Big Data Learning with Evolutionary Algorithms

Isaac Triguero
School of Computer Science
University of Nottingham
United Kingdom
Isaac.Triguero@nottingham.ac.uk

Mikel Galar
Dept. Automatic and Computation
Public University of Navarre
Spain
mikel.galar@unavarra.es

Outline

- Introduction to Big Data
- Big Data Analytics
- Evolutionary algorithms in the big data context
- A demo with MLlib
- Conclusions
There is no a standard definition!

“Big Data” involves data whose volume, diversity and complexity requires new techniques, algorithms and analyses to extract valuable knowledge (hidden).
What is Big Data? The 5V’s definition

- Volume
- Velocity
- Variety
- Veracity

Value
Big data has many faces

- Data Acquisition
- Storage infrastructure
- Computation infrastructure
- Databases/querying
- Analytics/Mining
- Visualization
- Security and Privacy
How to deal with data intensive applications? **Scale-up vs. Scale-out**
Traditional HPC way of doing things

Communication network (Infiniband)

Lots of communication
worker nodes
(lots of them)

Lots of computations

Limited I/O

input data
(relatively small)

central storage

Network for I/O

OS

source data

input data

(i)

Central storage

Source: Jan Fostier. Introduction to MapReduce and its Application to Post-Sequencing Analysis
Data-intensive jobs

Fast communication network (Infiniband)

Limited communication

Low compute intensity

doesn’t scale

Lots of I/O

Network for I/O

central storage

input data (lots of it)
Data-intensive jobs

Solution: store data on local disks of the nodes that perform computations on that data ("data locality")
Distributed systems in Big Data

• **Objective:** To apply an operation to all data
  – One machine cannot process or store all data
    • Data is distributed in a cluster of computing nodes
    • It does not matter which machine executes the operation
    • It does not matter if it is run twice in different nodes (due to failures or stalled nodes)
    • We look for an abstraction of the complexity behind distributed systems
  – **DATA LOCALITY** is crucial
    • Avoid data transfers between machines as much as possible
Distributed systems in Big Data

New programming model: **MapReduce**

– "Moving computation is cheaper than moving computation and data at the same time”

– **Idea**
  
  • Data is distributed among nodes (distributed file system)
  • Functions/operations to process data are distributed to all the computing nodes
  • Each computing node works with the data stored in it
  • Only the necessary data is moved across the network
MapReduce

• Parallel Programming model

• Divide & conquer strategy
  ▪ **divide**: partition dataset into smaller, independent chunks to be processed in parallel (*map*)
  ▪ **conquer**: combine, merge or otherwise aggregate the results from the previous step (*reduce*)

• Based on **simplicity** and **transparency** to the programmers, and assumes **data locality**

• Becomes popular thanks to the open-source project Hadoop! (Used by Google, Facebook, Amazon, ...)
MapReduce: How it works
MapReduce: Example

• WordCount
Hadoop

- **Open source framework for Big Data processing**
  - Based on two Works published by Google
    - Google File System (GFS)[Ghe03]
    - MapReduce algorithm[Dea04]
  - Composed of
    - Hadoop Distributed File System (HDFS) ➔ Storage
    - Implementation of the MapReduce algorithm ➔ Processing

---


Hadoop

• Hadoop is:
  – An open-source framework written in Java
  – Distributed storage of very large data sets (Big Data)
  – Distributed processing of very large data sets

• This framework consists of a number of modules
  – Hadoop Common
  – Hadoop Distributed File System (HDFS)
  – Hadoop YARN – resource manager
  – Hadoop MapReduce – programming model
Hadoop: A master/slave architecture

- **Master**: NameNode, JobTracker
- **Slave**: {DataNode, TaskTracker}, ..., {DataNode, TaskTracker}
Distributed File System: HDFS

- **HDFS – Hadoop Distributed File System**
  - Distributed File System written in Java
  - Scales to clusters with thousands of computing nodes
    - Each node stores part of the data in the system
  - Fault tolerant due to data replication
  - Designed for big files and low-cost hardware
    - GBs, TBs, PBs
  - Efficient for read and append operations (random updates are rare)
  - High throughput (for bulk data) more important than low latency
Hadoop MapReduce: Main Characteristics

• **Automatic parallelization:**
  – Depending on the size of the input data ➔ there will be multiple MAP tasks!
  – Depending on the number of Keys <k, value> ➔ there will be multiple REDUCE tasks!

• **Scalability:**
  – It may work over every data center or cluster of computers.

• **Transparent for the programmer**
  – Fault-tolerant mechanism.
  – Automatic communications among computers
Data Sharing in Hadoop MapReduce

Input

HDFS read

iter. 1

HDFS write

iter. 2

HDFS read

.

HDFS write

.

Input

HDFS read

query 1

result 1

query 2

result 2

query 3

result 3

.

Slow due to replication, serialization, and disk IO
Paradigms that do not fit with Hadoop MapReduce

• Directed Acyclic Graph (DAG) model:
  – The DAG defines the dataflow of the application, and the vertices of the graph defines the operations on the data

• Graph model:
  – More complex graph models that better represent the dataflow of the application
  – Cyclic models -> Iterativity.

