INDIANA UNIVERSITY
BLOOMINGTON

Manual for BEAR
Big Data Ensemble
of Adaptations for Regression
Version 1.0

Vahid Jalali
David Leake

October 5, 2015

Abstract

BEAR is a case-based regression learner tailored for big data processing. It works by applying an ensemble of adaptations for adjusting instance-based estimations, where adaptations are generated automatically by comparing pairs of cases in the local neighborhood of the input query. BEAR builds on previous work on EAR, which has been shown to improve regression accuracy but increase computational cost. BEAR speeds up source case retrieval and adaptation generation by using locality sensitive hashing for case retrieval, making the EAR approach feasible for case bases with several millions of cases on a relatively small cluster. This document reviews BEAR's foundation and algorithm and provides step by step instructions for building and using BEAR in practice.

1 Introduction

This document provides a brief overview of BEAR (Big data Ensembles of Adaptations for Regression) [1], a method that applies big data techniques to scale up automatic adaptation knowledge generation and application with EAR4 [2], a lazy learning method that works by applying an ensemble of adaptations for regression, and provides step by step instructions for building and using the source code of the BEAR method.

BEAR builds on two methods, EAR4 and locality sensitive hashing (LSH) [3]. BEAR uses LSH for finding nearest neighbors of a case. Compared to standard kNN, LSH has the advantage that it does not require examining the distance of all cases to a case in order to find its nearest neighbors, in contrast, it only requires applying a hashing method (or a family of hashing methods) to each case. LSH sacrifices accuracy (to a limited extent) for speed and can be useful in two scenarios. First, even on standard computational resources, LSH enables finding nearest neighbors with less computation. Second, when cluster resources and parallel computing (e.g. the MapReduce model) are available, the computational gain will become even more significant compared to standard kNN because cases can be distributed among different nodes based on their hash keys, and cases in each node can be processed independent of cases in other nodes.

BEAR's source code uses the implementation of EAR4's Weka plugin [4] and Apache DataFu [5], to support case-based regression and locality sensitive hashing respectively. The corresponding class from Apache DataFu used in BEAR's implementation is $L2PStableHash$, with a 2-stable distribution and default parameter settings.

2 Background

2.1 EAR4

EAR4 is a case-based regression method that works by applying ensemble of adaptions for adjusting the instance-based predictions. The generic process of case-based regression is illustrated in Fig. 1. Given a new problem, case-based regression generates a solution by retrieving a set of cases for similar problems, adjusting the solution values of the retrieved cases (which we will call source cases), and then combining the adjusted values to generate the final predicted value. A brief synopsis of case-based regression and EAR4 is available in EAR4's manual [4], and additional material is available in research publications on EAR4 [2].

2.2 Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) is a hashing method for decreasing the retrieval time of similar cases in d-dimensional Euclidean space. LSH achieves this goal by applying a set of hashing functions for which the probability of collision for similar cases is higher. Therefore, in contrast to nearest neighbor search, LSH does not require comparing a case with other cases to find its nearest neighbors, which results in lower time complexity of the retrieval process.

