Distributed Synthetic Minority Oversampling Technique

Rastogi, Avnish Kumar  
HCL Technologies  
India
avnishr@gmail.com

Narang, Nitin  
HCL Technologies  
India
nitin_n@hcl.com

Ajmal, Mohammad  
HCL Technologies  
India
majmaleric@gmail.com

ABSTRACT
Most real world prediction problems have imbalanced data. Here an imbalance in dataset is represented by a mismatch in class representation. An application of classification algorithms on imbalanced data is biased in favor of the majority class and gets further biased in case of high-dimensional data. This class imbalance problem can be reduced by under-sampling of majority data or by oversampling of minority data. Synthetic Minority Oversampling Technique (SMOTE) is one such popular technique proposed by Nitesh Chawala [1] that vastly improves random oversampling. Unfortunately, existing SMOTE [1] implementations are inept to handle big data, demanding need for distributed computing solution. In this paper we investigate challenges for implementing SMOTE in distributed environment and present algorithm and results of our distributed SMOTE [1] implementation. Our implementation is done in Apache Spark using variations of high and low dimensional data as well as large and small datasets. The results are compared to existing monolithic implementation in python SMOTE [1]. We were able to successfully demonstrate a distributed version of Spark SMOTE which generated higher or equal quality artificial samples in comparison to python implementation while preserving spatial distribution.

CCS CONCEPTS
• CCS → Theory of Computation → Design and Analysis of Algorithms → Distributed Algorithms

KEYWORDS
SMOTE; Imbalanced Classification; Metric Trees; M-Trees; Scalable K-means++; Nearest Neighbors; Spark; Map Reduce

1 INTRODUCTION
Classification is one of the most widely studied problems in the data mining and machine learning community. The dataset to extract information should contain all information necessary to learn the relevant concepts pertaining to the underlying generating function. However, the traditional approach has failed to address real world scenarios for classification problems e.g. intrusion detection, spam detection, fraud detection, credit risk etc. These scenarios are inherently a classification problem where one of the events is rare and typically manifests as class imbalance. The issue of class imbalance can be further confounded as these rare events are infrequently present [2-5], and are most likely to be predicted as rare occurrences, undiscovered or ignored, or assumed as noise or outliers. Ironically, the smaller class (minority) is often of more interest and importance, and therefore calls for a stronger case to be recognized. For example, identifying a credit card fraud among online transaction is critical to prevent fraud. Imbalanced nature of the data having disproportionate ratio of minority observations makes classification accuracy unacceptable in identifying fraud cases. Thus, it is important that the model should be able to identify rare occurrences (minority class) in datasets with a higher accuracy than predicting on a total dataset. It is very common to treat the minority sample as noise and ignore them, thus loosing important information from the samples. For example a dataset with imbalance ratio of 1:99 (i.e., for each training example of positive class, there are 99 training examples of negative class). Here, a classifier in its simplest form can classify all instances as negative in order to maximize its classification rule accuracy and hence obtain an accuracy of 99%.

To address the problem of imbalanced classification, a number of solutions have been proposed, which fall into three categories [6-9]. The first is the family of pre-processing techniques aiming to rebalance the training data [10]. The second relates to the algorithmic approach that alters the learning mechanism by taking into account the different class distribution by oversampling minority class, under-sampling majority class or a combination of both [11] while the third comprises of cost-sensitive learning approaches that considers a different cost for the misclassification of each class [12, 13].

In real world of growing data, preprocessing techniques should be modified when the data volume grows and the data as such can’t be processed by a single machine [14]. The data preprocessing scalability issue must be properly addressed to develop new solutions or adapt existing ones for Big Data studies [15-17]. Artificial generation of minority class data using SMOTE (Synthetic Minority Oversampling Technique (SMOTE) [1] is a commonly adopted technique for handling class imbalance. Currently SMOTE is available in non-distributed environment and there are challenges when large minority class data needs to be artificially generated using SMOTE. SMOTE works on generating minority class data by interpolating between several minority data instances that lie together. Distributed SMOTE has its challenges as data distribution across a cluster of machines needs to be managed properly for processing. The spatial arrangement of the samples needs to be preserved for SMOTE to generate data effectively. In absence of proper management, sampling is impacted by uneven distribution of the data across the cluster (nodes), thereby
impacting the effectiveness of the algorithm. In this paper, we present an algorithm, which uses K-Means and M-Trees to synthetically generate the minority data. We have used Apache Spark [18] (version 2.2.0) for implementing the distributed algorithms.

