Demo to dblink

Neil Marchant and Rebecca C. Steorts

Department of Statistical Science, affiliated faculty in Computer Science, | Biostatistics and Bioinformatics, the information initiative at Duke (iiD) and | the Social Science Research Institute (SSRI) | Duke University and U.S. Census Bureau | beka@stat.duke.edu |

October 29, 2019

Assumptions

0. You are familar with entity resolution and the literature.
1. You have read the paper Marchant et al. (2019) and related literature.
2. You have read the documentation on dblink at `https://github.com/cleanzr/dblink`
3. You have the correct software installed on your machine (see the documentation).[1]
4. You have a background in computer science and statistics and are proficient in programming in multiple languages. (This is not a black box program!)

[1]We require Java 8+, Spark 2.3.1, and Scala 2.11, Hadoop 2.7.

Installing dblink

Let's first go through the install instructions on our laptops or on a server.

0: Installing Java

The following two steps require that Java 8+ is installed on your system. To check whether it is installed on a macOS or Linux system, run the command

```
$ java -version
```

Remark: You should see a version number of the form 8.x (or equivalently 1.8.x). Installation instructions for Oracle JDK on Windows, macOS and Linux are available here.

Note: As of April 2019, the licensing terms of the Oracle JDK have changed. We recommend using an open source alternative such as the OpenJDK. Packages are available in many Linux distributions. Instructions for macOS are available here.

1. Get access to a Spark cluster

Since dblink is implemented as a Spark application, you'll need access to a Spark cluster in order to run it.[2]

In this guide, we take an even simpler approach: we'll run Spark in *pseudocluster mode* on your local machine. This is fine for testing purposes or for small data sets.

We'll now take you through detailed instructions for setting up Spark in pseudocluster mode on a macOS or Linux system.

[2] Setting up a Spark cluster from scratch can be quite involved and is beyond the scope of this guide.

1. Get access to a Spark cluster

First, download the prebuilt 2.3.1 release from the Spark release archive.

```
$ wget https://archive.apache.org/dist/spark/spark-2.3.1/spark-2.3.1-bin-hadoop2.7.tgz
```

then extract the archive.

```
$ tar -xvf spark-2.3.1-bin-hadoop2.7.tgz
```

Move the Spark folder to /opt and create a symbolic link so that you can easily switch to another version in the future.

```
$ sudo mv spark-2.3.1-bin-hadoop2.7 /opt
$ sudo ln -s /opt/spark-2.3.1-bin-hadoop2.7/ /opt/spark
```

1. Get access to a Spark cluster

Define the SPARK_HOME variable and add the Spark binaries to your PATH. The way that this is done depends on your operating system and/or shell. Assuming enviornment variables are defined in ~/.profile, you can run the following commands:

```
$ echo 'export SPARK_HOME=/opt/spark' >> ~/.profile
$ echo 'export PATH=$PATH:$SPARK_HOME/bin' >> ~/.profile
```

After appending these two lines, run the following command to update your path for the current session.

```
$ source ~/.profile
```

1. Get access to a Spark cluster

Notes: * If using Bash on Debian, Fedora or RHEL derivatives, environment variables are typically defined in `~/.bash_profile` rather than `~/.profile` * If using ZSH, environment variables are typically defined in `~/.zprofile` * You can check which shell you're using by running `echo $SHELL`

2. Obtain the dblink JAR file

In this step you'll obtain the dblink fat JAR, which has file name `dblink-assembly-0.1.jar`. It contains all of the class files and resources for dblink, packed together with any dependencies.

2. Obtain the dblink JAR file (Option 1)

(Recommended) Download a prebuilt JAR from here. This has been built against Spark 2.3.1 and is not guaranteed to work with other versions of Spark.

