ISSN (ONLINE): 2395-695X
ISSN (PRINT): 2395-695X

International Journal of Advanced Research in Basic Engineering Sciences and Technology (IJARBEST)
Vol.3, Special Issue.24, March 2017

ENHANCED k-NEAREST NEIGHBOUR CLASSIFIER FOR BIG DATA

M.I.Thufail Ahamed, *PG Scholar, Department of CSE, Kongu Engineering College, Tamilnadu*
Dr.R.Thangarajan, *Professor, Department of CSE, Kongu Engineering College ,Tamilnadu*

ABSTRACT

Big data analytics is the process of examining large data sets to uncover hidden patterns, unknown correlations, market trends, customer preferences and other useful business information. The analytical findings can lead to more effective marketing, new revenue opportunities, better customer service, improved operational efficiency, competitive advantages over rival organizations and other business benefits.The k-Nearest Neighbors classifier is a simple yet effective widely renowned method in data mining. The actual application of this model in the big data domain is not feasible due to time and memory restrictions. Several distributed alternatives based on MapReduce have been proposed to enable this method to handle large-scale data. However, their performance can be further improved with new designs that fit with newly arising technologies. In this work we provide a new solution to perform an exact k-nearest neighbor classification based on Spark. We take advantage of its in-memory operations to classify big amounts of unseen cases against a big training dataset.
Key words: K-nearest neighbors, Big data,Apache Hadoop, Apache Spark

I.INTRODUCTION

Big data is a term for data sets that are so large or complex that traditional data processing applications are inadequate to deal with them. Challenges include analysis, capture, data curation, search, sharing, storage, transfer, visualization, querying, updating and information privacy. The term "big data" often refers simply to the use of predictive analytics, user behavior analytics, or certain other advanced data analytics methods that extract value from data, and seldom to a particular size of data set[1].

The primary goal of big data analytics is to help companies make more informed business decisions by enabling data scientists, predictive modelers and other analytics professionals to analyze large volumes of transaction data, as well as other forms of data that may be untapped by conventional business intelligence (BI) programs[2]. That could include Web server logs and Internet click stream data, social media content and social network activity reports, text from customer emails and survey responses, mobile-phone call detail records and machine data captured by sensors connected to the Internet of Things.

II.RELATED WORK

C. Lynch (2008) in his work demonstrated how do the data grow[1].The author describes the various sources of data and the speed of data generated and the variety of data generated how they accelerate the growth of data.

M. Minelli , M. Chambers , A. Dhiraj (2013) proposed various techniques[3] that, deals with using big data analytics in analyzing the current trends in business techniques to improve the decision making process and choosing the best stratergy to improve the productivity and growth of the business..

Y. Bu, B. Howe, M. Balazinska, M.D. Ernst,(2010) , demonstrated the processing of big data using hadoop framework[3] where big data sets can be processed using massive parallelism in the hadoop environment using distributed systems

S. Ghemawat , H. Gobioff, S.-T. Leung(2003) explained in their work the parallel processing of google file systems(5).The google file system is

ISSN (ONLINE): 2395-695X
ISSN (PRINT): 2395-695X

International Journal of Advanced Research in Basic Engineering Sciences and Technology (IJARBEST)
Vol.3, Special Issue.24, March 2017

enhanced for Google's core data storage.It has multiple nodes that are divided into master node and slave nodes each chunk is divided into 64 bit label by master node at the time of creation and logical mapping of file and it is replicated several times throughout the network minimum three and are processed in hadoop environment.

A. Spark (2015) proposed a new concept for fast cluster computing using a new framework known as Spark framework[6].Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance which makes Spark an ideal choice for fast cluster computing.

C. Zhang, F. Li, J. Jestes,(2012) demonstrated in their work about efficient parallel kNN joins for large dataset[9] where the kNN algorithm which is simple yet an effective classification algorithm is implemented in a parallel approach which is executed by dividing the dataset into small subset of multiple datasets and kNN algorithm is applied over the multiple sub datasets and the final result is aggregated by mapreduce programming model.