2 RELATED WORK

In imbalance data classification problems [19], sometimes an imbalance present within a single class might be overlooked. This within class imbalance is referred as small disjunct [20, 21]. Under-sampling of the majority class may impact these small disjuncts. Under-sampling of majority class could lead to potential loss of important information [22] and is discouraged. Similarly, in generic oversampling, duplication of data increases the number of samples without enriching the data with new information about the class. Over sampling can increase the likelihood of over fitting [10], as it tends to strengthen all minority clusters disregarding their actual contribution to the problem itself.

Under-sampling methods where the minority class samples are broadcasted to all the nodes of a cluster processing the majority class are limited by the cases where minority samples don’t fit in main memory [23]. An insight into difficulties faced by imbalanced classification and the challenges of generating synthetic data for large imbalanced big data classification problems, suggests need for continuous research and is discussed by Chawla [20]. For running SMOTE on a large data set, the data set needs to be partitioned efficiently. Space partitioning methods consist of tree-based approaches like R-Tree [24], K-D Trees [25], which perform very well when the dimensionality is low. Distance based partitioning divides the points into disjoint sets, where each point in the cell is closer to the pivot of this cell than any other pivot. A scalable K-means++ algorithm can be used to effectively and efficiently partition the sample space in K spaces [26].

M Trees [27] is also an efficient way to search metric spaces for similar items. M-tree can index objects using features compared by distance functions [28]. M-Trees are very efficient and scalable when the number or records and dimensions increase [29, 30]. But all these techniques are designed to run on a single machine and hence become inefficient and impractical when working with big data. A combination of M-Trees and Spill Trees was found to scale well in a distributed sample space and was able to efficiently search similar samples [31]. In this paper we present a hybrid approach to generate artificial minority data, where we cluster similar samples together using K-Means++, build M-Trees around these clustered samples and search for k nearest samples by searching the M-Trees. As demonstrated by sample dataset, our algorithm was found to scale with data.

3 BENCHMARKING PARAMETERS

The problem of imbalanced classification as explained earlier cannot be compared using the accuracy metrics as the minority data is overwhelmed by the majority data present. For example, if a given dataset has a 98:2 distribution, the accuracy of the prediction can easily be 98% accurate by simply predicting all the data as the majority class. In the problem of imbalanced classification, we must consider the complete confusion matrix, which is explained in Table 1. From the confusion matrix we can obtain the classification performance of both, positive and negative classes:

- **True-positive rate (recall)** $TP_{rate} = TP/(TP + FN)$ is the percentage of positive instances correctly classified.
- **True-negative rate (specificity)** $TN_{rate} = TN/(FP + TN)$ is the percentage of negative instances correctly classified.
- **False-positive rate** $FP_{rate} = FP/(FP + TN)$ is the percentage of negative instances misclassified.
- **False-negative rate** $FN_{rate} = FN/(TP + FN)$ is the percentage of positive instances misclassified.

<table>
<thead>
<tr>
<th>Actual Class</th>
<th>Predicted Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>True Positive (TP)</td>
</tr>
<tr>
<td>Negative</td>
<td>False Positive (FP)</td>
</tr>
</tbody>
</table>

Two of the most common metrics that seek at maximizing the joint performance of the classes are both the AUC and Geometric Mean (GM) of the true rates [32]. The former demonstrates the trade-off between the benefits ($TP_{rate}$) and costs ($FP_{rate}$), whereas the latter attempts to maximize the accuracy of each one of the two classes at the same time [sensitivity or recall ($TP_{rate}$) and specificity ($TN_{rate}$)], as depicted below