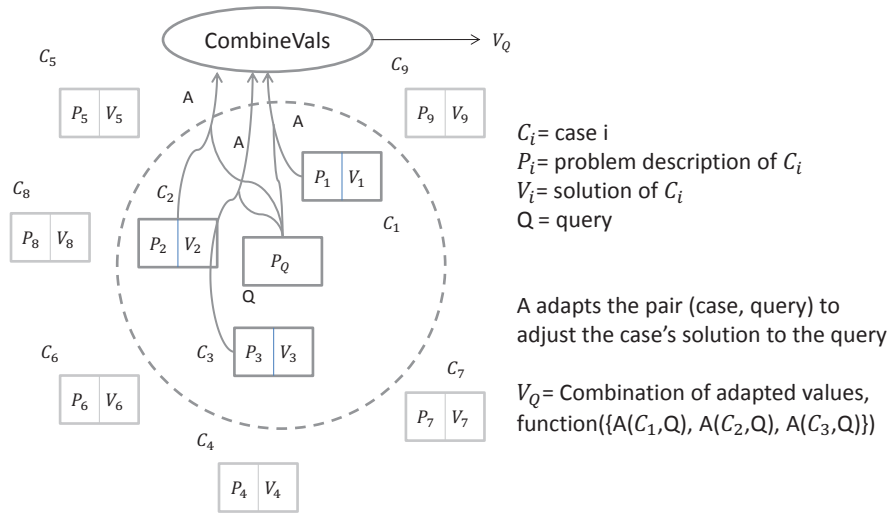

Figure 1: Illustration of the generic case-based regression process [6]

BEAR applies an implementation of LSH for the MapReduce framework, which enables parallel execution of tasks by distributing them among different nodes in a cluster. Its LSH method uses *Apache hadoop*, a popular open source implementation of MapReduce among both industrial and academic communities.

3 A Quick Sketch of BEAR's Approach

BEAR is an implementation of case-based regression for big data. It uses LSH to retrieve nearest neighbors of the input query and adjusts the values of the nearest neighbors by applying an adaptation of ensemble rules. Ensemble rules and nearest neighbors are both extracted from the local neighborhood of the input query (determined by LSH). Adaptation rules are generated by applying the *Case Difference Heuristic* [7] to generate rules based on comparing pairs of cases.

The overall process flow of BEAR is depicted in Fig. 2. The query is represented by a square and cases in the case base are represented by circles. As illustrated, the query and cases are distributed among different reducers by applying LSH. Next, based on the rules from the local neighborhood of the query, a set of adaptation is generated which can be used to build the final estimation.

For a research paper with details about BEAR's underlying process, a discussion of related work, and comparative evaluations of BEAR's performance, we refer the reader to Jalali and Leake [1].

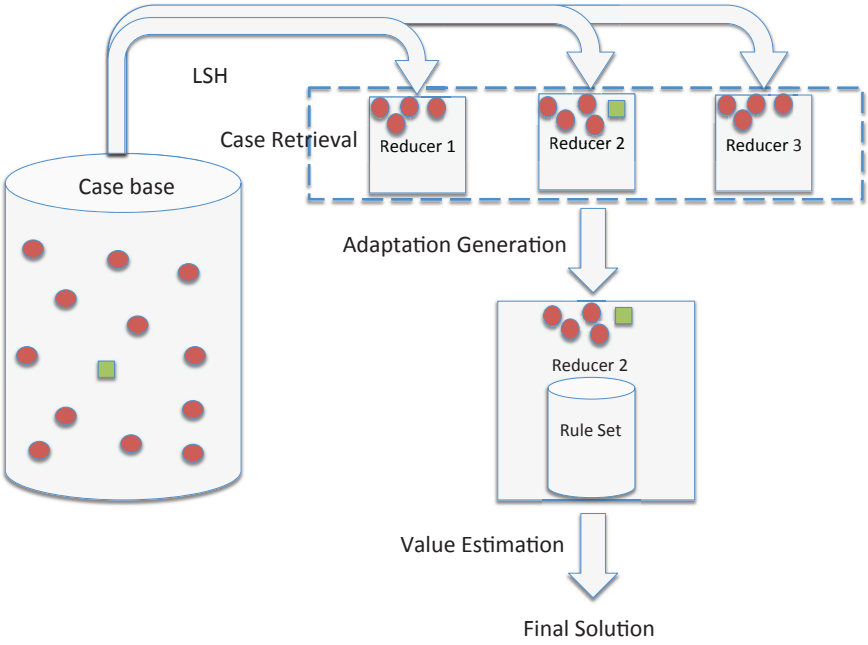

Figure 2: BEAR's process flow

4 How to Build and Use BEAR

4.1 Downloading Resources and Building EAR4 UDF

BEAR can be downloaded from BEAR package.[1] The current version of BEAR applies to domains with numeric input features and target values.

