A Distributed Sentiment Analysis Development Environment

Christopher Burdorf
NBCUniversal
5750 Wilshire Blvd
Los Angeles, CA, USA
Christopher.Burdorf@nbcuni.com

Abstract

This document describes a work-in-progress development of a distributed sentiment analysis system being developed for industrial usage. This project seeks to leverage advances in open source development in natural language processing and understanding, distributed databases, and distributed processing frameworks, as much as possible to advance the development process and to utilize the most recent technology. This paper describes the tools, technologies, and data sources we've utilized as well as issues we've encountered in their integration.

1 Introduction

Sentiment analysis of social and other online media shows promise for marketing and product development in determining the preferences of customers of media-related product lines (eg. film, television, and theme parks). Given the dynamic behavior of media-related customer sentiments, it would be highly advantageous to track customer sentiments as they are posted onto social media, forums, and message boards to track current changes in customer sentiments.

The goal of this project is to develop relatively low cost, scalable, and efficient software tools for industrial usage of sentiment analysis. In this paper, we begin with a review of candidate tools and determine which ones to use. Then we describe our initial implementation followed by a description of our selection of an appropriate social media source. We then analyze the results and discuss revision of the system based on initial performance and problems. Finally, we present future directions and conclusions.

2 Review and Selection of Tools

We have determined we need four components for our development environment. 1) A distributed database system for fast retrieval and scalability of large data sources. Distributed databases also provide high availability, where if a node goes down, the system will still process transactions using the remaining working nodes it has available. 2) A programming language and distributed computing framework which facilitates scalability of concurrent sentiment analysis on different messages. 3) A natural language processing system to process the social media messages and allow us to develop rules to customize the system to perform sentiment analysis using our institutional knowledge. 4) A social media source which will allow us to sample a large audience and determine their sentiments related to our products.

Given the extensive open source libraries developed in the Java programming language, we have decided to utilize a language that can access those libraries directly. Also, given the support for distributed processing, code reduction, and ability to use Java libraries, we have decided to investigate the Scala programming language. Akka(Typesafe, 2015) and Spark(Apache Software Foundation, 2015) are distributed processing libraries/frameworks which are written in Scala and provide interfaces to Scala which provide high-level support for distributed processing.

Scala is a hybrid functional/object-oriented programming language. It runs within the Java virtual machine (JVM) and can therefore utilize all Java libraries which are extensive.

The improvements of Scala over Java are well-documented (Eder, 2014). One of the biggest advantages is the reduced amount of code that needs to be written using Scala. Also, the support for distributed processing via functional programming is well-established(Milewski, 2014).

Hadoop HDFS and HBase (Apache Software Foundation, 2010) were the initial systems that we have worked with to store and retrieve social media text and related metadata and sentiment scores. HDFS/HBase scales well and is mature and stable. Also, the importation of data into HDFS/HBase can be done from a CVS file using a *map/reduce* utility *importTsv*. Recently, however, we have been migrating to Cassandra, because it has built-in high availability which we think will be critical as we opt for storage on large clusters of servers which will increase the probability of a single node failure.

We reviewed NLP systems and narrowed our search to NLTK(Steven Bird et. al., 2009) and Stanford CoreNLP(Manning et al., 2014). NLTK sentiment analysis uses Naive Bayes classifiers which is a well known and effective machine learning technique. However, two things that discouraged us from using NLTK were 1) a desire to utilize deep learning technologies which have been shown to be a big advancement for machine learning, and 2) distributed processing in the native language of NLTK, Python, is not as well supported as in Java and/or Scala. Instead, we chose the Stanford CoreNLP system, because it's sentiment analysis system uses recurrent neural networks which is a new deep learning technique, it recently added multi-threaded training, and it is actively maintained. Rather than write our own NLP system or purchase one, we preferred the rapid advancement we saw in the Stanford CoreNLP system, and the ability to access the code base as desired.

We then determined that we needed distributed processing for our industrial social media sentiment analysis system, because we expected a large number of users and messages, so if we wanted to process them in a reasonable amount of time, we needed to distribute the workload across a large cluster of machines. Both the Spark and Akka frameworks for distributed processing are native to Scala. Spark utilizes Akka to support seamless distributed collections (RDDs) automatically, therefore eliminating the need of the application developer to implement complex message passing or other distributed mechanisms to make use of the power of distributed computing.

Since we have used the Spark cluster computing framework on other projects (Burdorf, 2015), we were motivated to use it in association with the sentiment analysis annotator in the Stanford Core NLP system. However our initial experiments showed some interface issues. For example, the Stanford Core NLP Java class isn't serializable, so it doesn't work well within the Spark ecosystem.

With the Spark experience as a backdrop, we began investigating the use of Akka toolkit to handle concurrency and distributed processing. Spark is built on top of the Akka framework, so it clearly has the capabilities of providing the concurrency and performance that we need for this project.

Akka is an Actors-based distributed processing toolkit that is based on Carl Hewitt's Actor-based distributed computing model (Hewitt et al., 1973). An actor is a fundamental processing unit which supports processing, storage, and communication. Actors can run concurrently on multi-core systems and distributed over a cluster of multi-core systems.