$$GM = \sqrt{TP_{rate} \cdot TN_{rate}}$$

4 ALGORITHM

SMOTE based oversampling methods applied in distributed environments generally tend to fail [8, 33]. The failure is generally caused by a random partitioning of data generating artificial samples which have no spatial relationships. Our work tries to solve this problem by effectively partitioning and distributing the dataset spatially. In this paper, we present an Apache Spark based SMOTE implementation for big data minority dataset. In the context of the problem that we are addressing, we need to identify nearest neighbors of a point in d dimensional space, so that we can synthetically generate samples (for minority class) between these nearest neighbors. Random distribution of data to different nodes in a distributed cluster may cause points which are nearest to one another to be distributed to different nodes, thereby making it impossible for individual nodes to be aware of these nearest neighbors. Hence,
Distributed Synthetic Minority Oversampling Technique

it is essential that nearest points are grouped together and then
distributed to the different nodes in such a manner that nearest
points are always processed on the same node.

SMOTE generates “pre-determined” number of minority
instances by randomly creating instances between two nearest
neighbors. The most critical question is to efficiently partition
the data. One possibility is to randomly group the data into K
clusters. However, at query time, this would require each query
to run through all the clusters, limiting the throughput. There
are different ways to cluster large datasets. Scalable K Means
[26] is one such fast clustering algorithm. In this paper, we use
spark implementation of scalable K Means++ to cluster our large
dataset into K spaces. Once the dataset is clustered, we build M-
Trees to search the nearest k-neighbors of a point. The M-tree
partitions points by the relative distances (we use Euclidean
distance in our case), and stores these points into fixed-size
nodes, which correspond to constrained regions of the metric
space. The theoretical and application background of M-tree is
thoroughly discussed in [27-30].

To increase the throughput, we build trees in parallel and
searched trees in parallel. Once the clustering of points is done,
we push each cluster of points to individual nodes (partitions in
spark parlance), where these samples are arranged based on
distance in an M-Tree. The Algorithms for partitioning, building
and searching M-Trees are detailed in Algorithm 1, Algorithm 2,
Algorithm 3 and Algorithm 4.

Algorithm 1: Distributed SMOTE

Generate pivots = call k-means api in spark for the dataset
for each point \( p \) \in pivots do
label = closest pivot for each point
end
Reshuffle/Group the points based on the partition id. Push each
partition to a different node/machine/executor
Neighbors to be generated = (total count/minority count)\* upsampling ratio/n
for each partition \( p \) \in partitions do
for point \( p \) \in points in a partition do
Build a M-Tree (algorithm 2)
end
for each point \( p \) \in points in a partition do
knn for a point = Search “n” nearest neighbors
from M-Tree (algorithm 3)
for each of these “n” neighbors do
generate synthetic data (algorithm 4)
end
end

Algorithm 2: buildM-Tree

init: add the first point as a router
for \( p \) \in each point in the partition do
calculate the distances from each router
choose the router with minimum distance
update router radius if the distance > router

Algorithm 3: searchMTree

for \( p \leftarrow \) each point in the partition do
for \( n \) \in neighbors
calculate the distances from each router
choose the router with minimum distance
select top “n” closest points from queryPoint
check whether distance of point from the next router is > max distance of Nearest Neighbor selected
select the router. Repeat above steps.
return top N neighbors.
end

Algorithm 4: SMOTE

for partitions \( \leftarrow \) 0 to num of spark partitions
for \( p \leftarrow \) each point in partition
(repeat this loop for k neighbors)
gap = random number between 0 and 1
synthetic[attribute] = \( p \) [attribute] + gap * difference in values of this attribute between \( p \) and neighbor
done

The choice for “n” took some experimentation where-in we
tried for different single digit values of n and found that the best
results were obtained for n = 5.

5 DATASET

We have selected the ECBDL14 dataset that was used at the
data mining competition of the Evolutionary Computation for
Big Data and Big Learning held on July 14, 2014, in Vancouver
(Canada), under the international conference GECCO-2014 [34].

