Fast and Scalable Approaches to Accelerate the Fuzzy k Nearest Neighbors Classifier for Big Data

Jesus Maillo, Salvador García, Julián Luengo,
Francisco Herrera *Senior Member, IEEE,* and Isaac Triguero *Member, IEEE,*

***Abstract*—One of the best-known and most effective methods in supervised classification is the k nearest neighbors algorithm (kNN). Several approaches have been proposed to improve its accuracy, where fuzzy approaches prove to be among the most successful, highlighting the classical Fuzzy k nearest neighbors (FkNN). However, these traditional algorithms fail to tackle the large amounts of data that are available today. There are multiple alternatives to enable kNN classification in big datasets, spotlighting the approximate version of kNN called Hybrid Spill Tree. Nevertheless, the existing proposals of FkNN for big data problems are not fully scalable, because a high computational load is required to obtain the same behavior as the original FkNN algorithm. This work proposes Global Approximate Hybrid Spill Tree FkNN and Local Hybrid Spill Tree FkNN, two approximate approaches that speed up runtime without losing quality in the classification process. The experimentation compares various FkNN approaches for big data with datasets of up to 11 million instances. The results show an improvement in runtime and accuracy over literature algorithms.**

***Index Terms*—Fuzzy sets, k nearest neighbors, Classification, MapReduce, Apache Spark, Big Data**

I. Introduction

The Fuzzy k Nearest Neighbor algorithm (FkNN) [1] is developed with the aim of improving and alleviating the main weakness of the k Nearest Neighbor algorithm (kNN) [2]. This weakness resides in considering all neighbors as equally important in the classification, making the kNN algorithm more vulnerable to noise at the class boundaries, leading to a downgrading of the classification.

In the experimental analysis at [3], the classic algorithm FkNN stands out as one of the most effective approaches. FkNN is composed of two stages: class membership degree and classification. The first stage changes the label of the class by a vector of membership degree belonging to each class, according to the closest training instances. To calculate the nearest instances, it uses a similarity function, usually with a distance function (Euclidean or Manhattan). The second stage calculates the kNN with the information of the membership degree. Thus, it is possible to detect borders with greater precision, being less affected by noise and improving the kNN in most classification problems used in many applications such as medicine [4], spacecraft [5], and many other fields.

J. Maillo, S. García, J. Luengo and F. Herrera are with the Department of Computer Science and Artificial Intelligence, University of Granada, 18071 Granada, Spain (e-mail: {jesusmh, julianlm, salvagl, herrera}@decsai.ugr.es).

I. Triguero is with the Automated Scheduling, Optimisation and Planning Research Group. School of Computer Science. University of Nottingham. Jubilee Campus. Nottingham NG8 1BB, United Kingdom (e-mail: Isaac.Triguero@nottingham.ac.uk).

Nowadays, FkNN and kNN are used in many areas of data mining. They are used as data preprocessing techniques [6] to deal with imperfect data [7] and uncertainty in the classification process by means of aggregation operators [8]. Studies to improve the FkNN algorithm and its applications continue to develop in many areas such as convergence [9] and runtime improvement [10]. There are some recent proposals that enhance quality of the classic FkNN classifier, two proposals based on evolutionary algorithms [11] and [12] and one proposal based on parameter independent fuzzy weighted kNN [13]. Nevertheless, these solutions used to increase the computational complexity, making the algorithm less scalable for the application in big data problems. For this reason, we will focus on the classical FkNN algorithm.

In the big data environment [14], the kNN and FkNN algorithms have been key to solving different machine learning problems such as fuzzy-rough based NN classification [15], time-series forecasting [16] or data preprocessing to obtain quality data [17]. In this work, we are focused on standard classification. When handling large datasets the kNN and FkNN classifiers have problems regarding runtime and memory consumption. There is an exact proposal of the kNN algorithm to address big data problems and it is called *k Nearest Neighbor - Iterative Spark* (kNN-IS) [18]. In addition to this exact version, there are also approximate variations that drastically reduce execution times: Metric-Tree [19] and Spill-Tree. In [20], the authors studied the Metric-Tree and Spill-Tree models and proposed the Hybrid Spill-Tree model [21] (HS). HS is the hybridization of the two models with the aim of improving the runtime in big data.

Regarding the fuzzy approach, in [22], we investigate the feasibility of an exact approach to apply FkNN in big data called Global Exact Fuzzy k Nearest Neighbors (GE-FkNN) [22]. Even though it is able of scaling up to large datasets, the runtime of the first stage are considerably high, causing a bottleneck. Subsequently, the authors of contribution [23] present a preliminary study on the use of approximate kNN search to accelerate the execution time and alleviate the bottleneck.

The objective of this work is to design and develop a FkNN model capable of handling large datasets accurately and quickly. To do this, we use the Spark framework and use HS as the base algorithm due to its balance between scalability and accuracy that improves previous kNN proposals in the literature. The proposed algorithm is composed of the same two stages of classical FkNN: membership degree and

classification. The main difference of the proposed algorithm can be noted in the first stage, focusing on handling the bottleneck with two different approaches:

- Local Hybrid Spill Tree FkNN (LHS-FkNN): The local approach divides the dataset into different parts and calculates the class membership degree internally in each partition, without considering other partitions.
- Global Approximate Hybrid Spill Tree (GAHS-FkNN): The global approach is based on the HS model. It generates a tree with the instances of the training set and distributes it among all the computation nodes, considering all the instances for the calculation of the class membership degree.

