Exact Fuzzy k-Nearest Neighbor Classification for Big Datasets

Jesus Maillo, Julián Luengo, Salvador García, Francisco Herrera
Department of Computer Science and Artificial Intelligence
University of Granada, Granada, Spain, 18071
Email: {jesusmh, julianlm, salvagl, herrera}@decsai.ugr.es

Isaac Triguero
School of Computer Science
University of Nottingham, Jubilee Campus
Nottingham NG8 1BB, United Kingdom
Email: Isaac.Triguero@nottingham.ac.uk

***Abstract*—The k-Nearest Neighbors (kNN) classifier is one of the most effective methods in supervised learning problems. It classifies unseen cases comparing their similarity with the training data. Nevertheless, it gives to each labeled sample the same importance to classify. There are several approaches to enhance its precision, with the Fuzzy k-Nearest Neighbors (Fuzzy-kNN) classifier being among the most successful ones. Fuzzy-kNN computes a fuzzy degree of membership of each instance to the classes of the problem. As a result, it generates smoother borders between classes. Apart from the existing kNN approach to handle big datasets, there is not a fuzzy variant to manage that volume of data. Nevertheless, calculating this class membership adds an extra computational cost becoming even less scalable to tackle large datasets because of memory needs and high runtime. In this work, we present an exact and distributed approach to run the Fuzzy-kNN classifier on big datasets based on Spark, which provides the same precision than the original algorithm. It presents two separately stages. The first stage transforms the training set adding the class membership degrees. The second stage classifies with the kNN algorithm the test set using the class membership computed previously. In our experiments, we study the scaling-up capabilities of the proposed approach with datasets up to 11 million instances, showing promising results.**

I. Introduction

The k-nearest neighbor method (kNN) [1] is an instance-based classifier that compares new examples with labeled instances of a training set. As a lazy learning algorithm, instead of creating an explicit classification model, it defers all computations until the classification phase. Its classification rule is based on taking the k most similar samples of the training set for an unknown sample. Its similarity is usually a distance measure, e.g. the Euclidean or Manhattan distance. Despite its simplicity, the kNN algorithm highlights as one of top ten algorithms in data mining for its performance [2].

However, the kNN algorithm gives the same importance to every neighbor, assuming that the boundaries between classes are perfectly defined, which is not always true. There is an effective improvement of kNN that alleviates this issue by using fuzzy sets, named Fuzzy-kNN [3]. To do so, Fuzzy-kNN has two different phases. First, it changes the class label for a class membership degree. After that, it calculates the kNN with the membership information, achieving higher accuracy rates in most classification problems. There are different fuzzy approaches to kNN that has shown to be significant improvements. Nevertheless, the original Fuzzy-kNN has demonstrated to be one of the most effective in practice [4]. Also, it has been the preferred choice in multitude of fields such as medicine [5], economy [6] and many other applications.

The kNN algorithm and also his fuzzy variant has two main problems to tackle large datasets because of their lazy behavior: runtime and memory consumption. The Fuzzy-kNN algorithm increases both issues because it needs an extra phase to compute the class membership degree. This stage needs nearly the double of main memory available in order to compute the kNN method with the training set against itself.

These two big data issues can be handled with cloud-based technologies. The MapReduce paradigm [7] and Spark [8] as a framework highlight as powerful tools to tackle data-intensive applications compared to other schemes such as Message Passing Interface [9]. The main reasons for their success are the distributed file system and the fault-tolerant mechanism.

Currently, there are contributions to manage big data problems with the kNN algorithm. Focusing on classification problems, the method proposed in [10] is an approximate kNN algorithm consisted of two stages. First, it groups the data to separate the whole dataset in different splits, and secondly, it computes a kNN in each partition providing different results of the original kNN. Another relevant contribution is kNN-IS [11], where authors proposed an exact approach that can classify huge datasets. The map phase splits the training set and computes the kNN of every test sample. The reduce phase collects all the candidates to be the nearest neighbors and reports the final k nearest neighbors. Thus, it allows us to deal with big training and test sets with good runtimes and obtaining the same result than the original kNN algorithm.

