Distributing and Parallelizing K-Nearest Neighbor for efficient performance

Sharan Gohar Khan i16-1064
Asad Abdul Hameed i16-1068

***Abstract*—K-Nearest Neighbor(KNN) is a very simple and effective algorithm in the field of data mining. It works by comparing the unknown instances with existing data and makes its decision based on their similarity. KNN algorithms have decent accuracy and are quite simple to implement. Its simplicity is its main advantage but performs all of its work during the classification phase which makes it unsuitable for the scenarios when results are required sooner. Its memory requirement and computation complexity is too much. Also it takes very small time for training, but takes alot of time for classification phase. This is one of the primary reason why KNN is avoided especially for larger datasets. In this paper, we will propose a technique that can reduce the runtime of KNN by exploiting resources to their maximum potential.**

To do this we are going to run simple KNN and our proposed modified algorithm in undistributed environment the proposed scheme improves the run time performance by 11% after this we have implemented the modified algorithm in MapReduced approach in distributed environment which has improved run time performance by 30%.

***Keywords*—Big Data Analytics, KNN,Classificaton, Vectorization**

I. Introduction

The k-nearest neighbor (KNN) algorithm is a fundamental classification and regression method for machine learning. KNN is used for many tasks such as text classification prediction of economic events ,medical diagnosis, object classification in images, prediction of protein interac- tions, and so on.KNN works by finding the k nearest points to a given query point in the feature space. The continuous or discrete predicted output value for the query is then com- puted using the corresponding values of the neighbors (e.g. using a majority vote for discrete classification problems, or an average of values in a continuous regression setting). Given the huge volume of data these days it is almost impossible for human analysts to derive meaningful conclusions in a short time frame. Hence data mining techniques are looked upon as tools that can be used to automate the process of knowledge discovery and define relationships and patterns of resemblance given a completely random and raw data set. The majority of data collected for analysis is unsupervised. This gives rise to the need of effective techniques that can process such unsupervised data sets and convert what might seem to be completely random and meaningless into something sensible and valuable. For instance, given a collection of hourly or daily temperatures of a place, it is possible to determine the demography of that place by grouping the temperatures based on a similarity metric. Here, we use what might seem to be a random set of temperatures to determine the demography of a place by simply grouping temperatures and finding the mode temperature of each group. Hence processing data and data mining techniques play a critically important role in technological development.

II. Related Work

KNN and its variants have been used to solve many sort of problems. They are quite simple to implement and they have good accuracy. However in this era of big data [5], we are dealing with massive datasets. The classification time of KNN grows exponentially for such data, therefore researches have done proposed lots of work to reduce the training data. They have proposed techniques like similarity co relations withing different examples to reduce the number of rows in the training set, such models are called condensed KNN. Till now, work is being done to reduce the number of features where ever possible by using feature engineering techniques. Work has also been done to form clusters(group of similar examples), so that we can reduce the number of comparisons significantly as we can simply compare instances with similar clusters. Work can also been conducted on shipping the clusters to different machines. However the issue with this approach is that usually datasets have high variations or co-occurence relationships so we cannot create clusters. Latest research has been conducted on map reducing KNN [3]. In their approach, they split the training set and then ship it to different machines. For classification purposes, test sets are sent to cluster machines and distances are calculated. These distances are then joined together, then the voting for the top nearest neighbors is performed. However, they either take L2 distances or use other variants of nearest neighbors. A latest research [4] has also been conducted on parallelizing the calcula- tions of distances. Basically they first distribute the datasets to different machines and then exploit multi-core processors. This approach has demonstrated promising results for big data processing using KNN. We can take this concept further and involve GPUs as well. A very recent research [1] on this concept involves parallelizing the task using NVIDIA CUDA Cores. They have also utilized redundancy removal techniques to avoid unnecessary comparisions.

III. Background

A. Hadoop Framework

Hadoop is an open source software framework that can run large data-intensive, distributed applications. Hadoop comes

with its own file system called the Hadoop Distributed File System (HDFS) and a strong infrastructural support of managing and processing huge petabytes of data.
Concurrent access is the feature of Hadoop which means to access multiple disks at once to reduce time, the problem of using concurrent access is hardware failure but data replication is used to cater this situation.
HDFS cluster consists of a Namenode, a master server that manages the file system namespace and regulates access to files by clients, this will be a unique server. Read and write operations are performed by the Datanode, they report back periodically back to Namenode.
Hadoop is installed on one computer which is responsible for loading data into the cluster, submitting MapReduce jobs and viewing the results of the job once complete.
To speed up the processing, we need to run parts of the program in parallel. MapReduce is used for this which takes care of many issues such as clock synchronization and optimizing block locality etc.