The second stage classifies the unseen samples from the test set using the class membership degree knowledge calculated in the first stage. The classification stage is the same for both models, following a HS based approach and with a workflow similar to the first stage of GAHS-FkNN. The novelty of the proposal is the use of approximate kNN searches, presenting local and global approaches, achieving quality accuracy and scalability that allows execution with large datasets through the use of the MapReduce [24] paradigm and the Spark framework [25].

In order to study the performance of this model, experiments have been carried out on 8 datasets with up to 11 million instances and 631 features. The experimental study analyzes the accuracy and runtime making a comparison with existing algorithms of the literature.

In addition, we have developed a software package with FkNN algorithms for big data, making use of in-memory native operations and distributed computing from Apache Spark. The developed algorithms can be found in the repository https://spark-packages.org/package/JMailloH/HS_FkNN.

The paper is structured in the following five sections. Section II introduces the state of the art in the FkNN and Hybrid Spill-Tree algorithms. Next, Section III details the proposals of the FkNN algorithm. Section IV describes the experimental study and Section V includes multiple analyses of results. The Section VI concludes the document and outlines future work.

II. Preliminaries

This section provides background knowledge of the FkNN algorithm (Section II-A), the Hybrid Spill-Tree (Section II-B) and the big data technologies used (Section II-C).

A. Fuzzy k nearest neighbors and its computational complexity

FkNN needs a pre-computation stage in the training set, which calculates the class membership degree. Afterwards, FkNN calculates the nearest neighbors for each unseen instance and decides on the predicted class with the highest membership degree. A formal notation for the FkNN algorithm is as follows:

Let TR be a training set and TS a test set, composed of $\mathbf{n}$ and $\mathbf{t}$ instances respectively. Each instance $\mathbf{x}_i$ 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 i-th instance and j-th feature. For each instance of TR its class ω is known. However, for TS instances the class is unknown.

FkNN has two stages: class membership degree computation and classification. The first stage calculates the kNN for each instance of TR, keeping a scheme leave-one-out selecting the k instances with a shorter distance. Finally, it calculates the class membership degree according to the Equation 1. The result of the first stage is the TR modifying the class label ω, for a membership vector to each class $(\omega_1, \omega_2, \ldots, \omega_l)$ where l is the number of classes. This new set will be called Fuzzy Training Set, FTR.

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

For each instance of the TS, the classification stage calculates its kNN in FTR. Thus, it gets the membership vector of each neighbor and aggregates this vector by applying the Equation 2. Finally, the class with a higher membership will be predicted.

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

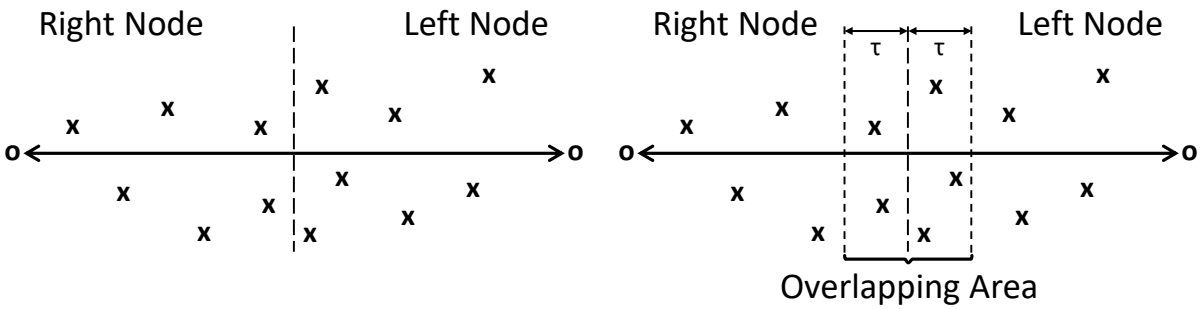
The first stage of FkNN, which is an extra stage compared to kNN, causes increased computational complexity and generates two issues to deal with big data problems:

- Runtime: The complexity of computing kNN for an instance is $\mathcal{O}(n \cdot c)$, where n is the number of instances of TR and c is the number of features. For more than one neighbor, it increases to $\mathcal{O}(n \cdot log(N))$. In addition, FkNN has an extra stage of computation for calculating the class membership degree.
- Memory consumption: To speed up the calculation, the TR and TS sets stored in the main memory are required. However, when both sets are large, the available main memory is easily exceeded.

To alleviate these difficulties, we worked on the design of two approximate models based on Hybrid Spill-Tree developed under the big data technologies of MapReduce and Apache Spark.

B. Hybrid Spill-Tree: Approximate kNN search

In the search for the nearest neighbor two approaches can be followed: Exact and Approximate. The exact approach aims to ensure that the instance identified as closest is actually the closest. To do this, it needs to calculate the distance to all the samples in TR and select the one with the lowest distance. In the big data environment, reducing runtime and increasing scalability is a very important factor, so the approximate approach is more relevant. The approximate approach can be tackled from different perspectives. Due to its high number of features, the dimensionality reduction [26] is a way to speed up the calculation of distance. The Locality-sensitive hashing algorithm [27] is a well-recognized algorithm for reducing



(a) Metric-Tree (b) Spill-Tree

Fig. 1: Partition methodology

dimensionality through hash functions, generating collisions between similar instances. This requires a previous stage of computation for the calculation of hash functions, reducing the scalability of the algorithm. When dealing with not so many features, but with a large number of instances, tree-based proposals get the best performance. In [20], the authors study tree-based approaches, and propose the Hybrid Spill-Tree algorithm (HS) [21] as the most promising algorithm to accelerate the search for the kNN.

