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
majmalcric@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 Tress; 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 property. 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 1: Confusion Matrix for a two-class problem

Actual Class	Predicted Class	
	Positive	**Negative**
Positive	True Positive (TP)	False Negative (FN)
Negative	False Positive (FP)	True Negative (TN)

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

$$GM = \sqrt{TPrate * TNrate}$$

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,

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 ← 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 ← partitions do
        for point ← points in a partition do
                Build a M-Tree (algorithm 2)
        end
        for each point ← 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
end
```

Algorithm 2: buildM-Tree

```
init:   add the first point as a router
for p ← 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
                covering radius
        add the point as leaf node in the router
        increment leaf count
        if leafCount > maxLeafSize:
                split the router node into two
                calculate two pivots (based on the min and
                max distance from points in the leaf)
                upgrade them as routers
                split the leafs under router Node by distance
end
```

Algorithm 3: searchMTree

```
for p ← each point in the partition do
        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.
done
```

Algorithm 4: SMOTE

```
for partitions ← 0 to num of spark partitions
        for p ← 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
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

Dataset	#Attr	Class (maj:min)	#Class (maj:min)	%Class
ECBDL14 2.89 million	631	1:0	2849275: 48637	98.3: 1.7
KEEL abalone19 4174 instances	8	1:0	4142:32	99.23: 0.77
KEEL yeast4 1484 instances	8	1:0	1433:51	96.56: 3.44
UCI SatImage	36	1:0	5809:626	90.27: 9.72

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

Algorithm	Parameters
Distributed Random Forest (h2o)	Number of Tress : 50 Maximum Tree Depth 20 NBins: 20 Sample Rat : 0.63
Default Random Forest	Number of Tress : 10

We used Distributed Random Forest (DRF) classification algorithms from h2o [40] to evaluate model accuracy on ECBDL dataset and tested against the 20% evaluation dataser. 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 upsampled 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)

Technique	AUC	Recall	GM	Confusion Matrix
No Sampling	0.50	0	0	0 5 0 830
Python SMOTE	0.78	0.60	0.76	3 2 26 804
Spark SMOTE	0.85	0.80	0.84	4 1 89 741

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.

Table 5: Performance Comparison (Yeast4 Dataset)

Technique	AUC	Recall	GM	Confusion Matrix
No sampling	0.57	0.14	0.37	1 6 0 290
Python SMOTE	0.90	0.85	0.90	6 1 14 276
Spark Smote	0.85	0.72	0.83	5 2 11 279

Table 6: Performance Comparison (UCISat Dataset)

Technique	AUC	Recall	GM	Confusion Matrix
No sampling	0.78	0.65	0.80	125 86 43 1746
Python SMOTE	0.78	0.75	0.76	160 51 113 1676
Spark Smote	0.78	0.83	0.87	175 36 169 1620

Table 7: Performance Comparison (ECBDL Dataset)

Technique	AUC	Recall	GM	Confusion Matrix
No Sampling	0.90	0.83	0.83	12124 2447 145949 708854
Python SMOTE	0.79	0.78	0.78	11442 3129 184402 670401
Spark Smote	0.89	0.81	0.81	11746 2823 166637 688166

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 Parihar, 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

[1] Chawla N.V, et al 2002 Synthetic Minority Over-Sampling Technique. Journal of Artificial Intelligence Research. 16, 321–357
[2] Yu H, Hong S, Yang X, Ni J, Dan Y, Qin B. 2013. Recognition of multiple imbalanced cancer types based on DNA microarray data using ensemble classifiers. BioMed Research International: 1-13
[3] Chen Y-S . 2016. An empirical study of a hybrid imbalanced-class DT-RST classification procedure to elucidate therapeutic effects in uremia patients. Medical and Biological Engineering & Computing, 54(6), 983–1001
[4] Haixiang G, Yijing L, Yanan L, Xiao L, Jinling L. 2016. BPSOAdaboost-KNN ensemble learning algorithm formulti-class imbalanced data classification. Eng Appl Artif Intell, 49, 176–193
[5] Elhag S, Fernández A, Bawakid A, Alshomrani S, Herrera F. 2015. On the combination of genetic fuzzy systems and pairwise learning for improving detection rates on intrusion detection systems. Expert Syst Appl 42(1), 193–202
[6] He H, García EA. 2009. Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering 21(9), 1263–1284
[7] Sun Y, Wong, Andrew and Mohamed Kamel. 2009. Classification of Imbalanced Data: A Review. International Journal of Pattern Recognition and Artificial Intelligence 23,Issue 04, 687-719
[8] Fernández, A., Chawla, Nitesh, García, S., Palade, V., Herrera, F. 2017. An insight into classification with imbalanced data: empirical results and current trends on using data intrinsic characteristics. Complex and Intelligent Systems 250(20), 113–141
[9] Krawczyk, B. 2016. Learning from imbalanced data: open challenges and future directions. Progress in Artificial Intelligence 5(4), 221–232
[10] Batista GEAPA, Prati RC, Monard MC. 2004. A study of the behavior of several methods for balancing machine learning training data. SIGKDD Explorations 6, 1, 20–29
[11] Ramentol E, Vluymans S, Verbiest N, Caballero Y, Bello R, Cornelis C, Herrera F. 2015.IFROWANN: imbalanced fuzzy-rough ordered weighted average nearest neighbor classification. IEEE Trans Fuzzy Syst 23(5), 1622–1637
[12] Domingos P. 1999. Metacost: A general method for making classifiers cost-sensitive, Proceedings of the 5th international conference on knowledge discovery and data mining (KDD'99, 155–164
[13] López V, Fernández A, Moreno-Torres JG, Herrera F. 2012. Analysis of preprocessing vs. cost-sensitive learning for imbalanced classification. Open problems on intrinsic data characteristics. Expert Syst Appl 39(7), 6585–6608
[14] D. Laney. 2001. 3D data management: Controlling data volume, velocity, and variety, Tech. rep., META Group
[15] Kambatla K, Kollias G, Kumar V, Grama A. 2014. Trends in big data analytics. J Parallel Distrib Comput 74(7), 2561–2573