• Iterative MapReduce model:
  – An extented programming model that supports iterative MapReduce computations efficiently
New platforms to overcome Hadoop’s limitations

**GIRAPH (APACHE Project)**
(http://giraph.apache.org/)
*Iterative graph processing*

**GPS - A Graph Processing System,**
(Stanford)
http://infolab.stanford.edu/gps/
Amazon's EC2

**Distributed GraphLab**
(Carnegie Mellon Univ.)
https://github.com/graphlab-code/graphlab
Amazon's EC2

**Twister**
(Indiana University)
http://www.iterativemapreduce.org/
Private Clusters

**PrlIter**
(University of Massachusetts Amherst, Northeastern University-China)
http://code.google.com/p/priter/
Private cluster and Amazon EC2 cloud

**HaLoop**
(University of Washington)
http://clue.cs.washington.edu/node/14
http://code.google.com/p/haloop/
Amazon’s EC2

**Spark**
(UC Berkeley)
http://spark.incubator.apache.org/research.html

**GPU based platforms**
Mars
GreX

**Grex**

**Amazon's EC2**
Big data technologies

The World of Big Data Tools

- DAG Model
  - Hadoop

- MapReduce Model
  - MPI
  - HaLoop
  - Twister

- Graph Model
  - Giraph
  - Hama
  - GraphLab
  - GraphX

- BSP/Collective Model
  - Harp
  - Stratosphere
  - Reef

For Iterations/Learning

- Dryad/DryadLINQ
- Pig/PigLatin
- Hive
- Tez
- Shark

For Query

- Drill
- MRQL

For Streaming

- S4
- Storm
- Samza
- Spark Streaming
What is Spark?

Fast and Expressive Cluster Computing Engine Compatible with Apache Hadoop

Efficient

• General execution graphs
• In-memory storage

Usable

• Rich APIs in Java, Scala, Python
• Interactive shell

Up to 10x faster on disk, 100x in memory

2-5x less code
What is Spark?

• Data processing engine (only)
• Without a distributed file system
  – Uses other existing DFS
    • HDFS, NoSQL...
    • Hadoop is not a prerequisite
• Works with different cluster management tools
  – Hadoop (YARN)
  – Mesos
  – Standalone mode (included in Spark)
What is Spark?

Spark SQL structured data
Spark Streaming real-time
MLib machine learning
GraphX graph processing

Spark Core

Standalone Scheduler
YARN
Mesos
Spark Goal

• Provide distributed memory abstractions for clusters to support apps with working sets

• Retain the attractive properties of MapReduce:
  – Fault tolerance (for crashes & stragglers)
  – Data locality
  – Scalability

**Initial Solution:** augment data flow model with “resilient distributed datasets” (RDDs)
RDDs in Detail

• An RDD is a fault-tolerant collection of elements that can be operated on in parallel.

• There are two ways to create RDDs:
  – Parallelizing an existing collection in your driver program
  – Referencing a dataset in an external storage system, such as a shared filesystem, HDFS, Hbase.

• Can be cached for future reuse
Operations with RDDs

- Transformations (e.g. map, filter, groupBy, join)
  - Lazy operations to build RDDs from other RDDs
- Actions (e.g. count, collect, save)
  - Return a result or write it to storage

<table>
<thead>
<tr>
<th>Transformations (define a new RDD)</th>
<th>Parallel operations (return a result to driver)</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>reduce</td>
</tr>
<tr>
<td>filter</td>
<td>collect</td>
</tr>
<tr>
<td>sample</td>
<td>count</td>
</tr>
<tr>
<td>union</td>
<td>save</td>
</tr>
<tr>
<td>groupByKey</td>
<td>lookupKey</td>
</tr>
<tr>
<td>reduceByKey</td>
<td></td>
</tr>
<tr>
<td>join</td>
<td></td>
</tr>
<tr>
<td>cache</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Operations with RDDs

• RDDs – Simple example

```python
>>> lines = sc.textFile("README.md") # Creates an RDD
>>> lines.count() # Counts the number of elements in the RDD
127
>>> lines.first() # First element of the RDD -> 1st line of README.md
u'# Apache Spark`
```