The HS algorithm is formed of Metric-Tree (MT) with its precise search and Spill-Tree (SP) with its fast search. The MT data structure organizes the dataset in a spatial hierarchy, performing a search that ensures the exact nearest instance is found. MT is a binary tree whose root includes the entirety of the samples, and where each child represents a subset of elements. Figure 1a illustrates how to divide the elements between the two children, selecting each child as the furthest possible instance (represented by ◯). The mean distance between the children will be the separation of these nodes. The tree will have a depth of $\mathcal{O}(log(N))$. In order to search for the nearest instance, it keeps the candidate with the shortest distance C and its own distance d. If the distance to a branch is more than d, prune it and continue the search. Once there is no branch in the tree with a distance less than d, the search is finished and C and d are returned. Note that a backtracking operation is made in the structure to ensure that C is the nearest, returning exactly the nearest instance.

The SP data structure is a variation of MT, performing an approximate search to speed up its execution. The main difference compared to MT consists in sharing instances between child nodes. Figure 1b shows how data is divided with the same procedure as MT, allowing a set of duplicate instances in the child nodes. The overlapping area is dependent on the τ parameter. When τ is 0, it would be a MT with no instances shared. If τ is too high, the depth of the tree rises to infinity because the overlap is high. SP does not backtrack to ensure that the nearest instance has been found, reducing execution times. Moreover, due to the overlapping area, it obtains representative instances of the problem. A common characteristic of MT and SP is that they perform a depth-first search, computationally dependent on the number of features. Thus, when the number of features increases, the runtime is higher.

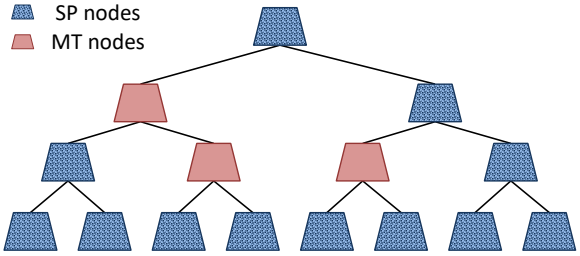


Fig. 2: Example of Hybrid-spill tree

HS is proposed with the objective of achieving a balance between accuracy and runtime. Thus, it merges the MT and SP models. To build a HS, it starts by building a SP, and if the number of instances in the overlapping area is less than the Balance Threshold (BT), it will continue to be a SP. If repeated instances exceed BT, it is reconstructed as MT. Figure 2 shows an example of HS, differentiating the MT nodes from the SP nodes. It is important to highlight the starting point for the development of this contribution, which is available in the library developed by the spark-packages community[1].

C. Apache Spark and MapReduce paradigm

The programming paradigm MapReduce [24] will be used in the development of the algorithm proposed in this paper. MapReduce aims to process large datasets through the distribution of data storage and execution through a cluster of computers.

The MapReduce implementation selected is Apache Spark [25] [28]. Spark parallelizes the calculation transparently through a distributed data structure called *Resilient Distributed Datasets* (RDD). RDDs allow data structures stored in main

[1]Hybrid Spill-Tree. https://spark-packages.org/package/saurfang/spark-knn

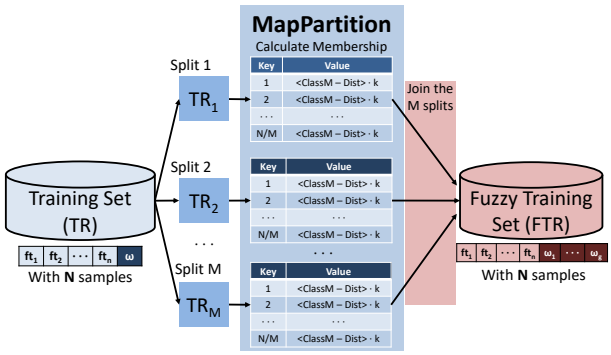


Fig. 3: Class membership stage: LHS-FkNN

memory to persist and be reused. Additionally, Spark was developed to cooperate with the distributed file system of Hadoop [29] [30] (Hadoop Distributed File System - HDFS). With this configuration, you gain the benefits provided by Spark: fault tolerance, data splitting and job communication.

MLlib [31] is the official library of machine learning in Spark. It incorporates a large number of stadistic techniques and algorithms in areas such as regression, classification, or clustering.

III. Fast and scalable FkNN classifiers for Big Data

This section presents two approximate and distributed proposals for the FkNN algorithm based on the HS method to address big data problems implemented in Spark. Two different approaches are proposed in the class membership degree stage: local and global. The local approach applies a divide-and-conquer approach, where each partition does not know the instances of the other partitions. The global approach has knowledge of all the instances of the TR and develops the use of the HS algorithm. Section III-A describes the local approach, performing the computation on each partition independently, without knowing information about the other partitions in the dataset. Section III-B presents the global approach based on HS, considering the totality of the data for the calculation of the class membership degree. Section III-C defines the classification stage, which is the same for both models and is based on the HS algorithm.

A. LHS-FkNN: Local Hybrid Spill Tree FkNN

The proposed local stage together with the classification stage is called Local Hybrid Spill Tree FkNN (LHS-FkNN). Figure 3 shows the class membership stage workflow. To alleviate the bottleneck, data is partitioned and distributed among the computation nodes. Subsequently, the membership to each partition is calculated independently. Finally, the results of each partition are joined, obtaining as output the FTR.

