Speech Analytics Software Library

Kyle White
Columbia University
kjw2146@columbia.edu

Abstract—**Performance of speech recognition systems is strongly dependent on the process of adapting general tools to a specific task. The adaptation process generally utilizes audio and transcription data from the domain of interest to bias the recognizer for strong performance in the domain of interest. The amount of training data modern acoustic and language models can utilize to increase expected performance before diminishing returns strongly kicks in has continued to grow. This growth in training data causing additional demands on the speech system builder to store this data and use it to train statistical models. The speech analytics software library is proposed to aid the speech system builder in managing this data explosion in a distributed environment.**

Keywords- distributed; n-gram; language model; speech recognition

I. INTRODUCTION

Performance of speech recognition systems is strongly dependent on the process of adapting general tools to a task. The adaptation process generally utilizes audio and transcription data from the domain of interest to bias the recognizer for strong performance in that domain. The amount of training data modern acoustic and language models can utilize to increase expected performance before diminishing returns strongly kicks in has continued to grow. This growth in training data causing additional demands on the speech system builder to store this data and use it to train statistical models. The speech analytics software library is proposed to aid the speech system builder in managing this data explosion in a distributed environment.

An example of statistical models making additional performance gain available with the use of large data sets are those backed by neural network classifiers. The classifiers can create complex classification boundaries and thus continue to constructively modify those boundaries beyond when more rigid classifiers see diminished returns. Even historically common methods of language modeling such as n-gram language models can benefit in terms of coverage and thus performance from larger training data sets. The first tool built to populate the speech analytics software library is a tool to build n-gram language models in a distributed environment made capable by Apache Hadoop [5].

II. RELATED WORKS

Existing toolkits have been integrating support for training, models, such as the ability to train a deep neural network on audio training data, across more than a single CPU or even on multiple GPUs [1]. Additionally, a significant amount of analysis and machine learning capabilities for distributed environments have been implemented and made available [2]. The combination of speech recognition systems moving towards distributed training and the availability of tool for the distributed environment make an exploration into this area a tempting proposition due to a known need and ability to leverage exiting tools to make significant progress.

III. SYSTEM OVERVIEW

The distributed language model generation tool from the speech analytics library works in two phases. The first phase accepts transcript documents stored in a HDFS, and using the Apache Lucene library for text analysis and tokenization, creates n-gram count documents from the transcriptions. The second phase accepts the n-gram count documents as input and utilizes the berkeleylm software library [3] to generate an n-gram language model representation with Kneser-Ney probability smoothing, as implemented in the SRI language modeling toolkit [4].

Usage of the tool and creation of its inputs are covered in a user manual included in the repository. The language model generation tool is currently callable as an Apache Hadoop job via the command line, or can be called from inside a custom Hadoop job given it is open source. All inputs and outputs are assumed to be in a HDFS. The input transcripts are required to be tagged with sentence beginning and end symbols, if they are needed, prior to being sent as input to the language model generation tool. The transcripts are also assumed to contain one sentence per line. The output of the language model generation tool is currently just an ARPA format representation of a language model. The file is output into a HDFS.

IV. METHOD

The language model generation tool is largely the result of the assembly of existing tools, along with the custom Java classes required to utilize a subset of these tools in a distributed environment enabled by Apache Hadoop. Phase I and phase II processing are serial in time.

Phase I processing is centered on the usage of the MapReduce paradigm implementation in Hadoop and

Lucene core library text analysis capabilities. Custom Java classes are created to enable Lucene text analysis across multiple data nodes in order to produce n-gram counts from each sentence in the input transcriptions. Custom Mapper, Reducer, Client, and Partitioner classes comprise the portion of the tool written anew and not assembled from existing libraries. Lucene text analysis is performed on each data node with a selected sequence of data filters [6]: WhitespaceTokenizer, LowerCaseFilter, ShingleFilter. This sequence of filters is suited for transcript data that should not be modified by aggressive text normalization prior to the creation of n-grams. The filter sequence breaks on whitespace, performs letter case normalization, and then performs word based n-gram creation on the input transcript utterance. The responsibility of the Reducer step depends on the use of the Partitioner class. The Partitioner class can either be utilized to cause N Reducers to be used, where N is the order of the largest n-gram created from the input text analysis, or be utilized by a single Reducer step to direct the output n-gram counts to separate files in a HDFS. The creation of different n-gram count documents is done to enable loading of the information into a utility to perform n-gram probability estimation; the ordering of input n-gram counts by n-gram order is a common requirement for the input to such tools [3].

Phase II processing is centered on the usage of berkeleylm to take the input from multiple n-gram count documents, where each document contains the n-gram counts for n-grams of a certain order and none others, and output an ARPA format representation of an n-gram language model. The input n-gram counts must be ordered, and the output ARPA format file is output to a HDFS. Berkeleylm was the chosen language model estimation tool because of its goal in handling large amounts of data. It is planned that future steps in utilizing the language models created by this tool may use berkeleylm to efficiently store and access them during runtime operations.

V. Software Package Description

The software package the language model generation tool lives in is named ‘ripley’, and it is publicly available [7]. The user is referred to the user guide and project notes documents for a more detailed description of the software package, its history, home, current status, and a graphical tour of its use.

VI. Experiment Results

The intention of the tool is to allow distributed generation of n-gram language model documents. Verification to this point has been for correctness of results. Further tests for speedup have yet to been tested because the data set of interest to the developer during the creation of this tool was not of a size to reap large rewards from distributed computing.

VII. Conclusion

The language model generation tool successfully completes its goal of performing language model training in a distributed environment. Further work and tuning is needed to optimize the tool for speed to reap the rewards of parallelizing the n-gram counting phase of the training process, and parallelizing the Reducer stages for each n-gram order.

Appendix

A number of documents are included in the repository: a user guide, a current shortfalls document regarding the library, and a document providing project notes and a graphical tour of the project.

References

[1] Povey, Daniel, et al. "The Kaldi speech recognition toolkit." *Proc. ASRU*. 2011.

[2] Apache. (2014). Apache Mahout (Version 0.9) [Software]. Available from http://mahout.apache.org/.

[3] Adam Pauls. (2014). Berkeleylm (Version 1.1.5) [Software]. Available from https://code.google.com/p/berkeleylm/.

[4] SRI International. (2014). SRI Language Modeling Toolkit (Version 1.7.1) [Software]. Available from http://www.speech.sri.com/projects/srilm/.

[5] Apache. (2014). Apache Hadoop (Version 2.6.0) [Software]. Available from http://hadoop.apache.org/.

[6] Apache. (2014). Apache Lucene (Version 4.10.2) [Software]. Available from http://lucene.apache.org/.

[7] Kyle White. (2014). Ripley (Version 0.0.1) [Software]. Available from https://github.com/kjw03/ripley.