• Simple wordCount in Spark

```python
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" "))
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```
Spark vs. hadoop

Lines of code for K-Means

Spark ~ 90 lines –

Hadoop ~ 4 files, > 300 lines

Spark

- **Driver and Workers**
  - A *Spark program is composed of two programs*
    - **Driver program**
    - **Workers program**
      - Executed in the computing nodes
      - Or in local threads
  - RDDs are distributed across the whole cluster
SparkSQL: Datasets y DataFrames

• **Structured APIs**
  – **DataFrames (>= Spark 1.3)**
    • **Idea:** RDDs of rows with columns that can be accessed by their names
    • **Similar to Pandas in Python (dataframes in R)**
    • Avoid Java serialization performed by RDDs
    • API natural for developers familiar with building query plans (SQL)
    • Introduced as a part of Tungsten project
      – Efficient memory management
    • Concept of schema to describe data
SparkSQL: Datasets y DataFrames

• **Structured APIs**
  – **Datasets (>= Spark 1.6)**
    • Idea: Strongly typed RDDs
    • **Functional transformations** (map, flatMap, filter)
    • **Best of both RDDs and DataFrames**
      – Object-oriented programming
      – Compile-time type safety
      – Catalyst optimization
      – Tungsten off-heap memory optimization
    • **Only for Scala and Java**
SparkSQL: **Datasets y DataFrames**

- **Structured APIs**
  - DataFrames and Datasets
  - Fused in Spark 2.0 (November 2016)
    - A DataFrame is just a Dataset of Rows: Dataset[Row]
  - Both make us of Catalyst and Tungsten projects
SparkSQL: **Datasets y DataFrames**

- **Structured APIs in Spark**
  - Analysis of the *reported errors* before a job is executed

<table>
<thead>
<tr>
<th>Syntax Errors</th>
<th>SQL (Runtime)</th>
<th>DataFrames (Compile Time)</th>
<th>Datasets (Compile Time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis Errors</td>
<td>Runtime</td>
<td>Runtime</td>
<td>Compile Time</td>
</tr>
</tbody>
</table>
Apache Flink is an open source platform for distributed stream and batch data processing.

Flink's core is a streaming dataflow engine that provides data distribution, communication, and fault tolerance for distributed computations over data streams.

Flink includes several APIs for creating applications that use the Flink engine:

1. DataStream API for unbounded streams embedded in Java and Scala, and
2. DataSet API for static data embedded in Java, Scala, and Python,
3. Table API with a SQL-like expression language embedded in Java and Scala.

Flink also bundles libraries for domain-specific use cases:

1. CEP, a complex event processing library,
2. Machine Learning library, and
3. Gelly, a graph processing API and library.

You can integrate Flink easily with other well-known open source systems both for data input and output as well as deployment.

Streaming First

High throughput and low latency stream processing with exactly-once guarantees.

Throughput

Elements per second (millions)

<table>
<thead>
<tr>
<th># CPU Cores</th>
<th>Flink</th>
<th>Storm</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>50</td>
<td>10</td>
</tr>
<tr>
<td>80</td>
<td>100</td>
<td>20</td>
</tr>
<tr>
<td>120</td>
<td>150</td>
<td>30</td>
</tr>
</tbody>
</table>

Batch on Streaming

Batch processing applications run efficiently as special cases of stream processing applications.

APIs, Libraries, and Ecosystem

DataSet, DataStream, and more. Integrated with the Apache Big Data stack.
Big Data: Technology and Chronology

2001-2010

2001
3V’s Gartner
Doug Laney

2010
Spark
U Berckkeley
Apache Spark
Feb. 2014
Matei Zaharia

2004
MapReduce
Google
Jeffrey Dean

2009-2013 Flink
TU Berlin
Flink Apache (Dec. 2014) Volker Markl

2008
Hadoop
Yahoo!
Doug Cutting

2010-2017:
Big Data
Analytics: Mahout, MLLib, ...

Hadoop Ecosystem
Applications
New Technology
Big Data Ecosystem

- Pig: Scripting para MapReduce
- Sqoop: Hadoop to RDBMS
- Storm: Streaming data
- Apache HBase: NoSQL on HDFS
- Zepelin: Web-based notebook
- Docker: Containers
- Kafka: Gestión de fuentes de datos
- Avro: Data serialization
- Flink: Streaming data
- Hive: SQL
- Tez: DAG execution
- Cassandra: NoSQL Database
- ZooKeeper: Coordinador
- Kubernetes: Cluster management
- Mesos: Cluster management
- Impala: Interactive SQL
- Flume: Log data
Outline

- Introduction to Big Data
- Big Data Analytics
- Evolutionary algorithms in the big data context
- A demo with MLlib
- Conclusions
Big Data Analytics

Potential scenarios

Clustering

Classification

Real Time Analytics/Big Data Streams

Association

Recommendation Systems

Social Media Mining
Social Big Data
# Big Data Analytics: A 3 generational view

<table>
<thead>
<tr>
<th>Generation</th>
<th>1st Generation</th>
<th>2nd Generation</th>
<th>3rd Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Examples</strong></td>
<td>SAS, R, Weka, SPSS, KEEL</td>
<td>Mahout, Pentaho, Cascading</td>
<td>Spark, Haloop, GraphLab, Pregel, Giraph, ML over Storm</td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td>Vertical</td>
<td>Horizontal (over Hadoop)</td>
<td>Horizontal (Beyond Hadoop)</td>
</tr>
<tr>
<td><strong>Algorithms</strong></td>
<td>Huge collection of algorithms</td>
<td>Small subset: sequential logistic regression, linear SVMs, Stochastic Gradient Descendent, k-means clustering, Random forest, etc.</td>
<td>Much wider: CGD, ALS, collaborative filtering, kernel SVM, matrix factorization, Gibbs sampling, etc.</td>
</tr>
<tr>
<td><strong>Algorithms</strong></td>
