Outlier Detection in Padmini

Sandipan Dey

May 6, 2010

1 Introduction

The Outlier detection algorithm is primarily aimed at detecting outliers in the domain of astronomical objects. We are interested in finding anomalous behavior 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 finding anomalous behavior among these celestial objects, i.e., find outlying objects. Also, we want faster outlier detection in a distributed manner. Hence, we partition the sky into several regions and process the data first locally and in parallel and then combine the processed information to obtain the global outliers. Here we note that finding outliers locally may not be a good choice, since the local outliers may not be global outliers. Instead, we shall use PCA and eigen-analysis and define the global behavior, by the notion of global eigenvectors that are obtained from the global global covariance matrix, obtained by aggregating the local covariance matrices.

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. Even the result of the query can bring in a huge amount of data, so 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 subregions 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.

2 The Algorithm

Our algorithm for distributed outlier detection will be based on PCA [2]. We shall compute distributed PCA on the data using the additively decomposable property (that comes from linearity of expectation) of the covariance matrix [2]. We use 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 eigen vectors are outliers [1].

Algorithm 1 Distributed || Outlier Detection

1: Horizontally partition the data $X_{m\times n}$ into N data chunks $X^i_{m_i\times n}$, $X = \bigcup_{i=1}^{N} X^i$ and assign i^{th} partition to node $\aleph_i$, (where $m = \sum_{i=1}^{N} m_i$).

2: Z-score-normalize the data matrix X_i (so that each column is with 0 mean) at each node $\aleph_i$.

3: Compute the local covariance matrix $C_i = E[X_i^T X_i] = \frac{1}{m_i}\sum_{i=1}^{m_i} X_i^T X_i$ on each node $\aleph_i$.

4: Combine all the local covariance matrices to obtain the global covariance matrix $C_g = E[X^T X] = \frac{1}{m}\sum_{i=1}^{m} X_i^T X_i = \frac{\sum_{i=1}^{N} m_i C_i}{\sum_{i=1}^{N} m_i}$ [2].

5: Compute the set of global eigenvectors by eigen decomposition of the global covariance matrix $C_g = V_g \Lambda_g V_g^T$.

6: Choose top k most dominant eigenvectors ($\hat{V}_g^k$, corresponding to the k largest eigenvalues from the diagonal matrix Λ_g) and send them back to each node $\aleph_i$.

7: Project the local data in each of the nodes $\aleph_i$ onto the top k most dominant global eigenvectors: $\hat{X}^i = X_i.\hat{V}_g^k.\hat{V}_g^{kT}$.

8: For each data tuple X_j^i at node $\aleph_i$, parallely calculate the corresponding error term in projection by $||X_j^i - \hat{X}_j^i||_2$ and assign a normalized outlier score (in the range $[0, 1]$, measuring the degree of outlierness, 1 with the most outlying properties) by $s_j^i = \frac{||X_j^i - \hat{X}_j^i||_2}{\max_j ||X_j^i - \hat{X}_j^i||_2}$.

9: Mark the top k outliers, with the heighest k outlier scores.

3 Implementation in PADMINI using Hadoop

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 covariance matrix and assignment of outlier scores are the ones that are to done locally and are with inherent parallelism, hence should be executed in parallel and independently in maps. The computation of the global covariance matrix by combining the local ones can be done in reduce phase by combining the map outputs as local covariance matrices. Since initially 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.

3.1 The First Map-Reduce Phase

The first map-reduce phase will find the global eigenvectors and store on HDFS. First the meta-data ((ra,dec) coordinates) will be divided into several chunks (by hadoop) and fed to parallel map instances. Input to each of the maps will simply be the meta-data tuples (as key-value pairs). The map phase will first query the VO with (ra, dec) coordinates as arguments and fetch the actual data from the VOs, which is a time-consuming task. Then it normalizes the data and computes the local covariance matrix from the data fetched. Send the local covariance matrices from all maps to the reduce phase, pass the fetched data tuples as well.

In reduce phase, first separate out the local covariance matrices from the data tuples. Combine the local covariance matrices obtained from the maps to find the global covariance matrix. Find the top k global eigenvectors of this global covariance marix and write them to HDFS, along with the normalized data.