[16] Zikopoulos PC, Eaton C, deRoos D, Deutsch T, Lapis G. 2011. Understanding big data—analytics for enterprise class hadoop and streaming data, 1st edn. McGraw-Hill Osborne Media, New York
[17] Wu X, ZhuX, WuG-Q, DingW. 2014. Data mining with BigData. IEEE Trans Knowl Data Eng 26(1), 97–107
[18] Apache Spark https://spark.apache.org/docs/latest/index.html
[19] Japkowicz, N. 2000. The Class Imbalance Problem: Significance and Strategies. In Proceedings of the 2000 International Conference on Artificial Intelligence (IC-AI'2000), 141-117
[20] Fernández, A., Río, Sara del, Chawla, Nitesh V. Francisco Herrera1. 2017. An insight into imbalanced Big Data classification: outcomes and challenges Complex Intell. Syst. 3, 105–120
[21] Prati, R.C., G.E. Batista, and M.C. Monard. 2004. Learning with class skews and small disjuncts, Advances in Artificial Intelligence, 296-306.
[22] García, S. and F. Herrera. 2009. Evolutionary undersampling for classification with imbalanced datasets: Proposals and taxonomy. Evolutionary Computation, 17(3, 275-306
[23] Triguero, I., Galar, M., Sola, HB., Herrera, F. 2016. Evolutionary Undersampling for Extremely Imbalanced Big Data Classification under Apache Spark. IEEE Congress on Evolutionary Computation. 641-647
[24] Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. SIGMOD Conference, 47-57
[25] Jon Louis Bentley. 1990. K-d trees for semidynamic point sets. Symposium on Computational Geometry,
[26] Bahmani, B., Moseley, B., Vattani, A., Ravi Kumar, Vassilvitskii, S. 2012. Scalable K means ++. Journal Proceedings of the VLDB Endowment. Vol 5 Issue 7, 622-633
[27] P. Ciaccia , M. Patella , P. Zezula. 1997. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. VLDB '97 Proceedings of the 23rd International Conference on Very Large Data Bases, 426-435
[28] P. Ciaccia , M. Patella , F. Rabitti , P. Zezula. 1997. Indexing Metric Spaces with M-tree. Proceedings Qunito Convegno Nazionale Sebd, 67-86
[29] Moore, A W. 2000. The Anchors Hierarchy: Using the Triangle Inequaliity to Survive High Dimensional Data. Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI2000)
[30] P. Zezula, P. Ciaccia, and F. Rabitti. 1996. M-tree: A dynamic index for similarity queries in multimedia databases. Technical Report 7, HERMES ESPRIT LTR Projects
[31] Ting, L., Charles, R., Henry A.R., 2007. Clustering Billions of Images with Large Scale Nearest Neighbor Search. IEEE Workshop on Applications of Computer Vision (WACV'07),
[32] Huang J, Ling CX . 2005. Using AUC and accuracy in evaluating learning algorithms. IEEE Trans Knowl Data Eng 17(3), 299–310
[33] Krawczyk, Bartosz . 2016. Learning from imbalanced data:Open challenges and future directions. Progress in Artificial Intelligence. Vol 5, Issue 4, 221 -232
[34] ECBDL'14 dataset. http://cruncher.ncl.ac.uk/bdcomp/
[35] ScikitLearn. http://contrib.scikit-learn.org/imbalanced-learn/stable/generated/imblearn.over_sampling.SMOTE.html
[36] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera. 2011. KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework. Journal of Multiple-Valued Logic and Soft Computing 17:2-3, 255-287
[37] Abalone dataset http://sci2s.ugr.es/keel/dataset.php?cod=115
[38] Yeast dataset http://sci2s.ugr.es/keel/dataset.php?cod=133
[39] UCI Satellite Image Dataset https://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite)
[40] H2o https://www.h2o.ai/
[41] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. 2005. Borderline SMOTE: A New Over Sampling Method in Imbalanced Data Sets Learning. International Conference on Intelligent Computing ICIC, 878-887