Algorithm 1 shows the steps and operations in Spark for calculating the class membership degree. It begins by reading the TR from HDFS and divides it into $\#Maps$ parts. Subsequently, a Spark mapPartition operation is used to calculate the class membership degree for each Training set Split (TRS_i) partition in a distributed manner. The membership calculation is represented in lines 6-12. For each y instance of each TRS_i partition, kNN is calculated and finally, the class membership degree is obtained by applying the Equation 1. Once the membership for each partition is obtained, the results are joined and form the FTR (Line 3), which will be the input of the classification stage.

Algorithm 1 Class membership degree stage - Local

Require: TR, k, $\#Maps$
1: $TRS \leftarrow$ repartition(TR, #Maps)
2: $FTRS \leftarrow$ mapPartition(computeMembership(TRP, k))
3: $FTR \leftarrow$ join($TRPD$)
4: **return** FTR
5:
6: **BEGIN computeMembership**
7: **for** y: TRP_i **do**
8: $Neigh_y \leftarrow$ computekNNLocal ($models$, k, y)
9: $membership_y \leftarrow$ computeMembership ($Neigh_y$)
10: $FTRS \leftarrow$ join(y, $membership_y$)
11: **end for**
12: **return** $FTRS$
13: **END computeMembership**

Algorithm 2 Class membership degree stage - Global

Require: TR, TS, k
1: $samples \leftarrow$ sample(TR,0, 2%)
2: $TopTree \leftarrow$ buildMT($samples$)
3: $\tau \leftarrow$ estimateτ($TopTree$)
4: $tree \leftarrow$ repartition(TR, $TopTree$, τ, $UE = 70\%$)
5: $model \leftarrow$ (broadcast($TopTree$),tree)
6: **for** y: TR **do**
7: $Neigh_y \leftarrow$ computekNN ($model$, k, y)
8: $membership_y \leftarrow$ computeMembership ($Neigh_y$)
9: $result_y \leftarrow$ join(y, $membership_y$)
10: **end for**
11: **return** $prediction_y$

B. GAHS-FkNN: Global Approximate Hybrid Spill Tree FkNN

The global stage together with the classification stage is called the Global Approximate Hybrid Spill Tree FkNN (GAHS-FkNN). Figure 4 specifies the workflow of the membership stage, which follows an approximate scheme based on HS. This approach aims to alleviate the bottleneck computation, with consideration of the data globally to obtain quality in the membership degree. Thus, this approach prioritizes quality over scalability. As in the local approach, the output from this stage is the FTR.

Algorithm 2 shows Spark's instructions for the membership degree stage with the global approach. Lines 1-5 correspond to the model creation stage based on HS, and the remaining lines correspond to the kNN and membership computation.

The model fit phase begins by reading the TR from HDFS. First, it takes a random subsample to construct a MT as described in Section II-B (the authors recommend 0.2%). This MT receives the name of top tree (TT) and is used to estimate the value of the τ parameter and partition the entire TR. The

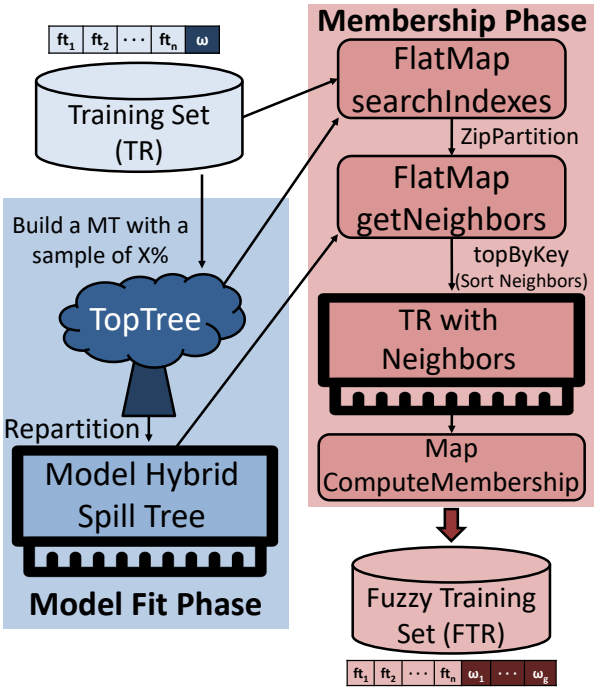


Fig. 4: Class membership phase: GAHS-FkNN

estimate of τ is the average distance between all the instances. To speed up this calculation, it is done with the TT instances.

The next step is to split the TR. To do this, the instances are distributed in the space taking as reference the TT. The value of τ defines the overlapping area. It starts building a SP, and checks if the number of instances in the overlapping area is less than 70%. Otherwise, a MT is reconstructed. When performing the search, the SP branches perform a faster search by not backtracking in the tree. However, those built as MT perform backtracking to ensure the nearest is found. The construction stage of the model ends up distributing the TT and the tree associated with the TR.

The membership phase is shown in lines 6-10. For each TR instance, kNN is calculated following the model generated. Algorithm 3 describes how to perform the kNN with native Spark operations. Using a flatMap operation, the indexes of the nearest instances of TR are computed and obtained. Thus, the distance to the right and the left nodes is calculated, and it continues the search of the nearest instance through the node with a shorter distance. When it reaches a leaf node, it returns the index of the selected instance.

With the neighbors, the class membership degree vector is calculated by Equation 1 (Line 8). The result of this phase is the FTR, and becomes the input of the classification stage.