However, there is only one approach of the Fuzzy-kNN algorithm to deal with big data problems. In [12], the authors proposed simply to split the data and apply Fuzzy-kNN in each split. Then it collects all the labels of each split and computes the final results by majority voting. With this scheme, it loses valuable information about the problem and could be applied with all the data mining algorithms. In addition, when training set and/or test set are big, it will need too many splits and that produces a lot of votes to consider in the last stage, becoming a scalability problem.

In this paper, we propose a MapReduce-based approach implemented on Spark for the Fuzzy-kNN algorithm. We take advantage of the in-memory primitives of Spark to manage large training set by splitting the data. Also, it handles enormous test sets by iterating over the chunks of this set, if necessary. As we explained briefly, the scheme of the Fuzzy-kNN algorithm has two stages. The first calculates the class membership degree and is the most complex computationally.The map

stage distributes the training set and it computes the kNN on each split. The reduce stage collects all the candidates to be the nearest neighbors and obtain the final k closest samples. Then, it computes the class membership and reports a new training set with this information. Hence, the runtime is speeded up and the scalability issue is alleviated. The second stage is divided into map and reduce phases. The map phase consists of distributing the computation of the distances between the samples of the test set and the split of the training data. Thus, each map obtain k candidates to be the k closest neighbors. Multiple reducer tasks collect all the candidates provided by the maps and it calculates the final k neighbors and then it will classify with the knowledge of the class membership degree previously computed. Through the text, we will denote it as Exact Fuzzy-kNN (EF-kNN).

In summary, the main contribution sof this work are as follows:

- Design and develop an exact model of Fuzzy-kNN. The implementation makes use of in-memory Spark operations in order to accelerate all the stages of the method.
- A experimental study of the scalability and accuracy of this model.

The remainder of this paper is organized as follows. Section II introduces the state-of-art in Fuzzy-kNN and the big data technologies. Then, Section III details the proposed Fuzzy-kNN model. The experimental study is described in Section IV. Section V concludes the paper and outline the future work.

II. Preliminaries

This section supplies the necessary background information on the Fuzzy-kNN algorithm (Section II-A) and the big data technologies (Section II-B).

A. Fuzzy k-Nearest Neighbors algorithm and complexity

The Fuzzy-kNN algorithm [3] is an improvement upon the standard kNN algorithm. It has demonstrated to be very competitive in comparison to others Fuzzy approaches in terms of accuracy. To carry out this method, it is necessary to pre-calculate the class memberships with the training set. After that, it calculates the kNN of each sample of the test set. A formal notation for the Fuzzy-kNN algorithm is the following:

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 $\mathbf{x}_i$ is a vector $(\mathbf{x}_{i1}, \mathbf{x}_{i2}, \mathbf{x}_{i3}, \ldots, \mathbf{x}_{ij})$, where, $\mathbf{x}_{ij}$ is the value of the j-th feature of the i-th sample. Every sample of TR belong to a known class ω, while it is unknown for TS. Fuzzy-kNN has two different stages.

The first stage calculates the k_{memb} nearest neighbors of the TR against itself keeping following a leave-one-out scheme. To do this, it searches the k_{memb} closest samples by calculating the distances between $\mathbf{x}_{train}$ and all the samples of TR. Once calculated the neighbors, it creates the class membership as shown Equation 1. Thus, the TR has a class membership vector instead of the original class label.

$$u_j(x) = \begin{cases} 0.51 + (n_j/k_{memb}) \cdot 0.49 & if \quad j = i \\ (n_j/k_{memb}) \cdot 0.49 & if \quad j \neq i \end{cases} \tag{1}$$

The second stage computes the k nearest neighbors like the first stage, but in this occasion, it calculates for each instance of TS the k closest in the TR. After that, it decides the resulting class as show the Equation 2 rather than as the kNN algorithm does (i.e. majority voting).

$$u_i(x) = \frac{\sum_{j=1}^{K} u_{ij}(1/|x - x_j|^{2/(m-1)})}{\sum_{j=1}^{K}(1/|x - x_j|^{2/(m-1)})} \tag{2}$$

Although Fuzzy-kNN algorithm improves in terms of accuracy to kNN method, it increases the algorithmic complexity provoking two main problems to handle large-scale data:

- Runtime: The complexity to find the nearest neighbor training example of a single instance is $\mathcal{O}(n \cdot D)$ where n is the number of training instances and D the number of features. When it searches for k neighbors, its algorithmic complexity rises to $\mathcal{O}(n \cdot log(N))$. Moreover, it repeats for each training sample in the first stage and does the same for each test sample in the second stage. The runtime of creating the class membership and predicting the output class could be despised comparing with the runtime of calculating the k nearest neighbors in both stages.
- Memory consumption: Fuzzy-kNN model needs to store in main memory the training and test dataset in order to boost the computation. When TR and TS are really big, they might easily exceed the available memory.

