# NASA Padmini Project: Outlier Detection

Sandipan Dey,
CSEE Department,
University of Maryland, Baltimore County,
MD, USA,
sandip2@umbc.edu

May 18, 2010

## 1 The Problem

Here we are interested in finding anamolous behaviour among the celestial objects in the sky. We have an enormous amount of data where each tuple represents an object, different attributes of which are being measured by the virtual observatory. We are interested in partitioning the sky into several regions and examine whether the local behavior in any of these sites differ significantly from the global behavior. We shall be using eigen-analysis to define the global behavior by the notion of global eigenvectors that are obtained by aggregating the local ones.

The user is going to submit a query to the VO in order to select a (circlular) region in the sky and get the data inside the region. Since even the result of the query can bring in a huge amount of data, we shall be interested in using the power of parallel computation to process the data parallelly, hence we shall horizontally partition the data, i.e., further divide the circular region into different sub-regions (sphere packing?) and process the data chunks parallelly. Since there is inherent parallelism in the processing of the data, we can use map-reduce framework to implement the same.

Note that it's crucial and the above approach makes sense only if we can ensure that each of the parallel phases we process data from strictly different (no-overlapping) location of the sky. We can ensure this by having different files for data from different locations.

## 2 The Approach

### 2.1 The Algorithm

Our algorithm for distributed outlier detection will be based on PCA. We shall be using the fact that the most dominant eigenvectors found by the eigen-analysis of the covariance matrix captures the directions with highest variance in data. Accordingly, tuples that are missed out by these eignvectors are outliers.

**Algorithm 1** Distributed || Outlier Detection

```
1: Horizontally partition the (z-score) normalized data $D_{m \times n}$ into $N$ data chunks $D^i_{m_i \times n}$, $D = \bigcup_{i=1}^{N} D^i$ and assign $i^{th}$ partition to node $\aleph_i$.
2: Run PCA parallelly on each of the local data $D^i$ (possessed by the node $\aleph_i$), i.e., to obtain a set of orthogonal eigenvectors $V^i_{n \times n}$ at each node.
3: Choose top $k$ most dominant eigenvectors ($\hat{V}^i = V^i_{k \times n}$, corresponding to the largest eigenvalues of the covariance matrix) from each of the nodes.
4: Combine these eigenvectors (use some aggregation as simple average or weighted average with weights as the corresponding eignevalues), in order to approximate the set of top $k$ global eigenvectors: $\hat{V}^i_g = \frac{1}{N} \sum_{i=1}^{N} \hat{V}^i$.
5: Project the local data in each of the nodes $\aleph_i$ onto the top $k$ most dominant global eigenvectors: $\hat{D}^i = D_i.\hat{V}^i_g.\hat{V}^{iT}_g$.
6: For each data tuple $D^i_k$ in node $\aleph_i$, parallely calculate the corresponding error term in projection by $||D^i_k - D^i_k||_2$ and assign a normalized outlier score (in the range $[0, 1]$, measuring the degree of outlierness, 1 with the most outlying properties) by $s^i_k = \frac{||D^i_k - D^i_k||_2}{\max_k ||D^i_k - D^i_k||_2}$.
7: Use threshold based on $3\sigma$ limits etc. to mark the outliers.
```

## 2.2 The Implementation

- As clear from the algorithm, the easiest way to implement this is to use a couple of map / reduce phases using hadoop.
- Both the computation of local eigenvectors (by PCA) and assignment of outlier scores are the ones that are to done locally and are with inherent parallelism, hence should be excuted parallelly and independently in maps.
- The computation of approximate global eigenvectors by combining the local ones can be done in reduce phase by combining the map inputs.
- Since intially we don't have the original data, rather we have the meta-data (ra,dec) coordinates provided by the user, we need to read the actual data by querying the VO in the first map phase, before starting the computation.

### 2.2.1 The First Map-Reduce Phase

The first map-reduce phase will find the global eigenvectors and store on HDFS.

1. The Map Phase:

   (a) The meta-data ((ra,dec) coordinates) will be divided into several chunks (by hadoop) and fed to parallel map instances.
   (b) Input to the map will simply be the meta-data tuples (as key-value pairs).
   (c) The map phase will first query the VO with (ra, dec) coordinates as arguments and fetch the actual data from the VOs. This is a time-consuming task.
   (d) Then it will do PCA on the data fetched, i.e., find the local eigenvectors from the covariance matrix computed.
   (e) Choose top $k$ eigenvectors from the local eignevector matrix.
   (f) Send the top-$k$ local eigenvectors to reduce phase, along with the eigenvectors it will pass the fetched data tuples as well.

2. The Reduce Phase:
   (a) In reduce phase, first separate out the eigenvectors from the data tuples.
   (b) Calculate the top $k$ global eigenvectors combining the local ones obtained from different maps by some aggregate (e.g., mean).
   (c) Write the top $k$ global eigenvectors computed to HDFS.
   (d) Write the data fetched to HDFS.

#### 2.2.2 The Second Map-Reduce Phase

The second map-reduce phase will compute and assign outlier scores.

1. The Map Phase:
   (a) The data fetched (in the first phase) will be divided into several chunks (by hadoop) and fed to parallel map instances.
   (b) Input to the map will simply be these data tuples (as key-value pairs).
   (c) Read the global top $k$ eigenvectors computed in the first phase.
   (d) Project the data (local to the map) onto the global top $k$ eigen vectors.
   (e) Compute the normalized error terms as described in the algorithm and assign outlier scores to the individual data tuples.
2. The Reduce Phase: Nothing.