Arnaiz-González, J.F. Díez-Pastor, J.J. Rodríguez, C. García-Osorio,(2016) proposed in their work to reduced the time complexity in processing big datasets to make them linear. two new algorithms with linear complexity for instance selection purposes are presented. The algorithm use locality-sensitive hashing to find similarities between instances. While the complexity of conventional methods (usually quadratic, or log-linear, O) means that they are unable to process large-sized data sets, the new proposal shows competitive results in terms of accuracy. Even more remarkably, it shortens execution time, as the proposal manages to reduce complexity and make it linear with respect to the data set size.

From the literature, the main disadvantages observed are time complexity .due to loading bigdatasets from hdfs.

III.PROPOSED WORK

III.PROBLEM STATEMENT

k-Nearest Neighbors algorithm is a method used for classification, The input consists of the *k* closest training examples.The output is a class membership. An object is classified by a majority vote of its neighbors(by euclidean distance), with the object being assigned to the class most common among its *k* nearest neighbors[9].

Let *TR* be a training dataset and *TS* a test set, they are formed by a determined number **n** and **t** of samples, respectively. Each sample **x** *p* is a tuple (**x** *p*1 , **x** *p*2 , ..., **x** *pD* , ω) , where, x *pf* is the value of the *f* -th feature of the *p* -th sample. This sample belongs to a class ω, given by x ω *p* , and a *D* -dimensional space. For the *TR* set the class ω is known, while it is unknown for *TS* . For each sample x *test* included in the *TS* set, the kNN algorithm searches the *k* clos- est samples in the *TR* set. Thus, the kNN calculates the distances between x *test* and all the samples of *TR* . The Euclidean distance is the most widely-used measure for this purpose. The training samples are ranked in ascending order according to the computed distance, taking the *k* nearest samples (neigh 1 , neigh 2 , ..., neigh *k*). Then, they are used to compute the most predominant class label.

SYSTEM ELUCIDATION

The enormous amount of data is generated day to day. To manage and analyse those big data is the challenging task. Thus to handle big data an efficient and scalable algorithms are needed.In this work we provide a new solution to perform an exact k-nearest neighbor classification based on Spark.

As a MapReduce model, this divides the computation into two main phases: the map and the reduce operations. The map phase splits the training data and calculates for each chunk the distances and the corresponding classes of the *k* nearest neighbors for every test sample. The reduce stage aggregates the distances of the *k* nearest neighbors from each map and makes a definitive list of *k* nearest neighbours. Ultimately, it conducts the usual majority voting procedure of the kNN algorithm to predict the resulting class. Map and reduce functions are now defined.

ISSN (ONLINE): 2395-695X
ISSN (PRINT): 2395-695X

International Journal of Advanced Research in Basic Engineering Sciences and Technology (IJARBEST)
Vol.3, Special Issue.24, March 2017

.