We took two imbalanced datasets (80:20) from the ECBDL14 data
with the same class distribution and used the larger set for
training and smaller (20%) for validation. ECBDL14 dataset has
large number of features (631) and we used feature reduction
to obtain the most relevant features, reducing to a subset of 30
features from 631 original features. The results from ECBDL14
dataset are used to demonstrate our implementation on Big data.
In addition, we have used three additional datasets from
KEEL[36] viz abalone[37], yeast4[38] and UCI sat image[39] to
demonstrate our algorithm results with the existing
implementations i.e, scikit-learn python implementation of
SMOTE [35]
### Table 2: Distribution of the dataset used

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Attr</th>
<th>Class (maj:min)</th>
<th>#Class (maj:min)</th>
<th>%Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECBDL14</td>
<td>2.89 million</td>
<td>631</td>
<td>1:0</td>
<td>2849275: 48637</td>
</tr>
<tr>
<td>KEEL abalone19</td>
<td>4174 instances</td>
<td>8</td>
<td>1:0</td>
<td>4142:32</td>
</tr>
<tr>
<td>KEEL yeast4</td>
<td>1484 instances</td>
<td>8</td>
<td>1:0</td>
<td>1433:51</td>
</tr>
<tr>
<td>UCI SatImage</td>
<td>36</td>
<td>1:0</td>
<td>5809:626</td>
<td>90.27: 9.72</td>
</tr>
</tbody>
</table>

The infrastructure used for these experiments was a cluster of 4 Centos 6.6 Linux Machines each having 8 cores and 20 GB RAM. The cluster was configured with Spark 2.2.0 and Hadoop HDFS was used to store the input and output data.

### 6 ANALYSIS

We synthetically generated minority class data using our algorithms. This data was combined with the 80% data set, which was set aside for training.

### Table 3: Distribution of the dataset used

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed Random Forest</td>
<td>Number of Tress: 50</td>
</tr>
<tr>
<td>(h2o)</td>
<td>Maximum Tree Depth: 20</td>
</tr>
<tr>
<td></td>
<td>NBins: 20</td>
</tr>
<tr>
<td></td>
<td>Sample Rat: 0.63</td>
</tr>
<tr>
<td>Default Random Forest</td>
<td>Number of Tress: 10</td>
</tr>
</tbody>
</table>

We used Distributed Random Forest (DRF) classification algorithms from h2o [40] to evaluate model accuracy on ECBDL dataset and tested against the 20% evaluation dataset. For the abalone, yeast4 and satimage datasets we used scikit python implementation of Random Forest. Model parameters are listed in Table 3.

To benchmarked accuracy of synthetic data from our Spark implementation, we generated minority data from scikit-learn python implementation of SMOTE [38]. The algorithm’s performance was tuned using different input parameters and it was found that the best results were achieved when the number of clusters was set to 4 and leaf size of the M-Tree set to 50 for small datasets (< 10000) and the number of clusters set to 8 or 16 and the leaf size to 1000 with larger dataset. Table 4 and Table 5 list the AUC, Recall, GM and Confusion Matrix obtained from Random Forest (Python Implementation for datasets Abalone, yeast4 and satimage) and Distributed Random Forest implementation by h2o for ECBDL dataset. The minority data is upsamped to 100% from existing distribution.

We observed Spark Smote based implementation matched or exceeded in comparison to Python SMOTE implementation for the selected datasets as listed in Table 4, Table 5, Table 6 and Table 7. In Table 7, we see that distributed Spark SMOTE implementation does well even when the dataset grows big. Applying SMOTE on large dataset i.e, ECBDL dataset did not improve the prediction of minority classes when compared with scenario where SMOTE was not applied. This may imply that this specific dataset might not benefit from SMOTE, as SMOTE does not guarantee improvement in prediction of minority class. However, Spark based distributed SMOTE performed better on dataset when compared with sklearn Python implementation, validating our implementation.

### Table 4: Performance Comparison (Abalone Dataset)

<table>
<thead>
<tr>
<th>Technique</th>
<th>AUC</th>
<th>Recall</th>
<th>GM</th>
<th>Confusion Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Sampling</td>
<td>0.50</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Python SMOTE</td>
<td>0.78</td>
<td>0.60</td>
<td>0.76</td>
<td>3 2</td>
</tr>
<tr>
<td>Spark SMOTE</td>
<td>0.85</td>
<td>0.80</td>
<td>0.84</td>
<td>4 1</td>
</tr>
</tbody>
</table>