C. Classification stage

The proposed classification stage receives as input the FTR calculated in the previous stage, the TS and the value of k. The TS is usually significantly smaller than FTR, for this reason, the classification stage has a lower computational cost than the membership stage. In order to obtain better results

Algorithm 3 Compute kNN

Require: $model$, k, x
1: $Indexes \leftarrow$ x.flatMap (searchIndexes($model.tree$))
2: $Neighs \leftarrow$ query($model.tree$, $Indexes$, k)
3: **return** $Neighs$
4:
5: **BEGIN searchIndexes**
6: $distLeft \leftarrow$ nodeLeft.dist(x)
7: $distRight \leftarrow$ nodeRight.dist(x)
8: **if** $node! = LEAF$ **then**
9: **if** $distLeft < distRight$ **then**
10: searchIndexes($nodeLeft$, ID)
11: **else**
12: searchIndexes($nodeRight$, $ID + childLeft$)
13: **end if**
14: **else**
15: **return** $Indexes$
16: **end if**
17: **END searchIndexes**

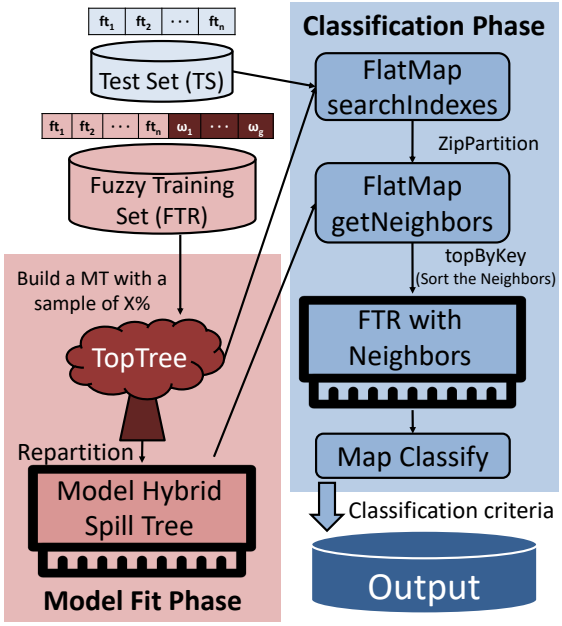


Fig. 5: Flowchart of the classification phase

in the classification, a global approach is followed, which considers all the instances for the decision making. However, it is approximate in nature in order to speed up the runtime and obtain a higher scalability. Figure 5 shows the HS-based classification stage workflow. It has two distinct phases: model fit and classification. In the first the tree is built and the instances are divided between the computation nodes. In the second the kNN of FTR is searched for and the predicted class is returned as output according to the membership degree vector.

Algorithm 4 shows the native instructions from Spark for the classification stage. Lines 1-5 correspond to the model fit phase, and the remaining lines correspond to the classification stage. Due to the similarity in the data flow with the calculation stage of class membership degree based on HS, only the

Algorithm 4 Classification Stage

Require: FTR, TS, k
1: $samples \leftarrow$ sample(FTR,0.2%)
2: $TopTree \leftarrow$ buildMT($samples$)
3: $\tau \leftarrow$ estimateτ($TopTree$)
4: $tree \leftarrow$ repartition(FTR, $TopTree$, τ, $UE = 70\%$)
5: $model \leftarrow$ (broadcast($TopTree$),tree)
6: **for** x: TS **do**
7: $NeighMemb_x \leftarrow$ computekNN ($model$, k, x)
8: $prediction_x \leftarrow$ computeMembership ($NeighMemb_x$)
9: $result_x \leftarrow$ join(x, $prediction_x$)
10: **end for**
11: **return** $prediction_y$

differences are detailed.

The first difference is in the input datasets. In this case, FTR and TS will be used. The model fit phase is not affected, since the input variables are not modified, and the distances of the instances are maintained. Thus, the model is built with the same methodology, modifying only the class label of the TR, by the membership degree vector of the FTR.

The FkNN calculation is the same as that applied in the HS-based membership calculation stage, which was described in Algorithm 3. In contrast to the membership calculation, the kNN calculation returns the membership degree vector instead of the class label (Line 7). In Line 8, it calculates the predicted class applying Equation 2, obtaining the predicted class for each TR instance as the final result.

IV. Experimental set-up

This section presents the issues involved in the experimental framework. It presents the performance measures used (Section IV-A), the details of the datasets (Section IV-B) and the algorithms used with their respective parameters (Section IV-C). Finally, the hardware and software used for the experimentation phase are specified (Section IV-D).

A. Performance measures

In this work, the efficiency and scalability of the models will be evaluated using the following metrics:

- *Accuracy:* The most widely used metric in the literature [32] [33] will be applied to evaluate the quality of the classifiers. This metric counts the number of correct classifications in relation to the total number of instances. Experimentation will be performed on classification problems with an appropriate class balance, where accuracy is a representative measure.
- *Runtime:* Time consumed in computation, also considering the readings and network communications by Spark. In addition, the runtimes will be taken for each one of the two stages that compose the fuzzy algorithms studied in order to analyze the time each of them require.

To validate the results of the experiments, we have used the pairwise non-parametric statistical tests based on Dirichlet Process called Bayesian Sign test [34]. Bayesian Sign test calculates a distribution with the differences of the results obtained from the confrontation of the two algorithms. Thus, a triangle is constructed that will determine depending on the position of the majority of the distribution, if there is a draw (rope position), victory of the first algorithm (right position) or victory of the second algorithm (left position). The statistical test and the graph shown in the experiments have been generated by the package in R called rNPBST [35].