3 Implementation

We began implementing Akka actors on a 64 core system. We implemented two Actor subclasses: one for the sentiment look ups from the NLP system to look up sentiment analysis on input messages and a second to write the results to a database.

The system creates 16 NLP Actor instances so that they can be executed in parallel on multiple cores. We choose 16 actors, because we need to share the server with other users and we did not want to overload it.

The database reads of social media messages are buffered into primary memory and then get sent as messages to the NLP Actors to process sentiment values which then get sent to a single writer actor to serialize writes to the database. Figure 1 illustrates the Actor classes that we developed and the direc-

tion of the messages passed between them. The input class initializes the database, and then periodically reads chunks of messages, filters out spam, and passes each individual non-spam message off to the NLP actor instances for processing. We found that we got better performance by inputting bunches of messages at a time (eg 100) and then sending each individual message to an NLP object rather than input a single message at a time. The NLP actor instances takes each message, and determines a sentiment value and then passes it to the writer instance which writes the sentiment value to the database for the social media message. We utilize

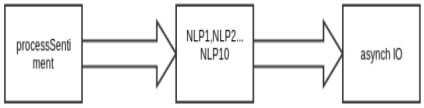

Figure 1: Actor Implementation

a round robin dispatcher to handle message passing between the buffered reads and the NLP actors instances. It ensured that the message processing was evenly distributed amongst the NLP actors concurrently. When there is no more data to be read from the database. When the end of the input data was reached, a poison pill message was sent to the NLP Actors telling them to finish and shutdown.

Using this Akka mechanism resulted in a six-fold performance increase over using Java Parallel Streams (Oracle Inc., 2015). Thus, we expect to continue our usage of Akka for distributing the sentiment analysis workload.

3.1 Training Mechanism

Prior to the usage of the Stanford CoreNLP system sentiment analysis annotator, one must develop a training set to teach the system what phrases to use to indicate levels of sentiment with social media mined messages. The system requires developing a data set of binary trees which have values associated with each element. Here is an example:

```
(0 (2 It) (2 (2 (2 's) (0 (2 like)
   (1 (2 watching) (0 (1 (2 a)
   (0 nightmare)) (2 (2 made)
   (2 flesh)))))) (2 .)))
```

In the above example, a value between 0 and 4 is set for each word, phrase, and sentence to train the system. the number 0 indicates the most negative sentiment and 4 indicates the most positive sentiment. The process of developing the training set involves analyzing messages and phrases, setting appropriate values, executing the training system, and then using it in the sentiment analysis system to produce sentiment values for each message. Sometimes it takes several iterations to get the values to provide the desired result.

There is then a separate training mechanism which reads the training set and generates an internal database using recurrent neural networks (Socher et al., 2014) which is then accessed to generate sentiment values for messages passed to the annotator.

4 Social Media Sources

To complete testing of our initial distributed sentiment analysis system we needed to gain access to rich collections of social media messages. Social media is a relatively new and dynamic technology. Growth in the area has been enormous, but access to social media messages has been inconsistent at best. Facebook used to allowed access to raw message data via public sources. However, due to privacy concerns among users, Facebook shut off public access to raw data and replaced it with Topic Data(Facebook, 2015) which is pre-analyzed. It does not allow us to do our own sentiment analysis but requires that we trust Facebook to do it for us. Unfortunately this does not meet our needs, because we would like to customize the analysis based upon our proprietary institutional knowledge.

We have also accessed the Twitter Firehose source(Twitter, 2015) to get raw messages posted to Twitter. This data is of the type we want, but it is very costly to access. There is a free public API, but it only retrieves a fraction of the messages posted to Twitter. Also, from our experience using Twitter Firehose messages, roughly 80 percent of the messages we get on topics of interest are spam and therefore make it not worth the cost.

The best source we have found is Disqus. Disqus is a blog comment hosting service for web sites.

It's most often used to manage, store, and display comments on news sites, blogs, and message boards. Disqus raw data is accessable and of good quality. The amount of spam in Disqus messages is relatively small compared to what we have dealt with from Twitter.

5 Results

Our goal is to produce a system that can process large amounts of social media messages and store sentiments ratings to databases that can help us better understand the likes and dislikes of customers towards our products. Since this is a work in progress, we have not yet fully realized that goal. However, we have been able to pinpoint from our results, the main focus areas that need improvement. The greatest problem we have encountered on this project is the development of training sets, because it is a very time consuming process that involves a certain amount of trial and error. The number of possible phrases that can be used to measure sentiment is very large, and the Stanford CoreNLP system requires that we provide specific rules that cover every variation of phrase and spelling that we could find on social media to perform sentiment analysis.