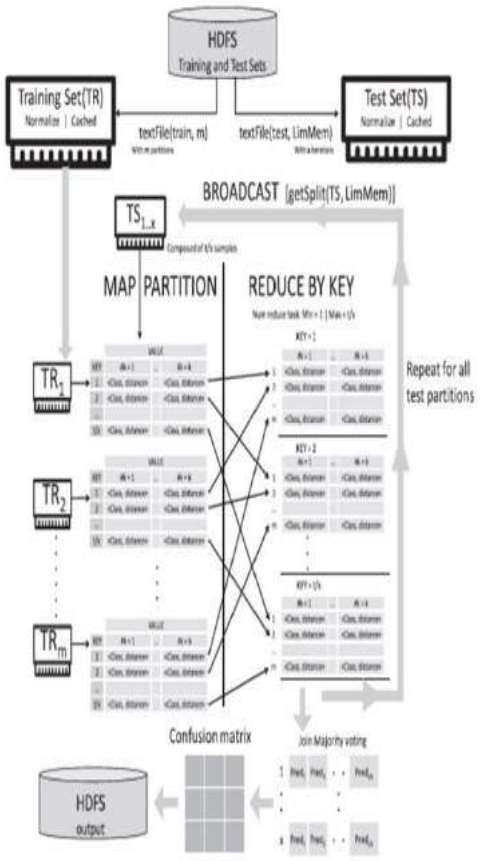

Figure 3.1 k-NN using Spark

Let us assume that the training set *TR* and the corresponding subset of test samples *TS i* have been previously read from HDFS as RDD objects. Hence, the training dataset *TR* has already been split into a user-defined number *m* of disjoint subsets when it was read. Each map task (*Map* 1 , *Map* 2 , ..., *Map m*) tackles a subset *TR j* , where 1 $\leq j \leq m$, with the samples of each chunk in which the training set file is divided. Therefore, each map approximately processes a similar number of training instances. To obtain an exact implementation of kNN, the input test set *TS i* is not split together with the training set, but it is read in each map in order to compare every test sample against the whole training set. It implies that both *TS i* and *TR j* are supposed to fit altogether in memory.

Every map *j* will constitute a class-distance vector *CD t, j* of pairs < *class, distance* > of dimension *k* for each test sample *t* in *TS i* . To do so, Instruction 2 computes for each test sample the class and the distance to its *k* nearest neighbors. To accelerate the posterior actualization of the nearest neighbors in the reducers, every vector *CD t, j* is sorted in ascending order regarding the distance to the test sample, so that, *Dist* (*neigh* 1) < *Dist* (*neigh* 2) < .... < *Dist* (*neigh k*).

The pseudocode of map function is shown in Algorithm 1.

I **Algorithm 1** Map function
Require: T R j T S i ; k
1: **for** t = 0 **to** size (T S i) **do**
2: CD t,j ← Compute kNN (T R j , T S i (x) , k)
3: result j ← (< key : t, value : CD t,j >)
4: EMIT(result j)
5: **end for**

The reduce phase consists of collecting, from the tentative *k* nearest neighbors provided by the maps, the closest ones for the examples contained in *TS i*. After the map phase, all the elements with the same key have been grouped. A reducer is run over a list(*CD t* , 0 , *CD t* , 1 , .., *CD t, m*) and determines the *k* nearest neighbors of this test example *t* . This function will process every element of such list one after another. Instructions 2 to 7 update a resulting list *results reducer* with the *k* neighbors. Since the vectors coming from the maps are or- dered according to the distance, the update process becomes faster. This consists of merging two sorted lists up to get *k* values, so that, the complexity in the worst case is O(k).

Therefore, this function compares every distance value of each of the neighbors one by one, starting with the closest neighbor. If the distance is lesser than the current value, the class and the distance of this position is updated with the corresponding values, otherwise we proceed with the following value.

Algorithm 2 provides the details of the reduce operation.

ISSN (ONLINE): 2395-695X
ISSN (PRINT): 2395-695X

International Journal of Advanced Research in Basic Engineering Sciences and Technology (IJARBEST)
Vol.3, Special Issue.24, March 2017

I.	**Algorithm 2** Reduce by key operation **Require:** *result key* , *k*
II.	1: cont=0
III.	2: **for** *i* = 0 **to** *k* **do**
IV.	3: **if** *r esult key* (*cont*) *.Dist* < *r esult reducer* (*i*) *.Dist* **then**
V.	4: *result reducer* (*i*) = *result key* (*cont*)
VI.	5: cont++
VII.	6: **end if**
VIII.	7: **end for**

As input, we receive the path in the HDFS for both training and test data as well as the number of maps *m* and reducers *r* . We also dispose of the number of neighbors *k* and the memory allowance for each map.

First, we create an RDD object with the training set *TR* formed by *m* blocks (Instruction 1). The test set *TS* is also read as an RDD without specifying a number of partitions. As this is read, we establish the key of every single test instance according to its position in the dataset (Instruction 2, function *zipWithIndex()* in Spark).