These difficulties encourage to design a distributed model of Fuzzy-kNN by using big data technologies as the MapReduce paradigm and Apache Spark platform.

B. MapReduce programming model: Apache Spark

The MapReduce programming paradigm [13], designed by Google in 2003, is a scale-out data processing tools. It is aimed at processing large-scale datasets by distributing the storage and execution through a cluster of machines.

The MapReduce model defines three stages to manage distributed data: Map, Shuffle and Reduce. The first one reads the raw data in form of <key-value> pairs, and it distributes through several nodes for parallel processing. The Shuffle is responsible for merging all the values associated with the same intermediate key. Finally, reduce phase combines those coincident pairs and it aggregates it into smaller key-value pairs. Figure 1 shows a scheme of this process. MapReduce provides some features for relieving the user from some technical details: data splitting, fault-tolerance and job communication. In [14], authors expose an exhaustive review of this framework and other distributed paradigms.

Apache Hadoop [1], is the most popular open-source implementation of MapReduce paradigm, but it can not reuse data through in-memory primitives. Apache Spark is a novel implementation of MapReduce that solves some of the Hadoop drawbacks [2]. The most important feature is the type of data

[1] Apache Hadoop. Web: http://hadoop.apache.org/

[2] Apache Spark. Web: https://spark.apache.org/

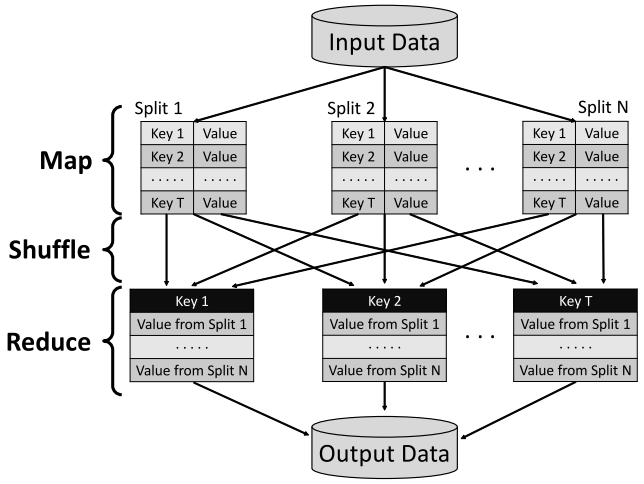

Fig. 1: Data flow overview of MapReduce

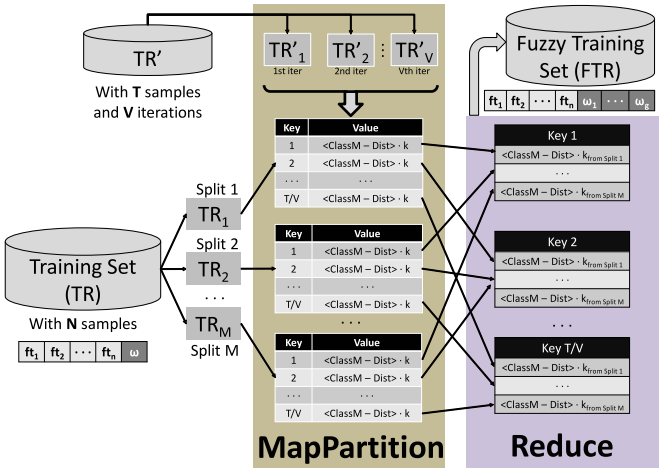

Fig. 2: Flowchart of the EF-kNN

structure that parallelize the computations in a transparent way, it is called Resilient Distributed Datasets (RDDs). In addition, RDD allows us to persist and reuse data, cached in memory. Moreover, it was developed to cooperate with Hadoop, specifically with its distributed file system (Hadoop Distributed File System)

Spark includes a scalable machine learning library on top of it known as MLlib[3]. It has a multitude of statistics tools and machine learning algorithms along different areas of KDD like classification, regression, or data preprocessing.