3.2 The Second Map-Reduce Phase

The second map-reduce phase will compute and assign outlier scores.
Again, the data fetched (in the first phase) will be divided into several chunks (by hadoop) and fed to parallel map instances. Input to each of these maps will simply be these data tuples (as key-value pairs). Also read the global top k eigenvectors computed in the first phase. Project the data (local to the map) onto the global top k eigen vectors. Compute the normalized error terms as described in the algorithm and assign outlier scores to the individual data tuples.

In the reduce Phase we have nothing to do, i.e., an identity reduce.

3.3 Schematic Diagram

The following diagram gives a detailed visual representation of the map reduce phases involved in the computation of Outlier Detection.

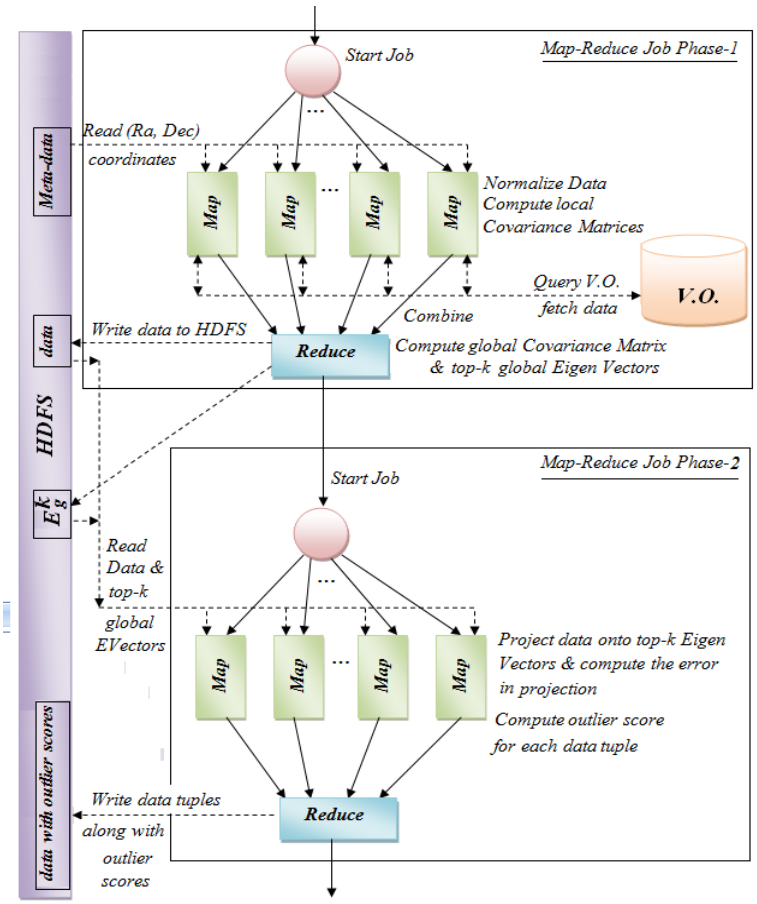


Figure 1: Flow diagram of the Outlier Detection algorithm on Hadoop

4 Experimental Results

The technique for outlier detection we described was PCA based (and hence not distance based). Since the most dominant eigen vectors capture the direction of maximum variance in the dataset, the least dominant ones are expected to reflect the outlier points in the dataset. The following figures elucidate the experimental results.

- We ran the outlier detection algorithm on a 6-attribute 30K tuples dataset

and got the outlier scores as shown in figure 2. The plots also show the variation in attribute values for each tuple along with the outlier scores assigned to each of them. As can be seen from the figure, the tuples having anamoluos attributes are assigned high outlier scores.

- We obtained the scatter-plots taking 2-ttributes at a time from the set of 6 attributes, as shown in figure 3. The tuples with high outlier scores are colored coded with darker colors. As expected, most of the outlier points are assigned high outlier scores (the ones marked by circles).

- Finally we obtained parallelcoords plot using matlab to see the variation along all the 6 attribute values for each of the tuple and grouped (with different colors) according to their scores assigned by the algorithm. As can be seen, the most outlier points obtained the highest scores.