After uncompressing the folder you should be able to see the file structure shown in Fig. 3.

1. The *data* folder contains sample training data (the initial case base) (mpg1.data) and test data (mpg2.data) for the MPG domain from the UCI machine learning repository [8].

2. The *pigs* folder contains the kNN and BEAR implementations in the PigLatin language for Apache Pig.

3. The file *shell* contains the commands for compiling the Java UDF and running the pig scripts.

4. The *src* folder contains the implementation of EAR4's Weka plugin as a Pig UDF in Java.

5. The *jars* folder contains the required jars for running the pig script and compiling EAR4's source code. I have also put

[1] http://sourceforge.net/projects/bear4/files/bear.zip/download

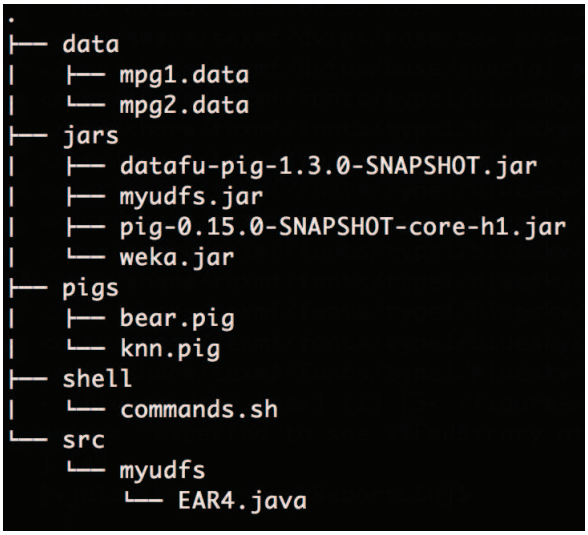

Figure 3: BEAR's package file structure

weka.jar, datafu-pig-1.3.0-SNAPSHOT.jar, pig-0.15.0-SNAPSHOT-core-h1.jar for your convenience. However, you can download these jar files from https://github.com/apache/incubator-datafu and https://pig.apache.org/releases.html#Download respectively.

If you wish to modify the implementation of EAR4 in your application, you can follow the steps below. Otherwise, you can use the existing myudf.jar in the package and skip the remainder of this subsection.

To build myudf.jar, after modifying EAR4.java in the src folder, compile the source by running the command shown in Listing 1. Next, build the jar file by running the command shown in Listing 2.

Listing 1: Compiling EAR4

```
#!/bin/bash
javac -cp /path/to/weka.jar:
    /path/to/pig-0.15.0-SNAPSHOT-core-h1.jar:
    /path/to/hadoop-2.6.0/share/hadoop/common/*
    /path/to/myudfs/EAR4.java
```

Listing 2: Building myudfs Jar

```
#!/bin/bash
jar -cf myudfs.jar /path/to/myudfs
```

4.2 Running BEAR

BEAR's implementation is adapted from the DataFu kNN implementation. The pig script for BEAR uses EAR4 as a UDF to apply ensemble of adaptations for adjusting nearest neighbors' values.

If you only wish to apply LSH to kNN, without applying BEAR, that can be done by applying the "L2PStableHash" class source code at L2PStableHash.java. For your convenience BEAR's package includes a version of kNN based on "L2PStableHash"'s source code. The basic functionality of this version is identical to that of the original version from datafu, but it has been modified as follows:

1. Train and test data are passed as input parameters to the code.
2. The number of input features is changed to the number of dimensions in MPG domain.
3. The absolute differences between actual and predicted values by k-NN and the final Mean Absolute Error of k-NN are printed out.

changed the code to print out the absolute difference between the actual values and k-NN predictions and the final Mean Absolute Error of k-NN.