III. EF-kNN: Exact Fuzzy-kNN for big data

In this section, we present an exact approach of distributed Fuzzy-kNN model for big data classification using Spark. We focus on the reduction of the runtime of the Fuzzy-kNN classifier, when the training and test sets are big. Developing in a parallel framework involves many factors that may impact the execution time, such as the number of distributed tasks (Maps), the number of Reduce jobs or the network traffic. Thus, writing these methods is a challenging and it provokes many key-points must be taken into account in order to design an efficient and scalable model.

The main workflow of the Fuzzy-kNN algorithm is composed of two stages. Sections III-A explains how it computes the class membership degree obtaining a new training set. This stage computes over the training set versus itself, becoming the heaviest computationally of the two stages. Section III-B shows the second stage, it uses the enriched knowledge of the FTR calculated to classify the TS.

A. Class membership degree stage

This subsection explains the MapReduce process that computes the class membership of the training set. Figure 2 shows the flowchart of the EF-kNN, dividing the computation into two phases: map and reduce operations. The map phase divides the TR and calculates for each split the distance and takes the classes of the k nearest neighbors for every training sample.

[3]Machine Learning Library for Spark. Web: http://spark.apache.org/mllib/

Algorithm 1 MapPartition function

Require: TR_j, TR_v, $kMemb$
1: **for** $t = 0$ **to** $size(TR_v)$ **do**
2: $Clas\&Dist_{t,j} \leftarrow$ Compute kNN (TR_j, $TR_v(t)$, $kMemb$)
3: $result_j \leftarrow (< key : t, value : Clas\&Dist_{t,j} >)$
4: **end for**
5: EMIT($result_j$)

The reduce stage joins all the candidates to be the k nearest neighbors and obtains the k closest definitives samples. With them, it reports a new TR that adds a vector with the class membership. It will called Fuzzy Training Set (FTS).

1) Map phase: Let us start with the training set TR read from HDFS as a RDD object. The TR has already been split into m parts, as a parameter defined by the user. Thus, there is one map task for each TR_j split ($Map_1, Map_2, \cdot, Map_m$ - where $1 \leq j \leq m$). Therefore, each map contains approximately the same number of training samples.

To obtain an exact approach of the class membership degree, it is necessary all the training samples in each map in order to compare every training sample against the whole training set. It supposes that TR_j and TR fit together in memory. Otherwise, the TR will be split into v chunks and it is iterated in a sequential way to allow for being stored in memory and properly executed.

Algorithm 1 encloses the pseudo-code of this function. In our implementation in Spark we use the *mapPartitions()* transformation, which runs the function defined on each split of the RDD in a distributed way.

Every map j will build a vector $Clas\&Dist_{t,j}$ of pairs <class, distance> of dimension $kMemb$ for each training sample t in TR_v. Instruction 2 calculates the class and the distance to its $kMemb$ closest samples. To accelerate the latest actualization of the nearest neighbors in the reducers, every vector $Clas\&Dist_{t,j}$ is sorted in ascending order.

Every map tasks reports a matrix of Clas&Dist that represents the candidate to be the nearest neighbors, which are identified by ID as shown Instruction 3. With this scheme, it could use multiple reducers to handle big training sets.

Algorithm 2 Reduce by key operation. EF-kNN

Require: $result_{key}, kMemb$
1: $cont1 \leftarrow 0$; $cont2 \leftarrow 0$
2: **while** $i < kMemb$ **do**
3: **if** $result_{key}(cont1).Dist \leq result_{reducer}(cont2).Dist$ **then**
4: $out(i) \leftarrow result_{key}(cont1)$
5: cont1++
6: **else if** $result_{key}(cont1).Dist = result_{reducer}(cont2).Dist$ **then**
7: **if** $i < kMemb$ **then**
8: i++
9: $out(i) \leftarrow result_{key}(cont2)$
10: cont2++
11: **end if**
12: **else**
13: $out(i) \leftarrow result_{key}(cont2)$
14: cont2++
15: **end if**
16: i++
17: **end while**
18: EMIT(out)