<td>Practically nothing</td>
<td>Vast no.: Kernel SVMs, Multivariate Logistic Regression, Conjugate Gradient Descendent, ALS, etc.</td>
<td>Multivariate logistic regression in general form, k-means clustering, etc. – Work in progress to expand the set of available algorithms</td>
</tr>
<tr>
<td><strong>Fault-</strong></td>
<td>Single point of failure</td>
<td>Most tools are FT, as they are built on top of Hadoop</td>
<td>FT: HaLoop, Spark Not FT: Pregel, GraphLab, Giraph</td>
</tr>
<tr>
<td><strong>Tolerance</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Mahout (Samsara)

• First ML library initially based on Hadoop MapReduce.
• Abandoned MapReduce implementations from version 0.9.
• Nowadays it is focused on a new math environment called Samsara.
• It is integrated with Spark, Flink and H2O
• Main algorithms:
  – Stochastic Singular Value Decomposition (ssvd, dssvd)
  – Stochastic Principal Component Analysis (spca, dspca)
  – Distributed Cholesky QR (thinQR)
  – Distributed regularized Alternating Least Squares (dals)
  – Collaborative Filtering: Item and Row Similarity
  – Naive Bayes Classification

http://mahout.apache.org/
Spark Libraries

MLlib types, algorithms and utilities

- Data types
- Basic statistics
  - summary statistics
  - correlations
  - stratified sampling
  - hypothesis testing
  - random data generation
- Classification and regression
  - linear models (SVMs, logistic regression, linear regression)
  - naïve Bayes
  - decision trees
  - ensembles of trees (Random Forests and Gradient-Boosted Trees)
  - isotonic regression
- Collaborative filtering
  - alternating least squares (ALS)

https://spark.apache.org/mllib/
FlinkML

FlinkML is the Machine Learning (ML) library for Flink. It is a new effort in the Flink community, with a growing list of algorithms and contributors. With FlinkML we aim to provide scalable ML algorithms, an intuitive API, and tools that help minimize glue code in end-to-end ML systems. You can see more details about our goals and where the library is headed in our vision and roadmap here.

Important: Maven artifacts which depend on Scala are now suffixed with the Scala major version, e.g. "2.10" or "2.11". Please consult the migration guide on the project Wiki.

Supported Algorithms
- Supervised Learning
- Data Preprocessing
- Recommendation
- Utilities
- Getting Started
- Pipelines
- How to contribute

Scalability

• **Speed-up (m cores)**
  – How much faster can the same data be processed with m cores instead of 1 core
    
    \[ \text{Speedup}(m) = \frac{\text{runtime on 1 core}}{\text{runtime on } m \text{ cores}} \]

  – The data size is kept constant and the number of cores is increased
  – Ideal speed-up is linear
    • Speedup(m) = m
  – In practice
    • Difficult to obtain due to communication and synchronization overhead

---

<table>
<thead>
<tr>
<th>number of processors</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sun</td>
</tr>
<tr>
<td></td>
<td>Gray</td>
</tr>
<tr>
<td></td>
<td>sg1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
</tr>
</tbody>
</table>

---
Scalability

• **Size-up (data, m)**
  - How much time does it take to execute m times larger data
    - \( Sizeup(data, m) = \frac{\text{runtime for processing } m \cdot \text{data}}{\text{runtime for processing data}} \)
  - The number of cores is kept constant and the data size is increased
  - Ideal size-up is linear
    - \( Sizeup(data, m) = m \)
  - In practice
    - Few algorithms are linear with respect to the data
Scalability

- **Scale-up (data, m)**
  - Measures the ability of the system to run a $m$-times greater job with a $m$-times larger system
    - $Sizeup(data, m) = \frac{\text{runtime for processing on 1 core}}{\text{runtime for processing } m \cdot \text{data on } m \text{ cores}}$
  - Both the number of cores and the data are increased
  - Ideal scale-up is 1
    - Scale-up(data, m) = 1
  - In practice
    - Few algorithms achieve a scale-up of 1
Machine Learning in Big Data: Global vs. Local

Two main ways for learning a model in Big Data:

– **Locally**
  
  • A model is created for each partition of the data (only using the data of that partition)
  
  • **All the models are combined** when predicting the class of a new example → Ensemble

– **Globally**

  • A **single model** is created using **all the available data**
  
  • They try to obtain the same model as the one that would be obtained if the method could be executed in a single node
Machine Learning in Big Data: Global vs. Local

Local model

- **Advantages**
  - Usually faster
  - Gets faster as the number of partitions is increased
  - Any existing model can be applied
  - Only the aggregation phase has to be designed

- **Disadvantages**
  - **Slow in test phase**, too many models have to be executed
  - **Loss of accuracy** as the number of partitions increases
    - With few partitions, accuracy can improve due to the ensemble effect
    - With too many partitions, the accuracy tends to drop, since there are not enough examples in each partition
  - They do not take advantage of the data as a whole
Global model

– **Advantages**
  • Greater **accuracy** is expected (not proved)
  • All the examples are used to learn a single model
  • Anyway, a global ensemble can also be built
  • The model is independent of the number of partitions
  • Faster in test phase

– **Disadvantages**
  • **More complex design and implementation**
  • Distributed nature of Big Data processing has to be taken into account (computation/communication)