Since we will use Euclidean distance to compute the similarity between instances, normalizing both datasets becomes a manda- tory task. Thus, Instructions 3 and 4 both perform a parallel oper- ation to normalize the data into the range [0,1]. Both datasets are also cached for future reuse. Instruction 5 calculates the minimum number of iterations # *Iter*. Instruction 6 will perform the partitioning of the test dataset by using the function RangePartitioner.

Next, the algorithm enters into a loop in which we classify sub- sets of the test set (Instructions 7-12). Instruction 7 firstly gets the split corresponding to the current iteration. We use the transfor- mation *filterByRange(lowKey, maxKey)* to efficiently take the corre- sponding subset. This function takes advantage of the split per- formed in Instruction 6, to only scan the matching elements. Then, we broadcast this subset *TS i* into the main memory of all the com- puting nodes involved. The *broadcast* function of Spark allows us to keep a read-only variable cached on each machine rather than copying it with the tasks. After that, the main map phase starts in Instruction 9. As stated before, the *mapPartition* function computes the kNN for each par- titions of *TR j* and *TS i* and emits a pair RDD with key equals to the number of instance and value equals to a list of *class-distance* . The reduce phase joins the results of each map grouping by key (Instruction 9). As a result, we obtain the *k* neighbors with the small- est distance and their classes for each test input in *TS i* .

More de- tails can be found in the previous section. The last step in this loop collects the right and predicted classes storing them as an array in every iteration (Instruction 11).Finally, when the loop is done, Instruction 13 computes the resulting confusion matrix and outputs the desired performance measures. that we will have to perform to manage the input data. To do so, it will use the size of every chunk of the training dataset, the size of the test set and the memory allowance for each map. The output of the combine function is then passed as the input to the reduce function. In reduce function, the sum of all the samples is performed and the computation for the total number of samples is done for the cluster. Therefore, we can get the new centers which are used for next iteration. The pseudocode for reduce function is given in the algorithm 3

Algorithm 3 kNN-IS **Require:** *T R* ; *T S*; *k* ; # *Maps* ; # *Reduces* ; # *MemAl l ow*
1: *T R −RDD raw* ← textFile(*T R* , # *Maps*)
2: *T S −RDD raw* ← textFile(*T S*).zipWithIndex()
3: *T R −RDD* ← *T R −RDD raw* .map(normalize).cache
4: *T S −RDD* ← *T S −RDD raw* .map(normalize).cache
5: # *Iter* ← calIter(*T R −RDD* .weight(), *T S −RDD* .weight, *MemAl l ow*)
6: *T S −RDD* .RangePartitioner(# *Iter*)
7: **for** *i* = 0 **to** # *Iter* **do**
8: *T S i* ← broadcast(*T S −RDD* .getSplit(i))
9: resultKNN ← *T R −RDD* .mapPartition(*T R j* → kNN(*T R j* , *T S i* , *k*))
10: result ← *r esultKNN.r educeByKey* (*combineResult*, #Reduces) .collect

ISSN (ONLINE): 2395-695X
ISSN (PRINT): 2395-695X

International Journal of Advanced Research in Basic Engineering Sciences and Technology (IJARBEST)
Vol.3, Special Issue.24, March 2017

11: right-predictedClasses[i] ← calculateRightPredicted(result)

IV RESULTS AND DISCUSSION

The performance for proposed methods can be evaluated by using certain metrics. Scalability is used for evaluating the performance of the algorithm.

.

ACCURACY

Represents the number of correct classifications against the total number of classified instances. This is calculated from a resulting confusion matrix, dividing the sum of the diagonal elements between the total of the elements of the confusion matrix. This is the most commonly used metric for assessing the performance of classifiers for years in standard classification.

RUNTIME

The total runtime for the parallel approach includes reading and distributing all the data, in addition to calculating *k* nearest neighbors and majority vote.