Algorithm 3 Map operation. EF-kNN

Require: $sample, neighbors_{kMemb}$
1: Initialize array membership to 0
2: **for** $j \leftarrow 0$ **to** $kMemb$ **do**
3: $membership(neigh_j) = +1$
4: **end for**
5: **for** $t = 0$ **to** $kMemb$ **do**
6: **if** $sample.label = t$ **then**
7: $membership(t) \leftarrow 0.51 + (membership(t)/kMemb) * 0.49$
8: **else**
9: $membership(t) \leftarrow (membership(t)/kMemb) * 0.49$
10: **end if**
11: **end for**
12: EMIT $sample\ join\ membership$

2) Reduce phase: Multiple reducers collect from the maps the tentative $Kmemb$ nearest neighbors and they aim to obtain the closest ones for each training sample contained in TR_v. To do so, all the elements are grouped by key and compute the class membership degree as shown in 2.

The reduce tasks will update the candidates selecting the $kmemb$ nearest by merging the output of the map. Since the vectors coming from the maps are ordered according to the distance, the update process becomes faster. This consists of merging two sorted lists ensuring that neighbors with the same distance are conserved both if possible. So that, the complexity in the worst case is $\mathcal{O}(kMemb)$. This function compares one by one every distance. If the distance is lesser than the current closest, the distance and the class is updated, if the distance is higher, it will discard and if it is exactly the same and there is enough space, it will conserve both.

At this point, we have the $kMemb$ nearest neighbors of each instance of the training set among the training set. After that, another map stage will calculate the class membership degree as show Algorithm 3. In our implementation in Spark we use the *map()* transformation, which runs the function defined on each sample of the RDD in a distributed way. To do so, it applies the Equation 1 and changes the single label for a vector that represents the membership of each class.

Finally, Algorithm 3 returns each original sample maintaining the original features and changing the label for the class membership computed. Thus, it generates a new Fuzzy Training Set, which is the input of the Final classification stage described in Section III-B.

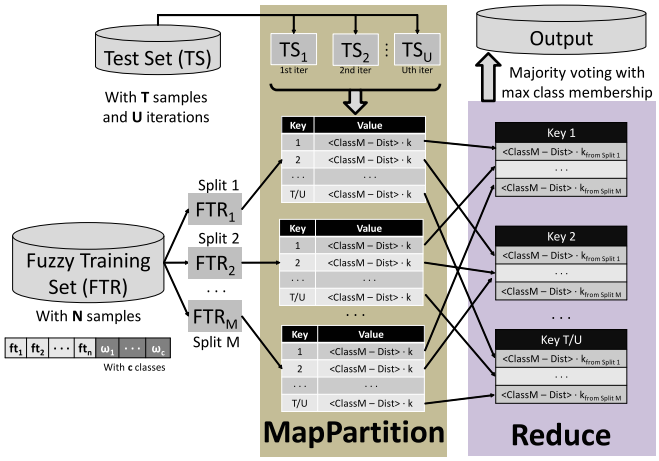

Fig. 3: Flowchart of the Final classification stage

Algorithm 4 Classification: Map function

Require: $TR_j\ TS; k$
1: **for** $t = 0$ **to** $size(TS)$ **do**
2: $neighbors\text{-}memb_{t,j} \leftarrow$ Compute kNN (TR_j, $TS(t)$, k)
3: $result_t \leftarrow (< key : t, value : neighbors_{t,j} >)$
4: **end for**
5: EMIT($result_t$)

B. Final classification stage

Figure 3 presents the flowchart of the classification stage divided into the two basic operations of MapReduce. The MapPartition need the corresponding split of the TR, TS and the parameter k. Thus, it computes for each TR_i chunk the k nearest neighbors taking the distance and the classes for every sample of the TS. The reduce stage collects all k nearest neighbors of each split and computes the definitives k closest samples. Finally, it reports the classification label of each sample by majority voting.

The classification stage is computationally lighter than the first stage because the test set is usually smaller than training set. Thus, it needs less main memory capacity in order to store in memory the data and the runtimes are lower due to compute fewer samples. Thus, the classification stage focuses on obtaining an exact result starting of the FTR.