Decision Trees for Big Data

Decision Trees in Spark

- **Differences** with respect to classical models
  - All the nodes in a level are learned with a single pass through the whole dataset
  - Numeric attributes are discretized into bins in order to reduce the computational cost
Decision Trees for Big Data

- **Decision Trees in Spark**
  - **Differences** with respect to classical models
    - All the nodes in a level are learned with a single pass through the whole dataset
Decision Trees for Big Data

• **Decision Trees in Spark**
  – **Differences** with respect to classical models
    • Numeric attributes are discretized into bins in order to reduce the computational cost
Outline

- Introduction to Big Data
- Big Data Analytics
- Evolutionary algorithms in the big data context
- A demo with MLlib
- Conclusions
Data Preprocessing: Tasks to **discover quality data** prior to use knowledge extraction algorithms.
Evolutionary algorithms for data preprocessing

• Many preprocessing stages can be modelled as optimisation processes. For example:
  – Feature selection/weighting
  – Instance selection/Generation
• Evolutionary algorithms have excelled in this task in data with a moderate size.
• However, their practical application is limited to problems with no more than tens of thousands of instances because of:
  – Excessive chromosome size
  – Runtime requirements
Evolutionary algorithms for instance reduction in Big data


Objective: reduce the number of samples to find better decision boundaries between classes, by selecting relevant samples or artificially generating new ones.

We focused on Prototype Generation (PG) models, which are based on the Nearest Neighbour (NN) classifier.

Advantages:

✓ Reduce Storage Requirements
✓ Remove noisy samples
✓ Speed up learning process
Evolutionary Prototype Generation (EPG)

- EPG algorithms adjust the positioning of the prototypes.
- Each individual encodes a single prototype or a complete generated set with **real codification**.
- The fitness function is computed as the classification performance in the training set using the Generated Set.
- Currently, best performing approaches use **Differential Evolution**.
- **Known issues:**
  - Dealing with big data becomes impractical

Evolutionary Prototype Generation for Big Data sets

Objectives

- The design of a scalable EPG approach that embraces the huge storage and processing capacity of cloud platforms.
- To do so, we rely on the success of **Hadoop MapReduce** in combination with a **windowing** scheme for evolutionary models.
Parallelising EPG with windowing

Training set

Iterations

Main properties:

✓ Avoids a (potentially biased) static prototype selection/generation
✓ This mechanism also introduces some generalization pressure
Parallelising EPG with windowing

Properties:

■ Within this scheme, the algorithm **disposes of the whole information** although it is accessed in successive iterations.

■ This model itself aims to **improve the runtime requirements** of EPG models. But it does not deal with the memory consumption problem.

■ This is why we use this strategy as a **second level parallelization** scheme after a previous distribution of the processing in a cluster of computing elements.
MRPR: Evolutionary Prototype Generation for Big Data sets
Experimental Study

- 4 big data sets: Poker (1M), KddCup (4.8M), Susy (5M), RLCP(5.7M).
- Performance measures: Accuracy, reduction rate, runtime, test classification time and speed up.
- 3x5 fold-cross validation
- Number of mappers = 64/128/256/512/1024.
- Number of reducers=1
- PG techniques tested: SSMA-SFLSDE, LVQ3, RSP3
- PS techniques tested: DROP3, FCNN
Evolutionary Prototype Generation for Big Data sets

Table: Results obtained for the PokerHand problem.
Evolutionary Prototype Generation for Big Data sets
Evolutionary Prototype Generation for Big Data sets

<table>
<thead>
<tr>
<th>#Windows nw</th>
<th>#Mappers</th>
<th>Training Avg.</th>
<th>Std.</th>
<th>Test Avg.</th>
<th>Std.</th>
<th>Runtime Avg.</th>
<th>Std.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
<td>0.5121</td>
<td>0.0028</td>
<td>0.5120</td>
<td>0.0031</td>
<td>15058.4740</td>
<td>1824.6586</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>0.5115</td>
<td>0.0035</td>
<td>0.5113</td>
<td>0.0036</td>
<td>8813.7134</td>
<td>678.1335</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>0.5038</td>
<td>0.0032</td>
<td>0.5039</td>
<td>0.0033</td>
<td>4666.5424</td>
<td>412.5351</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>0.5052</td>
<td>0.0060</td>
<td>0.5055</td>
<td>0.0057</td>
<td>4095.8610</td>
<td>941.5737</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>0.5041</td>
<td>0.0024</td>
<td>0.5034</td>
<td>0.0022</td>
<td>3244.0716</td>
<td>534.8720</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>0.5031</td>
<td>0.0042</td>
<td>0.5028</td>
<td>0.0041</td>
<td>2639.4266</td>
<td>360.3121</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>0.5000</td>
<td>0.0067</td>
<td>0.4998</td>
<td>0.0069</td>
<td>2099.5182</td>
<td>339.7356</td>
</tr>
</tbody>
</table>

![Graph showing speedup vs number of windows](image-url)
Evolutionary Prototype Generation for Big Data sets

- Evolutionary algorithms continue to be the best performing models for Instance reduction in the big data context.
- Great synergy between the windowing and MapReduce approaches. They complement themselves in the proposed two-level scheme.