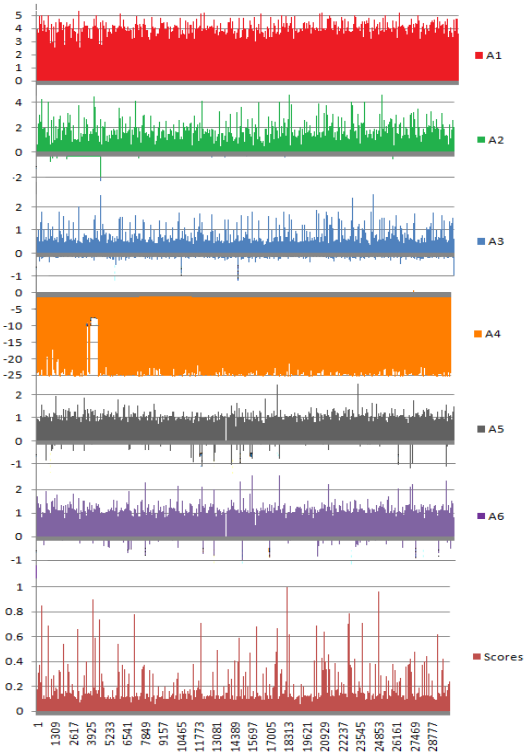


Figure 2: Variation in attribute values and assigned outlier scores for the data tuples