TABLE I: Description of the datasets

Dataset	#Examples	#Features	$\#\omega$
Covtype	581,012	54	2
ECBDL14-S	2,063,187	631	2
Epsilon	500,000	2,000	2
Higgs	11,000,000	28	2
Poker	1,025,010	10	10
Susy	5,000,000	18	2
Watch-acc	3,540,962	20	7
Watch-gyr	3,205,431	20	7

TABLE II: Number of instances per map

Dataset	TR - #Instances			TS - #Instances		
	64	128	256	64	128	256
Covtype	7,262	3,631	1,816	1,816	908	454
ECBDL14-S	25,790	12,895	6,447	6,448	3,224	1,612
Epsilon	6,250	3,125	1,562	1,562	781	390
Higgs	137,500	68,750	34,375	34,376	17,188	8,594
Poker	12,812	6,406	3,203	3,204	1,602	801
Susy	62,500	31,250	15,625	15,626	7,813	3,906
Watch-acc	44,262	22,131	11,066	11,066	5,533	2,766
Watch-gyr	40,068	20,034	10,017	10,016	5,008	2,504

B. Dataset

For the experimental study, we have selected eight datasets in a large number of instances. The ECBDL14 dataset is extracted from the competition [36]. Although it has an imbalance ratio greater than 45, to study the effect of a large number of features, we selected this dataset. However, in this paper we do not address the problem of imbalance classification, so it has been sub-sampled by obtaining an imbalance ratio of two. The Epsilon dataset has been taken from the LIBSVM repository [37] and it was artificially created for the Pascal Large Scale Learning competition [38]. This dataset was selected to analyze how a high number of features affects the proposed algorithms. The other six datasets have been extracted from the UCI repository [39]. The Table I presents the number of instances, characteristics and classes ($\#\omega$). The cross-validation scheme will be followed in 5 partitions, composed of 80% training instances and the remaining 20% test instances.

In the MapReduce schema, the number of instances processed in each worker depends on the number of instances of the dataset and the number of map tasks used in the execution. The Table II shows the number of instances for TR and TS according to the number of map tasks.

C. Algorithms and parameters

The experimentation carried out has been compared with other proposals of FkNN and its crisp analogs. The algorithms used and their acronyms are presented below:

- *Global Exact FkNN (GE-FkNN)* [22]: exact model of the FkNN algorithm to tackle big data problems, obtaining

TABLE III: Accuracy comparison between algorithms

Algorithm	k	Covtype	Epsilon	ECBDL14-S	Higgs	Poker	Susy	Watch-acc	Watch-gyr	Average
GE-FkNN	3	0.9299	0.5545	-	-	0.5264	0.7338	0.9330	0.9597	0.5797
	5	0.9151	0.5452	-	-	0.5329	0.7354	0.9069	0.9398	0.5719
	7	0.9014	0.5394	-	-	0.5371	0.7320	0.8880	0.9254	0.5654
L-FkNN	3	0.9360	0.5727	0.7659	0.5954	0.5240	0.7227	0.9216	0.9506	0.7486
	5	0.9268	0.5696	0.7526	0.5984	0.5274	0.7260	0.8935	0.9284	0.7403
	7	0.9167	0.5729	0.7455	0.6000	0.5286	0.7253	0.8768	0.9148	0.7351
GAHS-FkNN	3	**0.9375**	0.5808	**0.8054**	0.5969	0.5237	0.7298	**0.9601**	**0.9808**	0.7644
	5	0.9347	0.5897	0.8046	0.6084	0.5368	0.7461	0.9576	0.9790	0.7696
	7	0.9313	0.5946	0.8013	0.6163	0.5451	0.7514	0.9558	0.9776	0.7717
LHS-FkNN	3	0.9372	0.5838	0.8034	0.6047	0.5333	0.7298	0.9566	0.9790	0.7660
	5	0.9362	0.5957	0.8025	0.6145	0.5446	0.7461	0.9544	0.9779	0.7715
	7	0.9338	**0.6066**	0.7968	**0.6214**	**0.5502**	**0.7514**	0.9530	0.9765	**0.7737**

the same results as the original FkNN. Its two stages are global and exact.

- *Local FkNN (L-FkNN)*: developed proposal of the FkNN algorithm for this contribution. The first stage, which is responsible for calculating the class membership degree, is described in Section III-A. The second stage is global and exact, identical to the used by that GE-FkNN algorithm.
- *k Nearest Neighbor - Iterative Spark (kNN-IS)* [18]: crisp kNN's exact proposal to tackle big data problems, getting the same results as the original kNN.
- *Hybrid Spill-Tree kNN (HS-kNN)* [21]: Approximate proposal of crisp kNN for big data. Although approximate, consider all instances in the search.

The best-known FkNN parameter is the number of neighbors (k) considered in the classification. k may be different in the membership and classification stages. However, for the sake of simplicity, it is kept the same in both stages. For all the algorithms used, values 3, 5 and 7 have been used. In addition, the experiments on the GAHS-FkNN model and the LHS-kNN proposal are extended using values of k from 3 to 51. The distributed component adds an extra parameter, the number of partitions or map operations. In the experiments, they take values of 64, 128 and 256.

Models based on the HS algorithm need two parameters to build the model and speed up the search for the nearest instances. The first is the percentage of instances that will be taken into account to form the TT, which is then used to divide and distribute the data. The second is the BT, the admission percentage of repetition of instances between nodes of the tree to decide if a ST or a MT is constructed. The study has taken the optimal values recommended by the authors of HS: TT equal to 0.2% and BT equal to 70%.