The results validate that this implementation of distributing data and using M-Trees to identify nearest neighbor is able to preserve spatial arrangement of samples and hence is able to facilitate generation of high quality artificial samples in parallel achieving scale with volume.
Distributed Synthetic Minority Oversampling Technique

Table 5: Performance Comparison (Yeast4 Dataset)

<table>
<thead>
<tr>
<th>Technique</th>
<th>AUC</th>
<th>Recall</th>
<th>GM</th>
<th>Confusion Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>No sampling</td>
<td>0.57</td>
<td>0.14</td>
<td>0.37</td>
<td>1 6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 290</td>
</tr>
<tr>
<td>Python SMOTE</td>
<td>0.90</td>
<td>0.85</td>
<td>0.90</td>
<td>6 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14 276</td>
</tr>
<tr>
<td>Spark Smote</td>
<td>0.85</td>
<td>0.72</td>
<td>0.83</td>
<td>5 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11 279</td>
</tr>
</tbody>
</table>

Table 6: Performance Comparison (UCISat Dataset)

<table>
<thead>
<tr>
<th>Technique</th>
<th>AUC</th>
<th>Recall</th>
<th>GM</th>
<th>Confusion Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>No sampling</td>
<td>0.78</td>
<td>0.65</td>
<td>0.80</td>
<td>125 86</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>43 1746</td>
</tr>
<tr>
<td>Python SMOTE</td>
<td>0.78</td>
<td>0.75</td>
<td>0.76</td>
<td>160 51</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>113 1676</td>
</tr>
<tr>
<td>Spark Smote</td>
<td>0.78</td>
<td>0.83</td>
<td>0.87</td>
<td>175 36</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>169 1620</td>
</tr>
</tbody>
</table>

Table 7: Performance Comparison (ECBDL Dataset)

<table>
<thead>
<tr>
<th>Technique</th>
<th>AUC</th>
<th>Recall</th>
<th>GM</th>
<th>Confusion Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Sampling</td>
<td>0.90</td>
<td>0.83</td>
<td>0.83</td>
<td>12124 2447</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>145949 708854</td>
</tr>
<tr>
<td>Python SMOTE</td>
<td>0.79</td>
<td>0.78</td>
<td>0.78</td>
<td>11442 3129</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>184402 670401</td>
</tr>
<tr>
<td>Spark Smote</td>
<td>0.89</td>
<td>0.81</td>
<td>0.81</td>
<td>11746 2823</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>166637 688166</td>
</tr>
</tbody>
</table>

7 OBSERVATIONS

For greater accuracy in classification problems, it is imperative that we take entire dataset and should not limit analysis on a sample set. With our proposed algorithm, we can generate quality minority samples thereby improving the prediction of a classification problem. The traditional SMOTE implementation can’t scale with volume of data and has limited usage in big datasets, which is becoming a norm today. Following up on the research detailed in [8, 33], we expanded SMOTE for a Spark based distributed implementation and observed that the Spark based Implementation scored better on various benchmarking parameters. Further research needs to be done to complete SMOTE in totality but this implementation can be used as a baseline for further research.

8 SUMMARY

We have carried out an implementation and benchmarking of SMOTE implementation using Apache Spark Framework based on distributed K-Means and M-Trees. Further scope of work includes SMOTE implementation using other algorithms like Borderline SMOTE 1, 2 [41] and SVM-SMOTE. SMOTE in big data depends heavily on the clustering algorithm used. The more effective the clustering algorithm is, the better is the performance and accuracy of the results. Spill trees and hybrid trees are another area which can be used to improve the clustering and nearest neighbor algorithm.

ACKNOWLEDGMENTS

Authors would like to acknowledge the efforts of Chandan Parikh, Abhishek Mishra, Pratyush Sharma who helped the authors in this work. We also acknowledge the support from HCL Technologies for supporting this research and providing the necessary infrastructure.

REFERENCES

[6] He H, Garcia EA. 2009. Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering 21(9), 1263–1284


[34] ECDL’14 dataset. http://cruncher.ncl.ac.uk/bdc omp/.


[38] Yeast dataset http://sci2s.ugr.es/keel/dataset.php?cod=133


[40] H2o https://www.h2o.ai/