- Without windowing, EPG could not be applied to datasets larger than approximately ten thousands instances.
- The application of this model has resulted in a very big reduction of storage requirements and classification time for the NN rule.

https://github.com/triguero/MRPR
Evolutionary algorithms for imbalanced Big data sets


Class imbalance problem

An example of an imbalanced data-set
Class imbalance problem

Standard classifiers \(\rightarrow\) models biased in favour of the majority class
Class imbalance problem

- Skewed data distribution by itself is not harmful
- But... a series of **difficulties** usually turn up
  - Small sample size
  - Overlapping or class separability
  - Small disjuncts

(a) Class overlapping  
(b) Small disjuncts
Class imbalance problem

- Two main approaches to tackle this problem:
  - Data sampling
    - Undersampling
    - Oversampling
    - Hybrid approaches
  - Algorithmic modifications
  - Cost-sensitive approaches
  - Ensemble models

V. López et al, An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics, Information Sciences 250 (2013)
Evolutionary Undersampling

- Evolutionary undersampling (EUS) aims to select the best subset of negative instances from the original training set.

- EUS not only intends to balance the training set, but also to increase the overall performance on both classes of the problem.

- To do so, a genetic algorithm is used to search for an optimal subset of instances.

- This resulting set can be used by any standard classification model.

Evolutionary Undersampling in the big data context

- EUS does not generate more data, as opposed to oversampling methods.
- However, the increasing number of instances would lead to obtain an excessive chromosome size that can limit their application.
- The required runtime increases not only with the number of examples but also with the imbalance ratio (IR).
Evolutionary undersampling

- Representation of the solution

\[ V = (\nu_1, \nu_2, \nu_3, \nu_4, \ldots, \nu_n), \quad \nu_i \in \{0, 1\} \quad \text{for all} \quad i = 1, \ldots, n^- \]

- Performance: g-mean, 1NN hold-one-out

\[ g\text{-mean} = \sqrt{\text{TP}\text{rate} \cdot \text{TN}\text{rate}} \]

- Fitness Function

\[ \text{fitness}_{EUS} = \begin{cases} g\text{-mean} - \left| 1 - \frac{n^+}{N^-} \right| \cdot P & \text{if } N^- > 0 \\ g\text{-mean} - P & \text{if } N^- = 0, \end{cases} \]

- We use the CHC algorithm and GM as performance measure.

---

EUS-BD: A two-level parallelisation model for EUS

- A two-level parallelisation scheme:
  - The MapReduce phase will allow us to divide the computational effort over different machines.  
    \textit{Goal: Memory limitations and runtime.}
  - The windowing scheme will be applied to reduce the computational time required by EUS.  
    \textit{Goal: Runtime.}
Windowing for Class Imbalance

- Disjoint windows with equal class distribution may lead to information loss of the positive class.
- The minority class set will be always used to evaluate a chromosome.
- The majority class set is divided into several disjoint strata. The size of each subset will correspond to the number of minority class instances.
  - It means: Fixed number of strata.
The EUS-BD scheme
EUS-BD known issues

- This is a local model!
- Extremely imbalanced cases: **lack of density** from the minority class.
- We used Spark to alleviate that issue splitting training data into positive and negative sets. The positive set was accessible in all the maps (via broadcast)
EUS-BD for Extremely Imbalanced datasets
EUS-BD for Extremely Imbalanced datasets

<table>
<thead>
<tr>
<th>Data set</th>
<th>#features</th>
<th>#negative</th>
<th>#positive</th>
<th>IR</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECBDL’14 (50%)</td>
<td>631</td>
<td>17101086</td>
<td>344333</td>
<td>49</td>
</tr>
<tr>
<td>ECBDL’14 (25%)</td>
<td>631</td>
<td>8550324</td>
<td>172386</td>
<td>49</td>
</tr>
<tr>
<td>Kddcup DOS vs. PRB</td>
<td>41</td>
<td>3883370</td>
<td>41102</td>
<td>94.48</td>
</tr>
<tr>
<td>Kddcup DOS vs. R2L</td>
<td>41</td>
<td>3883370</td>
<td>1126</td>
<td>3448.82</td>
</tr>
<tr>
<td>Kddcup DOS vs. U2R</td>
<td>41</td>
<td>3883370</td>
<td>52</td>
<td>74680.25</td>
</tr>
</tbody>
</table>

TABLE III: Running times obtained by the Hadoop and Spark implementations of EUS-ImbBD and EUS-ExtImbBD.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>No. of maps</th>
<th>Build time (s)</th>
<th>Classif. time (s)</th>
<th>Build time (s)</th>
<th>Classif. time (s)</th>
<th>Spark improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kddcup DOS vs. PRB</td>
<td>128</td>
<td>422.4786</td>
<td>34.264</td>
<td>297.5048</td>
<td>0.2942</td>
<td>29.58%</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>240.4662</td>
<td>36.7934</td>
<td>143.3428</td>
<td>0.3566</td>
<td>40.39%</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>156.4354</td>
<td>48.424</td>
<td>87.0195</td>
<td>0.2739</td>
<td>44.37%</td>
</tr>
<tr>
<td>Kddcup DOS vs. R2L</td>
<td>128</td>
<td>444.7252</td>
<td>31.7255</td>
<td>320.8192</td>
<td>0.0876</td>
<td>27.86%</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>266.2424</td>
<td>36.1147</td>
<td>187.4562</td>
<td>0.1024</td>
<td>29.59%</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>178.8536</td>
<td>42.0057</td>
<td>148.319</td>
<td>0.1371</td>
<td>17.07%</td>
</tr>
<tr>
<td>Kddcup DOS vs. U2R</td>
<td>128</td>
<td>459.6002</td>
<td>31.8436</td>
<td>340.2297</td>
<td>0.0986</td>
<td>25.97%</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>248.1038</td>