1) Map phase: Let us assume that FTR and TS can be stored in main memory. FTR is split into m parts, which contain approximately the same number of samples. TS has to remain unpartitioned in order to compute all the candidate to be the k nearest neighbors in each partition of the FTR, calculated by distributed map operations.

Note that if the number of partitions of the FTR remains as the splits of TR, a shuffle stage is avoided and therefore, the efficiency will be increased in terms of runtime.

Algorithm 4 shows the pseudo-code of the map function. For each sample of the TS computes its k nearest neighbors. The variable $neighbors\text{-}memb$ saves the distances and the class membership degree vector. Finally, it adds the id of each test sample as a key and emits them to the reduce phase.

2) Reduce phase: The reduce stage aims to aggregate all the tentative nearest neighbors in order to finally get the

Algorithm 5 Classification: Reduce by key function

Require: $candidate_{key,1}$, $candidate_{key,2}$, k
1: $itC_1 = 0$, $itC_2 = 0$
2: **while** $i < k$ **do**
3: **if** $candidate_{key,1}(itC_1).Dist \leq candidate_{key,2}(itC_2).Dist$ **then**
4: $result(i) \leftarrow candidate_{key,1}(itC_1)$
5: **if** $candidate_{key,1}(itC_1).Dist = candidate_{key,2}(itC_2).Dist$ **then**
6: i++
7: **if** $i < kMemb$ **then**
8: $result(i) \leftarrow candidate_{key,2}(itC_2)$
9: itC_2++
10: **end if**
11: **end if**
12: itC_1++
13: **else**
14: $result(i) \leftarrow candidate_{key,2}(itC_2)$
15: itC_2++
16: **end if**
17: i++
18: **end while**

TABLE I: Description of the used datasets

Dataset	#Samples	#Features	$\#\omega$
PokerHand	1,025,010	10	10
Susy	5,000,000	18	2
Higgs	11,000,000	28	2

definitive k nearest neighbors of the whole FTR. Keeping the distance and the class membership degree vector.

Reduce by key function performs an aggregation of two sorted list by distance in a new sorted list of k size. Algorithm 5 shows the pseudo-code, considering that the distances could be the same for some examples. Hence, it keeps both neighbors in case the k parameter permit it. With this implementation, it is achieved to join the result of two maps with an algorithmic complexity of $\mathcal{O}(k)$.

Finally, with the resulting neighbors, it is applied one last map function to calculate the predicted class label. In this map the Equation 2 is applied to every sample of the TS.

IV. Experimental Framework and Results

In this section, we present all the questions related to the experimental study. Section IV-A establishes the experimental set-up and Section IV-B discusses the results achieved.

A. Experimental set-up

We will use three large datasets from the UCI machine learning repository [15] to evaluate our model: PokerHand, Susy and Higgs. Table I presents the number of samples ($\#Samples$), features ($\#Features$), and classes ($\#\omega$). In our experiments, we follow a 5 fold cross-validation scheme.

The original Fuzzy-kNN has 2 parameters and these are exactly the same for the proposed model:

- $kMemb$**:** We investigate 3,5 and 7 as the number of neighbors to compute class membership.
- k**:** The number of neighbors to predict unseen cases.
- $\#Maps$**:** An extra parameter is needed, this is the number of map tasks that will compute in a concurrent way and is the number of splits of the TR set.

In this work we evaluate the scalability and the efficiency with the following two measures:

- *Accuracy:* This measure reflect the efficiency of the algorithms. It represents the number of right predictions against the total number of predictions ([16] [17]).

TABLE II: Influence of $kMemb$ and k with Poker dataset

$kMemb$ & k	MembRuntme	ClasRuntime	TotalRuntime	Acc
3	462.2232	130.7489	592.9720	0.5257
5	484.2841	145.6419	629.9260	0.5313
7	499.0342	141.0916	640.1258	0.5336

TABLE III: Influence of map tasks with Susy dataset

$kMemb$ & k	**Accuracy**		
	128	256	384
3	0.7338	0.7246	0.7284
5	0.7350	0.7292	0.7278
7	0.7319	0.7291	0.7238

- *Runtime:* We will collect the time spent by the Fuzzy-kNN algorithm to compute the class membership of the training set and to classify a given test set versus the training set. The total runtime includes reading and distributing all the data.