D. Hardware and software used

All experiments have been performed on a cluster composed of 15 nodes: a master node and 14 computation nodes. All the nodes have the same configuration:

- Processor: Intel Xeon CPU E5-2620 (2 GHz) x2.
- Cores: 12 threads (6 cores).
- RAM: 64 GB.
- Network: 40 Gb/s Infiniband.
- Cache: 15 MB.

All nodes have the same software set-up:

- Operative System: CentOS 6.5.
- Apache Spark version: 2.2.1.
- Scala version: 2.11.6.
- Hadoop Distributed File System: Version 2.6.0-cdh5.8.0.

With this software and hardware configuration, there is a maximum of 256 concurrent map tasks available. Thus, there are approximately 2GB of main memory for each of these map tasks.

V. Analysis of results

In this section, we study the results compiled from different experimental studies. Specifically, we analyze the following points:

- First, we establish a comparison between the proposals and the state-of-the-art FkNN algorithms in terms of accuracy and runtime. (Section V-A)
- Second, we performed a scalability study on the successful proposals. To do this we will focus on the runtimes. (Section V-B)
- Third, we extend the two most promising models to higher values of k by focusing on accuracy. (Section V-C)
- Fourth, we compared the results obtained for each algorithm and dataset against the crisp kNN proposals similar to the fuzzy models studied. (Section V-D)

A. Accuracy study

The accuracy study starts by showing the results from Table III, which compares the accuracy between the algorithms in relation to the number of neighbors (k) and the number of map operations equal to 128 for all datasets. The best result for each dataset and the best average result are highlighted in bold. Those values that could not be executed due to scalability problems are represented with the symbol "-", for the calculation of the mean, they are considered as a zero in accuracy.

Figure 6 presents the probability distribution of the differences between the GAHS-FkNN and LHS-FkNN algorithms obtained with the Bayesian Sign Test.

Figure 7 shows the membership stage and the classification stage runtimes in seconds, for each dataset, algorithms and values of k equal to 3, 5 and 7.

Analyzing the table and figures presented, we can observe:

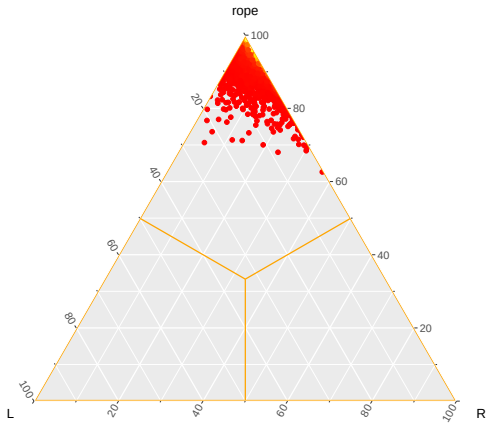


Fig. 6: Heatmap of the Bayesian Sign Test: GAHS-FkNN vs LHS-FkNN

- The GE-FkNN algorithm finds its scalability limit in its first stage, not being able to run for the ECBDL14-S and Higgs datasets.
- Models based on HS (GAHS-FkNN and LHS-FkNN) achieve better results than models with the exact classification stage (GE-FkNN and L-FkNN). This may be due to the approximate component of the subjascent model HS, which obtains a noise tolerance of the datasets in the GAHS-FkNN and LHS-FkNN algorithms, as the exact component of kNN can lead to a high overfit to the training set.
- Regarding the influence of the k, the GE-FkNN and L-FkNN algorithms do not obtain an appreciable improvement, with a stagnation of the results in accuracy. However, the GAHS-FkNN and LHS-FkNN algorithms display an increase in accuracy. For this reason, the two most promising algorithms will be studied for higher k values. In relation to the runtime, it should be noted that the value of k affects the GE-FkNN and L-FkNN models to a certain extent, whereas it does not drastically affect the proposed GAHS-FkNN and LHS-FkNN algorithms.
- Figure 6 shows how the GAHS-FkNN and LHS-FkNN algorithms are statistically equal in accuracy, although on average, the LHS-FkNN algorithm is slightly better than GAHS-FkNN. For this reason, it is important to analyze the scalability of the models in relation to the number of map operations used.

B. Scalability study

The scalability study starts by presenting the results from Figure 8, which compares the runtime of the membership stage and the runtime of the classification stage in seconds, for each datasets, with values of k equal to 3, 5 and 7 and number of maps equal to 64, 128 and 256.

According to the figure shown:

- The GAHS-FkNN algorithm is affected when dealing with a large number of features. This is due to the HS structure generating trees with a high depth in their branches as it has a high number of features, resulting in higher runtimes. This can be observed in the Epsilon and ECBDL14-S datasets, where the runtime obtained by LHS-FkNN are much faster than the runtimes for the GAHS-FkNN algorithm.
- LHS-FkNN scale depending on the number of maps thanks to its first local stage, obtaining a performance associated with the hardware used.
- GAHS-FkNN gets good runtimes without significantly affecting the hardware used, showing interesting behavior but limiting the scalability of the model.
- If we focus on the runtime of the classification stage, it is shared by GAHS-FkNN and LHS-FkNN as both models follow the same scheme on this stage.