<td>35.5862</td>
<td>193.0784</td>
<td>0.1081</td>
<td>22.18%</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>152.3752</td>
<td>46.6194</td>
<td>101.683</td>
<td>0.1275</td>
<td>33.27%</td>
</tr>
</tbody>
</table>
EUS-BD for Extremely Imbalanced datasets

<table>
<thead>
<tr>
<th>Method</th>
<th>#Maps</th>
<th>Build time(s)</th>
<th>Classif. time(s)</th>
<th>AUC</th>
<th>GM</th>
</tr>
</thead>
<tbody>
<tr>
<td>EUS-S-ImbBD</td>
<td>8192</td>
<td>497.0958</td>
<td>0.4999</td>
<td>0.5556</td>
<td>0.3572</td>
</tr>
<tr>
<td></td>
<td>4096</td>
<td>775.2016</td>
<td>0.5531</td>
<td>0.4821</td>
<td>0.4276</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td>1476.3512</td>
<td>0.7878</td>
<td>0.6674</td>
<td>0.6645</td>
</tr>
<tr>
<td>EUS-S-ExtImbBD</td>
<td>8192</td>
<td>2181.5089</td>
<td>3.5404</td>
<td>0.6641</td>
<td>0.6640</td>
</tr>
<tr>
<td></td>
<td>4096</td>
<td>3456.5938</td>
<td>6.0428</td>
<td>0.6662</td>
<td>0.6657</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td>6433.8072</td>
<td>9.4064</td>
<td>0.6731</td>
<td>0.6704</td>
</tr>
<tr>
<td>RUS-S-ImbBD</td>
<td>8192</td>
<td>557.4869</td>
<td>0.5540</td>
<td>0.6319</td>
<td>0.6066</td>
</tr>
<tr>
<td></td>
<td>4096</td>
<td>518.1471</td>
<td>1.0960</td>
<td>0.4659</td>
<td>0.4339</td>
</tr>
<tr>
<td></td>
<td>2048</td>
<td>483.7361</td>
<td>1.0960</td>
<td>0.6651</td>
<td>0.6622</td>
</tr>
</tbody>
</table>
Can we develop a Global Evolutionary model?

- Ideally, EUS should have access to all the data as a whole. **Can we do that with the current technology?**

- A binary representation of the selected instances implies a chromosome size equal to the number of negative instances.

- However, the resulting selected dataset tends to be fairly small, and balanced.

- **Assumption:** The number of positive instance is so reduced that it perfectly fits in main memory of a single computer.
Global Evolutionary Undersampling

HDFS side

Training data

Repeat until Stopping criteria are reached

Data Distribution

Initialisation

Evaluate Initial Population

Combine

Evaluate new Chromosomes

If d <= 0

Re-initialise population

Evaluate population

Driver node

PosTrainRDD

Map M

NegTrainRDD

Map M

Node M

PosTrainRDD

Map 1

NegTrainRDD

Map 1

Node 1

Evaluate function

collect()

For each Chromosome

negativeSetSelected

Union

ReducedSet

Learn a Model

Classify Window of Training Set

Driver node
Outline

- What is Big data?
- How to deal with Data Intensive applications?
- Big Data Analytics
- A demo with MLlib
- Conclusions
Demo

• In this demo we will show two ways of working with Apache Spark:
  – Interactive mode with Spark Notebook.
  – Standalone mode with IntelliJ.

• All the code used in this presentation is available at:
DEMO with Spark Notebook in local

SPARK NOTEBOOK

http://spark-notebook.io/

Interactive and Reactive Data Science using Scala and Spark. spark-notebook.io

347 FORKS 1.5K STARS
DEMO with Spark Notebook in local

Advantages:
✓ Interactive.
✓ Automatic plots.
✓ It allows connection with a cluster.
✓ Tab completion

Disadvantages:
❑ Built-in for specific spark versions.
❑ Difficult to integrate your own code.
DEMO with IntelliJ IDE

https://www.jetbrains.com/idea/
MapReduce in a cluster

- Hadoop V2
  - YARN
MapReduce in a cluster

- Hadoop V2
MapReduce in a cluster

• **Hadoop V2**
  – **Container**
    • A subset of the resources of the cluster (part of a node)
      – Number of **cores**
      – Quantity of **RAM memory**
    • A hold request is made
    • Once granted, a process (task) can be run in the container
MapReduce in a cluster

• **Hadoop V2**
  – **ApplicationMaster**
    • New concept
    • **Responsible for the processing**
    • Responsible for negotiating with the ResourceManager and working with the NodeManagers
    • **In charge of the fault tolerance**
      – ResourceManager is no longer used for this task
MapReduce in a cluster

• **Hadoop V2**
  – **Execution** of a MapReduce process
    1. The client launches the process (connection with the ResourceManager)
MapReduce in a cluster

- **Hadoop V2**
  - Execution of a MapReduce process

2. The ResourceManager requests a container where the ApplicationMaster is executed
MapReduce in a cluster

• **Hadoop V2**
  
  – **Execution** of a MapReduce process
    
    3. The ApplicationMaster request the containers to execute all the tasks (in different nodes)
MapReduce in a cluster

• **Hadoop V2**
  – **Execution** of a MapReduce process
    4. All the tasks are executed in the containers
      – Containers are released once its tasks are finished
    5. The ApplicationMaster ends when all the tasks have been executed. Then, its container is released
Spark: Execution in a cluster

- **SparkContext** (sc) is created in the driver
  - Using the sc a connection with the cluster manager is established
  - Once connected, executors are requested
    - The processes that perform the computation and store the data
  - The driver sends the code and tasks to the executors
Outline

- Introduction to Big Data
- Big Data Analytics
- Evolutionary algorithms in the big data context
- A demo with MLlib
- Conclusions
Conclusions

• We need new strategies to deal with big datasets
  – Choosing the right technology is like choosing the right data structure in a program.
• The world of big data is rapidly changing. Being up-to-date is difficult but necessary.
• Evolutionary models are powerful tools in data mining. They need to be adapted and redesigned to take the maximum advantages of their promising properties.