All the executions have been run on a cluster composed of sixteen computing nodes managed by another master node. All the nodes have the following features has 2 Intel Xeon CPU E5-2620 processor, 6 cores (12 threads) per processor, 2 GHz and and 64 GB of RAM. The network is Infiniband 40Gb/s. This hardware was configured providing a maximum number of current tasks to 384. Each task has 2 GB of main memory available. Every node runs with Cent OS 6.5 as operating system and was configured with Spark 1.6.2. Thus, we can not explore a number of maps greater than 384 in order to obtain realistic result to study the scalability properly.

B. Analysis of results

This section studies the results collected from the experimental study. Specifically, we analyze the proposed method in terms of runtime and accuracy.

To do this, we use the Poker, Susy and Higgs datasets with the proposed algorithm. We could not run further of these datasets because the runtime increases too much.

Table II shows the runtime of compute class membership stage (MembRuntime), classification stage (ClasRuntime) and total runtime (TotalRuntime) in seconds, and the test accuracy (Acc) with Poker dataset. In order to simplify the results shown, the value of $kMemb$ and k will be the same and equal to 3, 5 and 7. In addition, the number of maps will be set to 256 for this purpose.

Table III presents the accuracy with the number of maps 128, 256 and 384 depending on the number of neighbors to be considered. Thus, it shows how much the accuracy is affected w.r.t the number of map tasks.

Figure 4 shows the total runtime (in seconds) versus the number of map tasks. In order to simplify the results shown, the value of $kMemb$ and k will be the same and equals to 3.

Figure 5 shows the runtime (in seconds) obtained by the three datasets, with $kMemb$ and k set to 3 and 384 as a number of maps. Note that the runtime is very high with Higgs dataset. For this reason, Higgs dataset will not set the different values of k, $kMemb$ or number of maps in this experimental study.

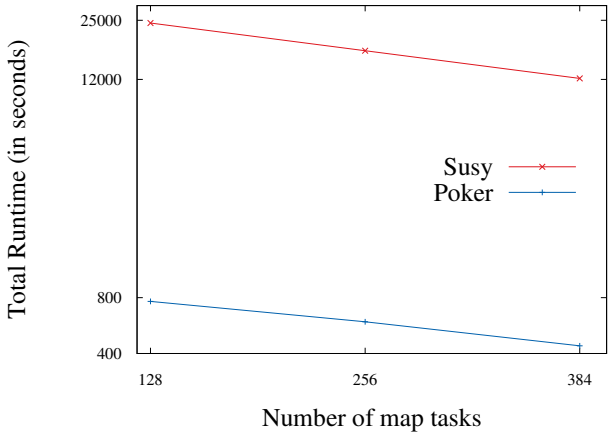

Fig. 4: Influence of the number of maps in the total runtime

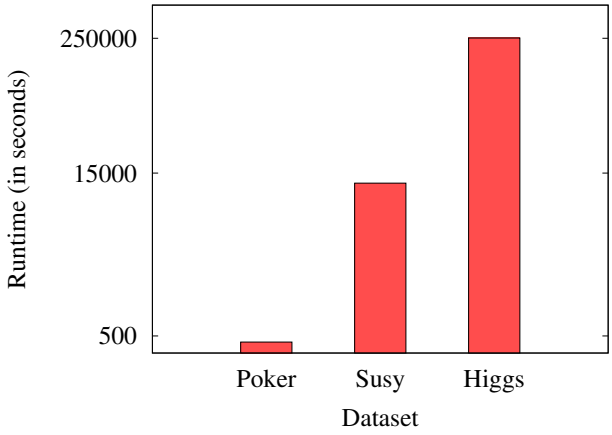

Fig. 5: Runtime with Poker, Susy and Higgs datasets

According to these tables and plots, we can conclude that:

- According to the Table II, regarding $kMemb$ and k parameters, the total runtime does not increase too much in any of the three models, despite the increase in network traffic and the calculation of the neighbors in the reduce stage, due to the design performed.
- We can observe from Table III how the accuracy changes depending on the number of map tasks. This is because for a test example there are several training data points whose distances are exactly the same. When it is a distributed execution, the definitive neighbors will depend on the order in which they arrive from each map output. This uncertainty arises also in the sequential version of the algorithm, but the criterion of maintaining the first one that arrives or to always update the last one makes its results immutable.
- Analyzing Figures 4 and 5, the scalability of the proposed model obtains a linear behavior. However, the runtime on Higgs dataset is pretty high, and reveals a weakness of the proposed algorithm. More hardware will be needed when the runtime matter.

