processing
- Gaz Parse ...
- Annot Search Indexing

S3
- SD XML
- Annot Files
- Indices

ScienceDirect Production Workflow

Architecture of CAT 3

EC2
- Spark
- Spark Synch
- INP++
- DBricks Index ...

S3
- SD XML
- XML Snapshot & Updates (.hsf)
- Annots & Updates (.par)
- Annot Search Indices (.par)

ScienceDirect Production Workflow

Labs
Production
Labs
Production
Effect of concurrent intratracheal lipopolysaccharide and human serum albumin challenge on primary and secondary antibody responses in poultry

Abstract
Activation of the innate immune system by pathogen-associated molecular patterns (PAMPs) may direct specific immune responses and as a consequence probably significantly affect vaccination. Previously, we described modulation of specific antibody responses to systemically administered model antigens by intravenously (i.v.) as well as intratracheally (i.t.) administered PAMP such as the endotoxin lipopolysaccharide (LPS). In this study effects of various doses of i.t.-administered LPS on primary and secondary specific total and individual isotype (IgM, IgG and IgA)-specific antibody responses of chickens simultaneously i.t. challenged with various doses of human serum albumin (HuSA) were determined.
i.t.-administered LPS enhanced primary and secondary HuSA-specific total and isotype-specific antibody titers depending on the dose of LPS and the dose of simultaneously administered HuSA. i.t.-administered HuSA enhanced primary and secondary total antibody responses to ‘environmental’ LPS as shown in birds receiving the zero i.t. LPS treatment, which also depended on dose of HuSA. HuSA administration decreased antibody responses to high doses of LPS. Body weight gain as a measurement of an acute phase cachectin response to LPS was affected by a HuSA by LPS interaction, indicating that simultaneously administered higher doses of HuSA decreased LPS-induced cachectin responses of the birds. Our results suggest a complex interaction of innate and specific immune system activating airborne antigens, which may have significant consequences for vaccination and husbandry management procedures.
"Demo": Abbreviation Handling

- Common patterns:
  - Three Letter Acronyms (TLA).
  - TLA (Three Letter Acronym).
- Schwartz-Hearst Algorithm:

Three Letter Acronym (TLA)

Look for T Look for L Look for A

Plot the most ambiguous abbreviations
Count how many different distributions we have for each abbreviation, sort by freq, and plot.

- A unique abbreviation strings appear only once in the abbrevCountByFreq list, so we can pick out A the code and do a workaround to see how many different definitions there are:

  ```
  abbrevCountByFreq = abbrevCountByFreq.countBy(entry => (entry.key[0], entry.key[1], entry.key[2]))
  abbrevCountByFreq.value = abbreviated(key => abbrevCountByFreq.reduceByKey((a, b) => a + b))
  ```

Output:

```json
```

Command took 0.068s

- A new draw the plot:

  ```
  abbrevCountByFreq = abbrevCountByFreq.countBy(entry => (entry.key[0], entry.key[1], entry.key[2]))
  abbrevCountByFreq.value = abbreviated(key => abbrevCountByFreq.reduceByKey((a, b) => a + b))
  abbrevCountByFreq.value = abbrevCountByFreq.value.sortByKey((k, v) => -v)
  ```

  ```sql
  select
    abbrev by freq
  from abbrevCountsTable
  order by abbrev desc
  ```
Conclusion

• Combination of Spark, Notebook UI, and library import provides a very productive environment for text analysis.

• Data quality depends on detecting errors. Interactive environment, and the ability to use good supporting tools like NLTK's tokenizer instead of toy solutions like split(' ') for tokenization, leads to improved data quality.

• Very easy to compute a variety of analyses and keep them for future use.

• Combining the analyses to achieve a goal is the real trick

• Combination is slightly simplified by best practice of using SchemaRDDs and common keys