2. Obtain the dblink JAR file (Option 2)

Building the fat JAR from source using a tool called sbt. You'll need to install sbt on your system. Instructions are available for Windows, macOS and Linux in the sbt. We give alternative installtion in the second set of instructions for those using bash on MacOS. documentation

2. Obtain the dblink JAR file (Option 2)

On macOS or Linux, you can verify that sbt is installed correctly by running.

```
$ sbt about
```

Once you've successfully installed sbt, get the dblink source code from GitHub:

```
$ git clone https://github.com/ngmarchant/dblink.git
```

then change into the dblink directory and build the package

```
$ cd dblink
$ sbt assembly
```

This should produce a fat JAR at `./target/scala-2.11/dblink-assembly-0.1.jar`.

Note: IntelliJ IDEA can also be used to build the fat JAR. It is arguably more user-friendly as it has a GUI and users can avoid installing sbt.

3. Run dblink

Having completed the above two steps, you're now ready to launch dblink. This is done using the `spark-submit` interface, which supports all types of Spark deployments.

As a test, let's try running the RLdata500 example provided with the source code on your local machine. From within the `dblink` directory, run the following command:

```
$SPARK_HOME/bin/spark-submit \
  --master "local[1]" \
  --conf "spark.driver.extraJavaOptions=-Dlog4j.configuration=log4j.properties" \
  --conf "spark.driver.extraClassPath=./target/scala-2.11/dblink-assembly-0.1.jar" \
  ./target/scala-2.11/dblink-assembly-0.1.jar \
  ./examples/RLdata500.conf
```

3. Run dblink (Options)

This will run Spark in pseudocluster (local) mode with 1 core. You can increase the number of cores available by changing `local[1]` to `local[n]` where `n` is the number of cores or `local[*]` to use all available cores. To run dblink on other data sets you will need to edit the config file (called `RLdata500.conf` above).

Instructions for doing this are provided here.

4. Output of dblink

dblink saves output into a specified directory. In the RLdata500 example from above, the output is written to `./examples/RLdata500_results/`.

1. `run.txt`: contains details about the job (MCMC run). This includes the data files, the attributes used, parameter settings etc.
2. `partitions-state.parquet` and `driver-state`: stores the final state of the Markov chain, so that MCMC can be resumed (e.g. you can run the Markov chain for longer without starting from scratch).
3. `diagnostics.csv` contains summary statistics along the chain which can be used to assess convergence/mixing.
4. `linkage-chain.parquet` contains posterior samples of the linkage structure in Parquet format.

4. Output of dblink (optional files)

1. `evaluation-results.txt`: contains output from an "evaluate" step (e.g. precision, recall, other measures). Requires ground truth entity identifiers in the data files.
2. `cluster-size-distribution.csv` contains the cluster size distribution along the chain (rows are iterations, columns contain counts for each cluster/entity size. Only appears if requested in a "summarize" step.
3. `partition-sizes.csv` contains the partition sizes along the chain (rows are iterations, columns are counts of the number of entities residing in each partition). Only appears if requested in a "summarize" step.

Exercise

Run RLdata500 using the config script provided.

Hint, we want to run the following:

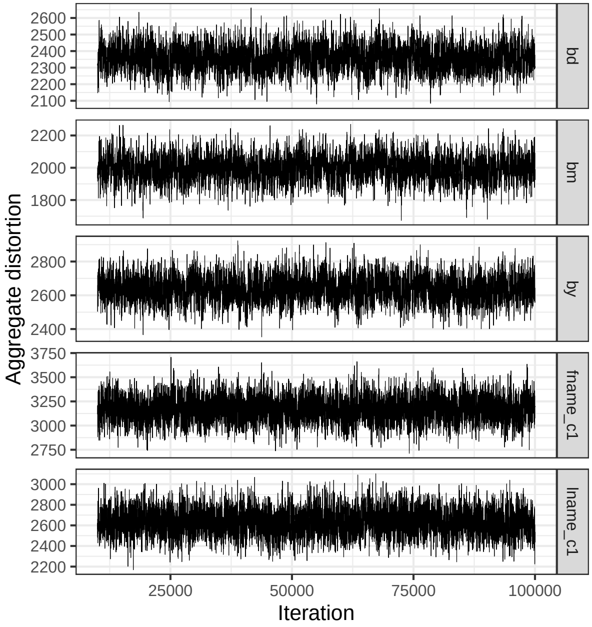
From within the `dblink` directory, run the following command:

```
$SPARK_HOME/bin/spark-submit \
  --master "local[1]" \
  --conf "spark.driver.extraJavaOptions=-Dlog4j.configuration=log4j.properties" \
  --conf "spark.driver.extraClassPath=./target/scala-2.11/dblink-assembly-0.1.jar" \
  ./target/scala-2.11/dblink-assembly-0.1.jar \
  ./examples/RLdata500.conf
```