- MapReduce works by breaking the processing into two phases: The map phase and reduce phase
- Each phase has key-value pairs as input and output.
- The programmer specifies two functions: the map function and the reduce function

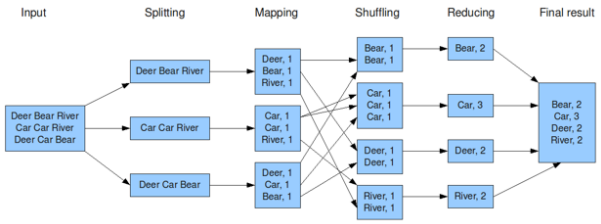

Fig. 1: MapReduce technique

B. Apache Spark

We are going to use Apache Spark for implementation of Knn in distributed environment. It extends MapReduce model to efficiently support more types of computations, including interactive queries and stream processing. Previously work has been done in JAVA MapReduce [3] [4] our approach is going to use Spark Framework. Spark also provides Machine learning library called MLib it contains multiple machine learning algorithms but Knn is not present yet implementing Knn algorithm can be a contribution to MLib library too. [7]
Spark can run in Hadoop clusters and access any Hadoop data source. Spark can be used in JAVA, Python and Scala we are going to use Scala for this project. In Spark, we express computation through operations on distributed collection that are automatically parallelized across the cluster that is called RDD(resilient distributed datasets). RDDs are Sparks fundamental abstraction for distributed data and computation.

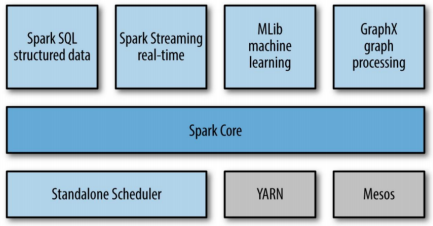

Fig. 2: The Spark Stack

IV. Implementation

Our implementation takes a different approach for distributing KNN. After shipping the dataset clusters, we performed matrix multiplication instead of computing L2 distances. The benefit of this approach is that we will not have to take squares and square root of the values. We get an estimate of similarity between two instances. Also matrix multiplication can be optimized for memory and performance [6]. For example using blocking for efficient cache utilization. GPUs are also specialized for matrix operations. We can also parallelize processing using GPUs at this stage [2]. This pattern is called distributed-parallel processing, we can see its example in animation studios such as Disney, where the graphics rendering tasks are shipped to different machines that have highly capable GPUs on them. However because of the technical complexities of the whole process, well focus majorly on distributing part in our proposed methodology.
We used Apache Spark for implementation. It extends the map-reduce model to efficiently support more types of computations, including interactive queries and stream processing. Previously work has been done on map-reducing using Java. Spark also provides Machine learning library called MLib, which contains many machine learning algorithms that have been adapted for distributed computing. However KNN is not yet present in the library, therefore implementing can also be counted as a contribution to the MLib library. Algorithm for the map function is given as:

Algorithm 1: Map function

Result: Classification labels vector for test set
1 Initialization: Create an empty matrix of size M * N;
2 Create partitions on test set with key-value pair to track output;
3 Ship subsets to slave machines;
4 Put the results back to corresponding matrix locations;
5 Perform voting;
6 Generate and return output vector;

In details, we created test and training splits preferably equal to the number of machines available. In order to keep track of the operations,we created a matrix for each test set multiplied with training set. Once we get the results, we can simply take the voting decision from top similar examples. The representation of the matrix is given as:-

Here Tr represents the training set split and Ts is the test set split. After the data has been sent to the

Tr 1, Ts 1	Tr 1, Ts 2	Tr 1, Ts 3
Tr 2, Ts 1	Tr 2, Ts 2	Tr 2, Ts 3
Tr 3, Ts 1	Tr 3, Ts 2	Tr 3, Ts 3

Fig. 3: Tracking Matrix Representation

slave machines, we have to perform the reducer tasks. The algorithm for the reducer function is given as:

Algorithm 2: Reducer function

Result: Similarity Vector
Load the corresponding training set;
Perform matrix multiplication;
Return similarity vector;

V. Experimental Setup

The implementation of k-Nearest Neighbor technique will be on four nodes which will be connected over a private LAN. One of the node is going to be the Namenode and JobTracker and the other three nodes will be used used as Datan- odes. All the nodes will have the latest version of Hadoop, Spark and Ubuntu installed. We used Scala programming language and integrated Maven build tool to make our development process faster. We calculated run time and speedup as a measure for performance evaluation. We see that the accuracy will remain same with the performance optimizations.