Big Data Learning with Evolutionary Algorithms

Isaac Triguero
School of Computer Science
University of Nottingham
United Kingdom
Isaac.Triguero@nottingham.ac.uk

Mikel Galar
Dept. Automatic and Computation
Public University of Navarre
Spain
mikel.galar@unavarra.es

Extra slides
Evolutionary algorithms for Feature Selection/Weighting in Big Data


Feature Selection

The outcome of FS would be:

- Less data → algorithms could learn quicker
- Higher accuracy → the algorithm generalizes better
- Simpler results → easier to understand them
Each individual represents a set of selected features (**binary vector**).

The individuals are crossed and mutated to generate new candidate sets of features.

Fitness function:
- Classification performance in the training dataset using only the features in the corresponding set.

---

Evolutionary Feature Weighting

- Each individual represents the importance of the features in the range 0,1 (real vector).
- The individuals are crossed and mutated to generate new candidate importances.
- Fitness function:
  - Classification performance in the training dataset using taking into consideration the weights of the features.

Evolutionary Feature Selection/Weighting for Big Data

MapReduce EFS process

The vector of weights is binarized with a threshold
Evolutionary Feature Selection/Weighting for Big Data

Dataset reduction

The maps remove the discarded features
Evolutionary Feature Selection

Experimental Study: EFS scalability in MapReduce

Table 3: Execution times (in seconds) over the epsilon subsets

<table>
<thead>
<tr>
<th>Instances</th>
<th>Sequential CHC</th>
<th>MR-EFS</th>
<th>Splits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>391</td>
<td>419</td>
<td>1</td>
</tr>
<tr>
<td>2000</td>
<td>1352</td>
<td>409</td>
<td>2</td>
</tr>
<tr>
<td>5000</td>
<td>8667</td>
<td>413</td>
<td>5</td>
</tr>
<tr>
<td>10000</td>
<td>39576</td>
<td>431</td>
<td>10</td>
</tr>
<tr>
<td>15000</td>
<td>91272</td>
<td>445</td>
<td>15</td>
</tr>
<tr>
<td>20000</td>
<td>159315</td>
<td>455</td>
<td>20</td>
</tr>
<tr>
<td>400000</td>
<td>—</td>
<td>6531</td>
<td>512</td>
</tr>
</tbody>
</table>

- CHC is quadratic w.r.t. the number of instances
- Splitting the dataset yields nearly quadratic acceleration
Evolutionary Feature Selection

Experimental Study: Classification

- Two datasets
  - epsilon
  - ECBDL14, after applying Random Oversampling
- The reduction rate is controlled with the weight threshold

- Three classifiers in Spark
  - SVM
  - Logistic Regression
  - Naïve Bayes
- Performance measures
  - \[ \text{AUC} = \frac{\text{TPR} + \text{TNR}}{2} \]
- Training runtime

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Training instances</th>
<th>Test instances</th>
<th>Features</th>
<th>Splits</th>
<th>Instances per split</th>
</tr>
</thead>
<tbody>
<tr>
<td>epsilon</td>
<td>400 000</td>
<td>100 000</td>
<td>2000</td>
<td>512</td>
<td>~780</td>
</tr>
<tr>
<td>ECBDL14</td>
<td>31,992,921</td>
<td>2,897,917</td>
<td>631</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>ECBDL14-ROS</td>
<td>65,003,913</td>
<td>2,897,917</td>
<td>631</td>
<td>32,768</td>
<td>~1,984</td>
</tr>
</tbody>
</table>
Evolutionary Feature Selection

Experimental Study: results

Table 4: AUC results for the Spark classifiers using epsilon

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Features</th>
<th>Logistic Regression</th>
<th>Naive Bayes</th>
<th>SVM ($\lambda = 0.0$)</th>
<th>SVM ($\lambda = 0.5$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Training</td>
<td>Test</td>
<td>Training</td>
<td>Test</td>
</tr>
<tr>
<td>0.00</td>
<td>2000</td>
<td>0.6786</td>
<td>0.6784</td>
<td>0.7038</td>
<td>0.7008</td>
</tr>
<tr>
<td>0.55</td>
<td>721</td>
<td>0.6985</td>
<td>0.7000</td>
<td>0.7154</td>
<td>0.7127</td>
</tr>
<tr>
<td>0.60</td>
<td>337</td>
<td>0.6873</td>
<td>0.6867</td>
<td>0.7054</td>
<td>0.7030</td>
</tr>
<tr>
<td>0.65</td>
<td>110</td>
<td>0.6496</td>
<td>0.6497</td>
<td>0.6803</td>
<td>0.6794</td>
</tr>
</tbody>
</table>
Evolutionary Feature Weighting at GECCO-2014

- Details of the training data:
  - 32 million training samples (56.7GB of disk space)
  - 2.9 million of test samples (5.1GB of disk space)
  - 631 features (539 real & 92 nominal values)
  - 2 labels; 98% non-contact samples

## Results

<table>
<thead>
<tr>
<th>Team</th>
<th>TPR</th>
<th>TNR</th>
<th>TPR * TNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efdamis</td>
<td>0.73043</td>
<td>0.73018</td>
<td>0.53335</td>
</tr>
<tr>
<td>ICOS</td>
<td>0.70321</td>
<td>0.73016</td>
<td>0.51345</td>
</tr>
<tr>
<td>UNSW</td>
<td>0.69916</td>
<td>0.72763</td>
<td>0.50873</td>
</tr>
<tr>
<td><strong>Efdamis-Without FS</strong></td>
<td><strong>0.7041</strong></td>
<td><strong>0.7103</strong></td>
<td><strong>0.500175</strong></td>
</tr>
<tr>
<td>HyperEns</td>
<td>0.64003</td>
<td>0.76338</td>
<td>0.48858</td>
</tr>
<tr>
<td>PUC-Rio_ICA</td>
<td>0.65709</td>
<td>0.71460</td>
<td>0.46956</td>
</tr>
</tbody>
</table>

Evolutionary Feature Selection/Weighting

- The proposed MapReduce processes provide several advantages:
  - It enables tackling Big Data problems
  - The feature weight vector is more flexible than a binary vector
- The data reduction process in MapReduce provides a scalable and flexible way to apply the feature selection/weighting,
- Both the accuracy and the runtime of the classification were improved after the preprocessing.

https://github.com/triguero/MR-EFS