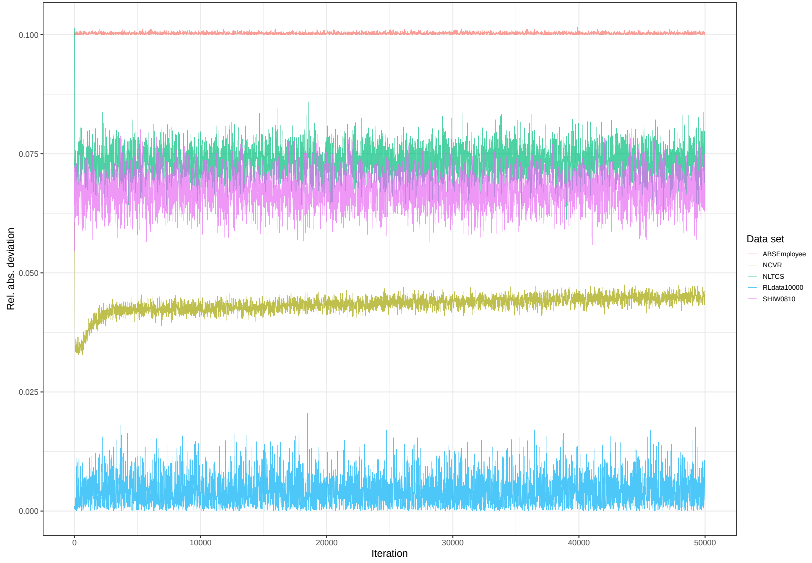
5. How do we analyze some of the output for all of our experiments?

- We can immediately look at the precision, recall, F-measure.
- We can look at trace plots for summary statistics of interest for each data set.
- We can look at the balance loading of the partitions.
- We can look at the posterior bias plot.

Traceplot for feature distortion of RLdata500

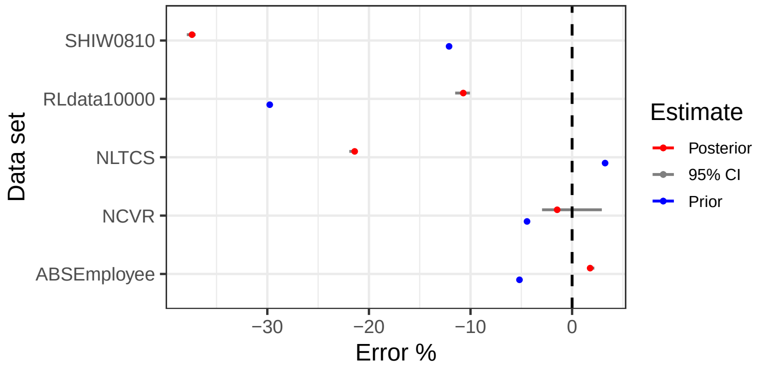


Partition size versus number of MCMC iterations



Balance is measured in terms of the relative absolute deviation from the perfectly balanced configuration. The number of partitions P= 64,64,16,2,8 for each data set (in the order listed in the legend).

Posterior Bias Plot



Suggestions and Exercises

1. Please try and replicate these exercises this week on your own.
2. Replicate all experiments in the paper for all settings that we have specified.
3. Once these have been verified, we can then talk about running this on real data in the wild!

Questions?