VI. Results

Figure 4, 5, 6 show the results of our optomized Knn algorithm

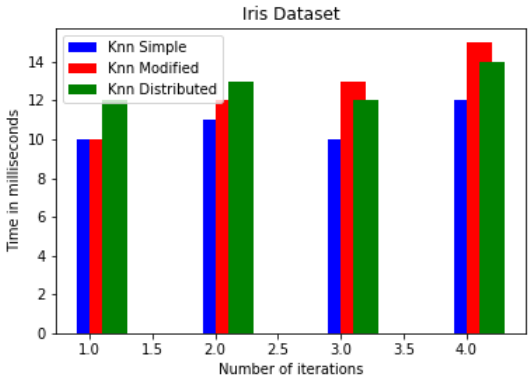

Fig. 4: Comparison on Iris Dataset

The comparison done in Figure 4 was by using Iris Dataset. Iris Dataset is a small dataset with only 150 instances, overall all the implementations performed in fastest time but in most cases Knn Distributed was performing poorly because the splitting of dataset into multiple parts and giving it to cluster was an overhead for the algorithm.

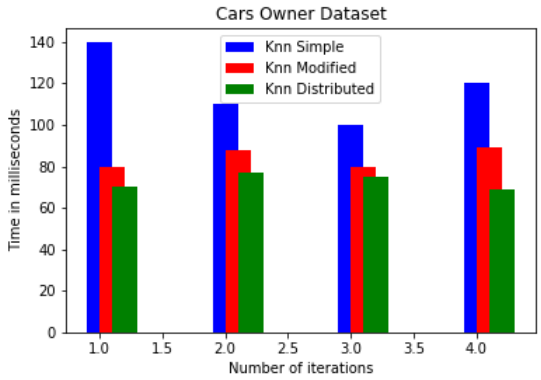

Fig. 5: Comparison on Cars Owner Dataset

The comparison done in Figure 5 was by using Cars Onwer Dataset. Car Owner Dataset is medium size dataset with 2500 instances. Knn Modified performed better with our proposed changes, while Knn Distributed performed close to Knn modified.

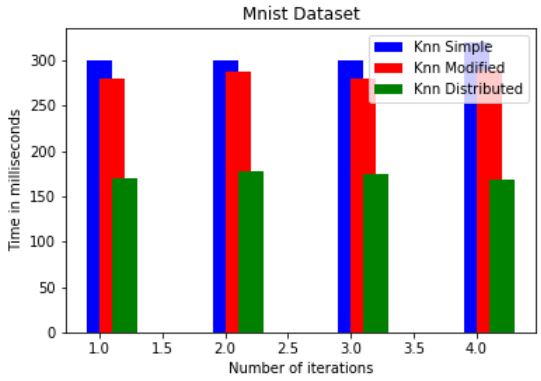

Fig. 6: Comparison on Mnist Dataset

The comparison done in Figure 6 was by using Mnist Dataset. Mnist dataset contains vectorized features of handwritten images. In distributed Knn the jobs were divided among the workers which significantly improved the performance of the algorithm.

VII. Conclusion

We implemented a new approach for distributing KNN algorithm. It has importance for big data processing. It is also significant for academic community as well as machine learning and distributed computing practitioners. Since existing

machine learning library of spark called MLib doesnt have KNN, therefore our implementation can also be taken as a contribution for it.
Overall, this paper shows that the proposed algorithm improves performance by 30% in distributed enviroment compare to the improved undistributed knn algorithm which had improved the performance by 11%.

REFERENCES

[1] G. Chen, Y. Ding, and X. Shen. Sweet knn: An efficient knn on gpu through reconciliation between redundancy removal and regularity. In 2017 IEEE 33rd International Conference on Data Engineering (ICDE), pages 621632, April 2017.

[2] Fabian Gieseke, Justin Heinermann, Cosmin Oancea, and Christian Igel. Buffer k-d trees: Processing massive nearest neighbor queries on gpus. In Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32, ICML14, pages I172I180. JMLR.org, 2014.

[3] Jesus Maillo, Isaac Triguero, and Francisco Herrera. A MapReduce-Based k-Nearest Neighbor Approach for Big Data Classification. Proceedings - 14th IEEE International Conference on Trust, Security and Privacy in Comput- ing and Communications, TrustCom 2015, 2:167172, 2015.

[4] Md Mostofa Ali Patwary, Nadathur Rajagopalan Satish, Narayanan Sundaram, Jialin Liu, Peter Sadowski, Evan Racah, Suren Byna, Craig Tull, Wahid Bhimji, Prabhat, and Pradeep Dubey. PANDA: Extreme Scale Par- allel K-Nearest Neighbor on Distributed Architectures. Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016, pages 494503, 2016.

[5] Josef Spillner and Alexander Schill. Algorithms for dispersed processing. In Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC 14, pages 914921, Washington, DC, USA, 2014. IEEE Computer Society.

[6] Chenhan D. Yu, Jianyu Huang, Woody Austin, Bo Xiao, and George Biros. Performance optimization for the k-nearest neighbors kernel on x86 architec- tures. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 15, pages 7:17:12, New York, NY, USA, 2015. ACM.

[7] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks MLlib: Machine Learning in Apache Spark.Journal of Machine Learning Research. 2016