Upon using the training mechanism, we quickly determined that a major bottleneck to the development of the sentiment analysis tree bank was the time required to train the system. On average training runs were taking 8 hours. As we expanded the training set, it took longer. Also, the training mechanism computes a converging time series to determine completion. This sort of algorithm does not lend itself to parallel or distributed computation. However, upon investigating the lower level portions of the code and communicating with John Bauer(Bauer, 2015), who is one of the implementors, we were able to determine that computation done on inner loops of the convergence model were indeed parallelizable. Upon attempting several paradigms including Actors, and Parallel collections, we found that the level of parallelism was too fine grain to enhance performance, because overhead costs were too high. Upon further communications with John Bauer, we were able to determine a way to chunk the computations into single threads to reduce the overhead costs. We were able to then get sub linear performance improvements using this mechanism. Subsequently, we found out that Bauer had concurrently added his own implementation and our testing found his implementation to be 10 percent faster than ours, so we choose to go with his implementation which is now part of the Stanford CoreNLP distribution.

6 Future Work

We seek to develop a methodology for determining the optimum training set for sentiment analysis. To facilitate the development of the training set, we look at results of the outcomes we would like to see predicted (eg. the success of certain products) and any potential predictors that could help us determine their successes based on social media sentiments. The goal would be to find an optimal training set that will allow us to confidently predict success based on sentiments mined from social media. Thus, we are developing statistically significant samplings of social media messages and applying different training sets to them and comparing the results. The training sets range from general sets, content specific sets, and even completely untrained systems. Over the course of the project, we want to find the approach to data set building which provides the greatest overall success relative to the prediction of product success and/or failure.

As mentioned previously, spam filtering is an important aspect of what we need to do in our system. We have found significant quantities of spam in social media sources. Some sources are so overloaded with spam in social media messages, that significant resources are required to filter them. Spam is also dynamic in that spammers are constantly finding new ways to override filters and flood social media with whatever they seek to propagate. Our current spam filtering system is brute force. It just accesses a file of phrases to look for those to filter out and then it does pattern matching on each method to see if any portion of it matches anything in the spam file list. We have investigated a much better method using deep learning from H20 using their Sparkling Water system(Mahlolava, 2015). The H20 system allows a very similar format to the format we are already using. For example, it allows users to create a data file containing phrases and sentences and label-

ing them as *ham* or *spam*. The system can then be trained to learn to look for such phrases in the text that it is applied to. It will then produce results that from our testing is highly accurate and executes in a fraction of the time that is required by our brute force method.

Another important direction for our work is to increase the level of distribution of our Akka-based system. Currently, it distributes to multiple cores on a single machine. While this does provide significant performance improvements, we would also like to distribute across nodes to leverage multiple cores on a cluster of servers. Distribution over multiple servers is relatively seamless using the Spark clustering framework, but with Akka, it requires manually setting up listeners on each server that can accept actor messages and process them while sending out messages to actors that will also be listening on other servers. The scalability of this approach will allow us to fully exploit a very large cluster, because using distributed databases and distributed Akka actors will allow us to perform sentiment analysis on each message concurrently.

Finally, we seek to fully migrate from Hadoop to Cassandra. The advantage we seek from Cassandra is built-in high availability which will allow other servers to take over automatically is one is down.

7 Conclusion

We are in the process of developing a distributed and scalable sentiment analysis system for industrial use. We are seeking to do so by leveraging open source libraries as much as possible to reduce development time, and to apply the latest techniques available. We have shown through experimentation with our system that performance improvements using distributed processing is achievable. Also, high availability through the usage of distributed database systems like Cassandra provides additional benefits that apply to our domain.

Acknowledgements

Thanks to David McArthur for reviewing this document and providing numerous helpful suggestions.

References

Apache Software Foundation. 2010. Hadoop. https://hadoop.apache.org.

Apache Software Foundation. 2015. Spark Cluster Computing Framework. http://spark.apache.org.

John Bauer. 2015. forum communication: java-nlp-support@lists.stanford.edu.

Christopher Burdorf. 2015. Use of Spark MLlib for Predicting the Offlining of Digital Media. Spark Summit.

Lukas Eder. 2014. The 10 Most Annoying Things About Java after using Scala. https://jaxenter.com/the-10-most-annoying-things-about-java-after-usi html.

Facebook. 2015. Topic Data: Learn What Matters to Your Audience. https://www.facebook.com/business/news/topic-data.

Carl Hewitt, Peter Bishop, and Richard Steiger. 1973. A universal modular ACTOR formalism for Artificial Intelligence. In *IJCAI'73 Proceedings of the 3rd international joint conference on Artificial intelligence*, pages 235–245, San Francisco. Morgan Kaufmann Publishers.

M. Mahlolava. 2015. Hands on Sparkling Water. https://github.com/h2oai/h2o-world-2015-training/blob/master/tutorials/sparkling-water/SparklingWater.pdf.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

Bartosz Milewski. 2014. The Downfall of Imperative Programming. https://www.fpcomplete.com/blog/2012/04/the-downfall-of-imperative-programming.

Oracle Inc. 2015. Java 8 streams. https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html.

Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2014. Recursive deep models for semantic compositionality over a sentiment treebank.

Steven Bird et. al. 2009. Natural language processing with python. Natural Language Processing with Python.

Twitter. 2015. Twitter firehose. https://dev.twitter.com/streaming/firehose.

Typesafe. 2015. Akka Toolkit. http://Akka.io.