In order to run BEAR in local mode, follow the steps in Listing 3.

Listing 3: Running BEAR in local mode

```
#!/bin/bash

export PIG_CLASSPATH=\$PIG_CLASSPATH:/path/to/weka.jar

pig -x local -f /path/to/bear.pig
    -param train=/path/to/mpg1.data
    -param test=/path/to/mpg2.data
```

To run the script in a cluster, remove "-x local" and use the hdfs path for train and test data.

4.3 Tuning BEAR's Predictions

BEAR's script takes two input parameters (i.e. paths to train and test data) as explained in the previous subsection. In addition to those input parameters, BEAR uses three constants to tune the prediction process:

- k: determines the number of nearest neighbors to use for building the final solution
- $attrNo$: the number of input features for the underlying domain. For example, for MPG domain the value of attrNo is seven.
- m_R: the number of rules to be applied per source case to adjust its value.

These constants are basically EAR4's parameters and their values can be set by the user. One way of tuning the values of these parameters is using hill climbing. More details about these values can be found at [2].

4.4 Tuning LSH

In addition to tuning BEAR by setting BEAR-specific parameters, users may set parameters for the underlying LSH. The LSH used in BEAR has five parameters that are set in knn.pig and bear.pig, where "LSH" is defined. The first parameter represents the number of input features of the test domain, which obviously does not need to be tuned. The second parameter is a double value representing the quantization parameter (also known as the projection width). This is set to 100 as its default value, but it could be tuned for different domains. The third parameter is "sRepeat", the number of internal repetitions. The fourth parameter, "sNumHashes", is the size of the hash family (for k-NN, this corresponds to the value of k). This parameter could also be tuned, but because it is shared between BEAR (as k) and LSH, it need only be tuned once (e.g as BEAR's input parameter). Finally, the last parameter is "sSeed", the seed to use when constructing LSH family. Comments in the LSH code provide some additional information on these parameters.

4.5 BEAR's Output

BEAR's script outputs the data in standard output by using the PigLatin dump command. To instead store the results in HDFS, simply replace "dump" with the "store" command. If you run bear.pig as explained above, it will dump two sets of numbers; first, individual actual values of cases in the test set with their predicted values by BEAR (these values have been used for calculating the statistical significance of BEAR's results compared to kNN), second, the Mean Absolute Error of BEAR for the provided test and train sets. You can modify BEAR's script to output other statistics of interest.

References

[1] Jalali, V., Leake, D.: CBR meets big data: A case study of large-scale adaptation rule generation. In: Case-Based Reasoning Research and Development, ICCBR 2015, Berlin, Springer (2015) In press.

[2] Jalali, V., Leake, D.: Extending case adaptation with automatically-generated ensembles of adaptation rules. In: Case-Based Reasoning Research and Development, ICCBR 2013, Berlin, Springer (2013) 188–202

[3] Indyk, P., Motwani, R.: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing. STOC '98, New York, NY, USA, ACM (1998) 604–613

[4] Jalali, V., Leake, D.: Manual for EAR4 and CAAR weka plugins, case-based regression and ensembles of adaptations, version 1. Technical Report TR 717, Computer Science Department, Indiana University, Bloomington, IN (2015)

[5] Hayes, M., Shah, S.: Hourglass: A library for incremental processing on hadoop. In: Big Data, 2013 IEEE International Conference on. (Oct 2013) 742–752

[6] Jalali, V., Leake, D.: Enhancing case-based regression with automatically-generated ensembles of adaptations. Journal of Intelligent Information Systems (2015) In press.

[7] Hanney, K., Keane, M.: Learning adaptation rules from a case-base. In: Proceedings of the Third European Workshop on Case-Based Reasoning, Berlin, Springer Verlag (1996) 179–192

[8] Blake, C., Merz, C.: UCI repository of machine learning databases (2000) http://www.ics.uci.edu/∼mlearn/MLRepository.html.