SPEEDUP

Proves the efficiency of a parallel algorithm comparing against the sequential version of the algorithm. Thus, it measures the relation between the runtime of sequential and parallel versions. In a fully parallelism environment, the maximum theoretical speed up would be the same as the number of used cores, according to the Amdahl's Law.

Speedup = base _ line /paral l el _ time

where base_line is the runtime spent with the sequential version and parallel _time is the total runtime achieved with its improved version.

The k-NN algorithm implemented in spark produces ouput almost same as when implemented in sequential execution and the time complexity has been reduced to a great extent due to the in-memory operations in spark frame work.The accuracy calculated using confusion matrix is 88.76%.And the value of k influences the accuracy of the classification.

V CONCLUSION AND FUTURE WORK

CONCLUSION

This work presented a Iterative MapReduce solution for the k-Nearest Neighbors algorithm based on Spark. It is denominated as kNN-IS. The proposed scheme is an exact model of the kNN algorithm that we have enabled to apply with large-datasets. Thus, kNN-IS obtains the same accuracy as kNN. However, the kNN algorithm has two main issues when dealing with large-scale data: Runtime and Memory consumption. The use of Apache Spark has provided us with a simple, transparent and efficient environment to parallelize the kNN algorithm as an iterative MapReduce process.

FUTURE WORK

To tackle big datasets that contain missing values by using kNN-IS to impute them, and datasets with a very large number of features by using multi-view approaches[10]. Also to extend the use of kNN-IS to instance selection techniques for big data, where it reports good results[11]. And also to extend the application of the presented kNN-IS approach to a big data semi-supervised learning context[12] as well as to implement adaptive k-NN which finds out the optimal k value for the training set thereby improving the performance of k-NN algorithm

REFERENCES

[1] . Lynch, Big data: how do your data grow? Nature 455 (7209) (2008) 28–29.

[2] M. Minelli , M. Chambers , A. Dhiraj , Big Data, Big Analytics: Emerging Business Intelligence and Analytic Trends for Today's Businesses (Wiley CIO), 1st edition, Wiley Publishing, 2013 .

[3] Y. Bu, B. Howe, M. Balazinska, M.D. Ernst, Hadoop: efficient iterative data processing on large clusters, Proc. VLDB Endow. 3 (1-2) (2010) 285–296, doi: 10. 14778/1920841.1920881 .

[4] K. Grolinger, M. Hayes, W. Higashino, A. L'Heureux, D. Allison, M. Capretz, Challenges for mapreduce in big data, in: Services (SERVICES), 2014 IEEE World Congress on, 2014, pp. 182–189, doi: 10.1109/SERVICES.2014.41 T.Sun, J.Deng and K. Deng (2008), 'Scale-free network model with evolving local-world,' in Proc. 4th Int. Nat. Comput. Conf., Vol. 1, pp.237–240

[5] S. Ghemawat , H. Gobioff, S.-T. Leung , The google file system, in: Proceedings of the nineteenth ACM symposium on Operating systems principles, SOSP '03, 2003, pp. 29–43

[6] A. Spark , Apache Spark: Lightning-fast cluster computing, 2015 . [Online; accessed July 2015].

[7]] H. Karau , A. Konwinski , P. Wendell , M. Zaharia , Learning Spark: Lightning-Fast Big Data Analytics, O'Reilly Media, Incorporated, 2015 .

[8] C. Zhang, F. Li, J. Jestes, Efficient parallel knn joins for large data in mapreduce, in: Proceedings of the 15th International Conference on Extending Database Technology, in: EDBT '12, ACM, New York, NY, USA, 2012, pp. 38–49, doi: 10. 1145/2247596.2247602 .

[9] Á. Arnaiz-González, J.F. Díez-Pastor, J.J. Rodríguez, C. García-Osorio, Instance selection of linear complexity for big data, Knowl. Based Syst. (2016), doi: 10. 1016/j.knosys.2016.05.056 .

[10] I. Triguero, S. García, F. Herrera, Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study, Knowl. Inf. Syst. 42 (2) (2013) 245–284, doi: 10.1007/s10115- 013-0706- y .