V. Conclusions and Further Work

In this paper, we have developed a scalable and distributed solution for the Fuzzy k-NN algorithm based on Spark. Its main achievement is to handle large-scale datasets with the same accuracy results than the original Fuzzy kNN algorithm. Thankful to Spark framework and the model designed, we only found the dependency in the hardware capabilities. In addition, despite generating more network transfer from the map operations to the reducers, the number of neighbors does not drastically increase to the total runtime. As future work, we aim to speed up the Fuzzy-kNN by an approximate model, focusing on the bottleneck that is the computing of the class membership degree.

Acknowledgment

This work has been supported by the projects TIN2014-57251-P and P11-TIC-7765. J. Maillo holds a FPU scholarship from the Spanish Ministry of Education.

References

[1] T. M. Cover, P. E. Hart, Nearest neighbor pattern classification, IEEE Transactions on Information Theory 13 (1) (1967) 21–27.
[2] X. Wu, V. Kumar (Eds.), The Top Ten Algorithms in Data Mining, Chapman & Hall/CRC Data Mining and Knowledge Discovery, 2009.
[3] J. M. Keller, M. R. Gray, J. A. Givens, A fuzzy k-nearest neighbor algorithm, IEEE Transactions on Systems, Man, and Cybernetics SMC-15 (4) (1985) 580–585.
[4] J. Derrac, S. Garcia, F. Herrera, Fuzzy nearest neighbor algorithms: Taxonomy, experimental analysis and prospects, Information Sciences 260 (2014) 98 – 119.
[5] H. L. Chen, C. C. Huang, X. G. Yu, X. Xu, X. Sun, G. Wang, S. J. Wang, An efficient diagnosis system for detection of parkinsons disease using fuzzy k-nearest neighbor approach, Expert Systems with Applications 40 (1) (2013) 263 – 271.
[6] H. L. Chen, B. Yang, G. Wang, J. Liu, X. Xu, S. J. Wang, D. Y. Liu, A novel bankruptcy prediction model based on an adaptive fuzzy k-nearest neighbor method, Knowledge-Based Systems 24 (8) (2011) 1348 – 1359.
[7] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Communications of the ACM 51 (1) (2008) 107–113.
[8] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, 2012, pp. 1–14.
[9] W. Gropp, E. Lusk, A. Skjellum, Using MPI: portable parallel programming with the message-passing interface, Vol. 1, MIT press, 1999.
[10] Z. Deng, X. Zhuand, D. Cheng, M. Zong, S. Zhang, Efficient knn classification algorithm for big data, Neurocomputing 195 (2016) 143 – 148, learning for Medical Imaging.
[11] J. Maillo, S. Ramirez, I. Triguero, F. Herrera, knn-is: An iterative spark-based design of the k-nearest neighbors classifier for big data, Knowledge-Based Systems 117 (2017) 3 – 15.
[12] O. Hegazy, S. Safwat, M. E. Bakry, A mapreduce fuzzy techniques of big data classification, in: 2016 SAI Computing Conference (SAI), 2016, pp. 118–128.
[13] J. Dean, S. Ghemawat, Map reduce: A flexible data processing tool, Communications of the ACM 53 (1) (2010) 72–77.
[14] A. Fernández, S. Río, V. López, A. Bawakid, M. del Jesus, J. Benítez, F. Herrera, Big data with cloud computing: An insight on the computing environment, mapreduce and programming frameworks, WIREs Data Mining and Knowledge Discovery 4 (5) (2014) 380–409.
[15] M. Lichman, UCI machine learning repository (2013).
URL http://archive.ics.uci.edu/ml
[16] E. Alpaydin, Introduction to Machine Learning, 2nd Edition, The MIT Press, 2010.
[17] I. H. Witten, E. Frank, M. A. Hall, C. J. Pal, Data Mining: Practical machine learning tools and techniques, Morgan Kaufmann, 2016.