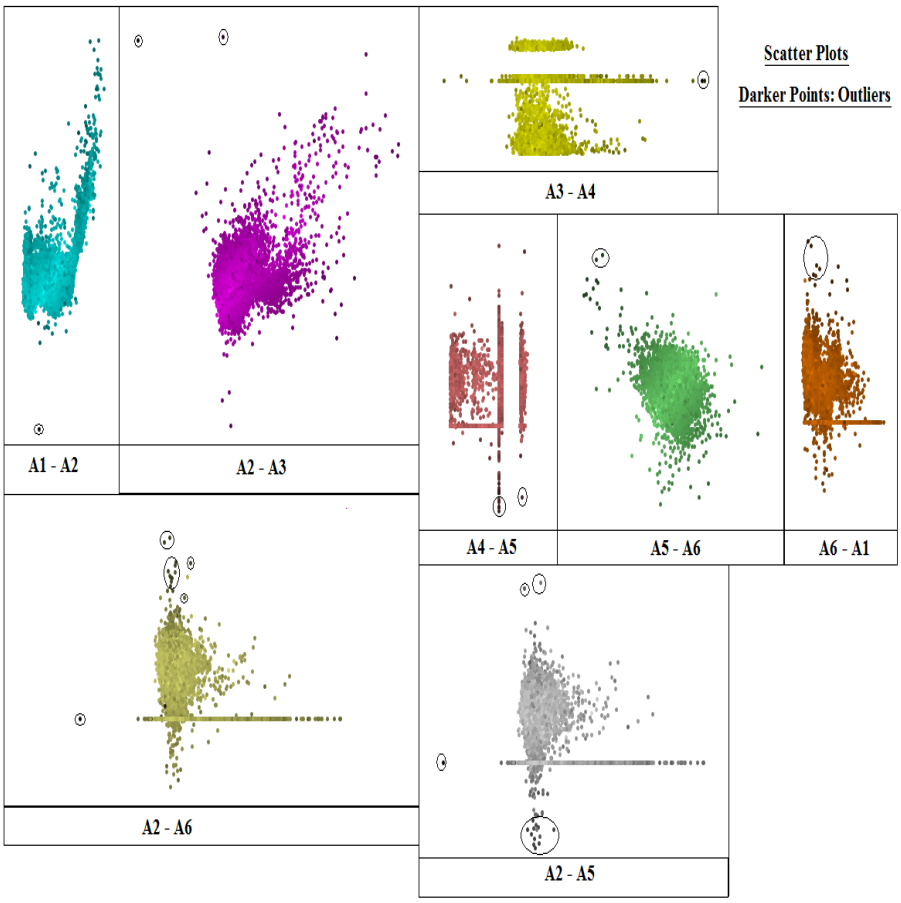
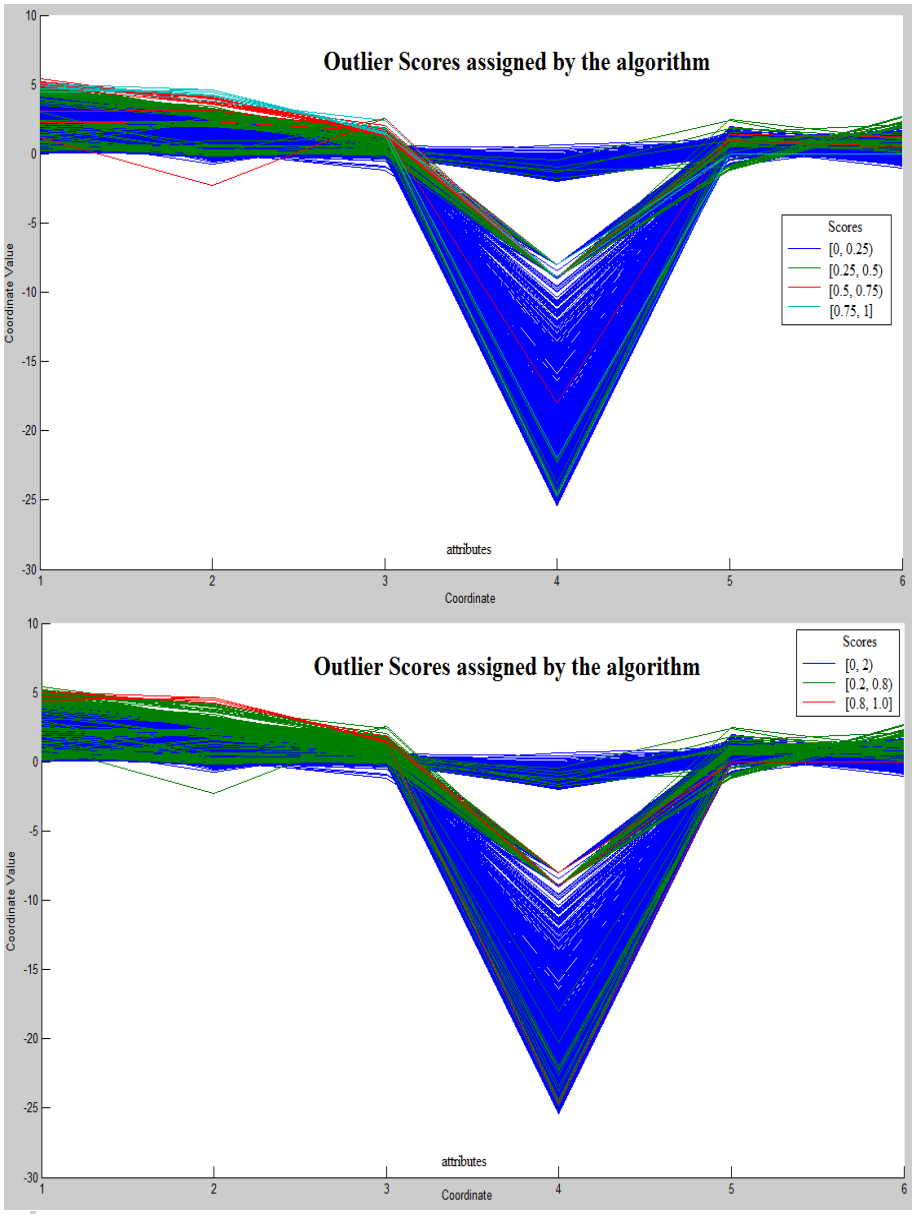


Figure 3: Scatter plots with different 2-attributes combinations and color coded display of outliers


Outlier Scores assigned by the algorithm
Scores
[0, 0.25)
[0.25, 0.5)
[0.5, 0.75)
[0.75, 1]
Coordinate Value
attributes
Coordinate
Outlier Scores assigned by the algorithm
Scores
[0, 2)
[0.2, 0.8)
[0.8, 1.0]
Coordinate Value
attributes
Coordinate

References

[1] Haimonti Dutta, Chris Giannella, Kirk D. Borne, and Hillol Kargupta. Distributed top-k outlier detection from astronomy catalogs using the demac system. In *SDM*, 2007.

[2] Hillol Kargupta, Weiyun Huang, Krishnamoorthy Sivakumar, and Erik Johnson. Distributed clustering using collective principal component analysis. *Knowledge and Information Systems*, 3:2001, 1999.