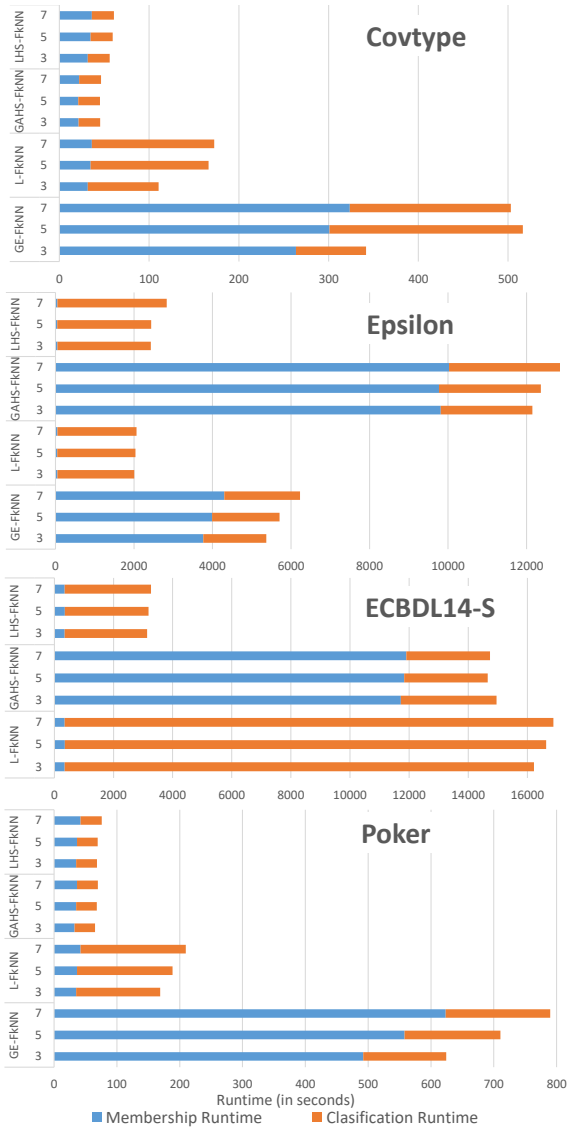
C. Study for higher values of k

According to the results shown, the GE-FkNN and L-FkNN algorithms show a stagnation in accuracy with the values of k set to 3, 5 and 7. In addition, the runtime is not drastically affected by the k. However, the GAHS-FkNN and LHS-FkNN algorithms keep improving the results. For this reason, this section extends the values of k up to 51, studying the accuracy obtained by both algorithms and the 8 datasets, setting the number of maps to 128.

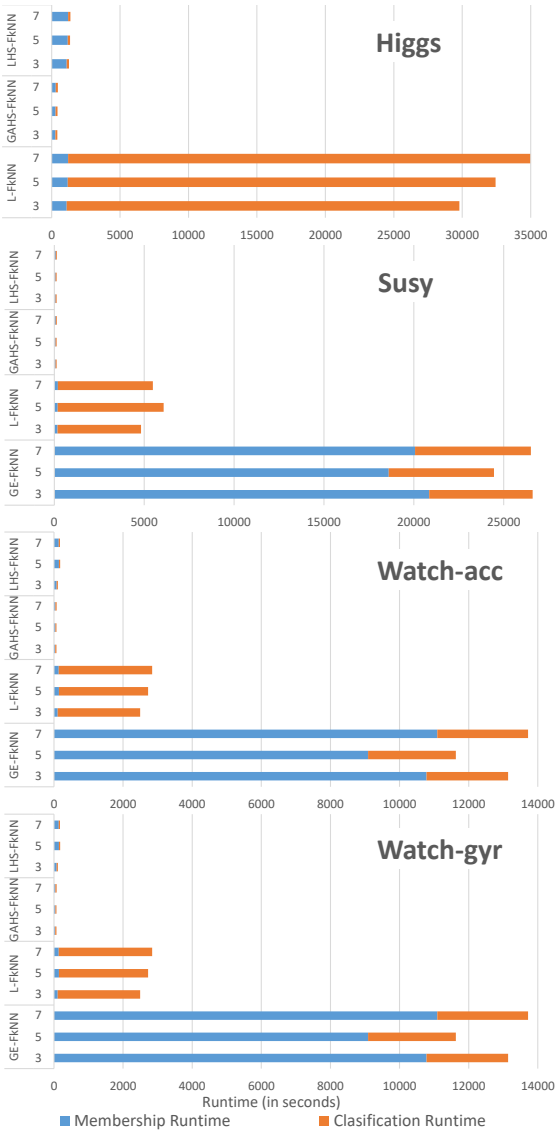
Figure 9 presents the accuracy obtained for each dataset with the GAHS-FkNN and LHS-FkNN algorithms, with values of k between 3 and 51. In order to facilitate the visualization of the results, two figures are shown due to the differences in accuracy between the datasets.

According to the figures presented we can observe:

- The results obtained (accuracy) according to the value of k follow a similar behavior pattern for both algorithms. Focusing on the datasets, high values of k improve the accuracy in the Higgs, Poker, Epsilon and Susy datasets. However, low values of k improve accuracy for Covtype, Watch-acc, Watch-gyr and ECBDL14-S datasets. This is a natural behavior for the FkNN algorithm, which occurs with classical datasets from the literature. Therefore, the proposed algorithms show the same behavior in large datasets.
- Comparing GAHS-FkNN and LHS-FkNN in terms of accuracy, we see that the differences are very low, and when this difference is accentuated somewhat more in the Covtype and Epsilon datasets, LHS-FkNN is the clear winner.



(a) Datasets: Covtype, Epsion, ECBDL14-S and Poker



(b) Datasets: Higgs , Susy, Watch-acc and Watch-gyr

Fig. 7: Runtime comparison between algorithms: GAHS-FkNN, LHS-FkNN, GE-FkNN and L-FkNN

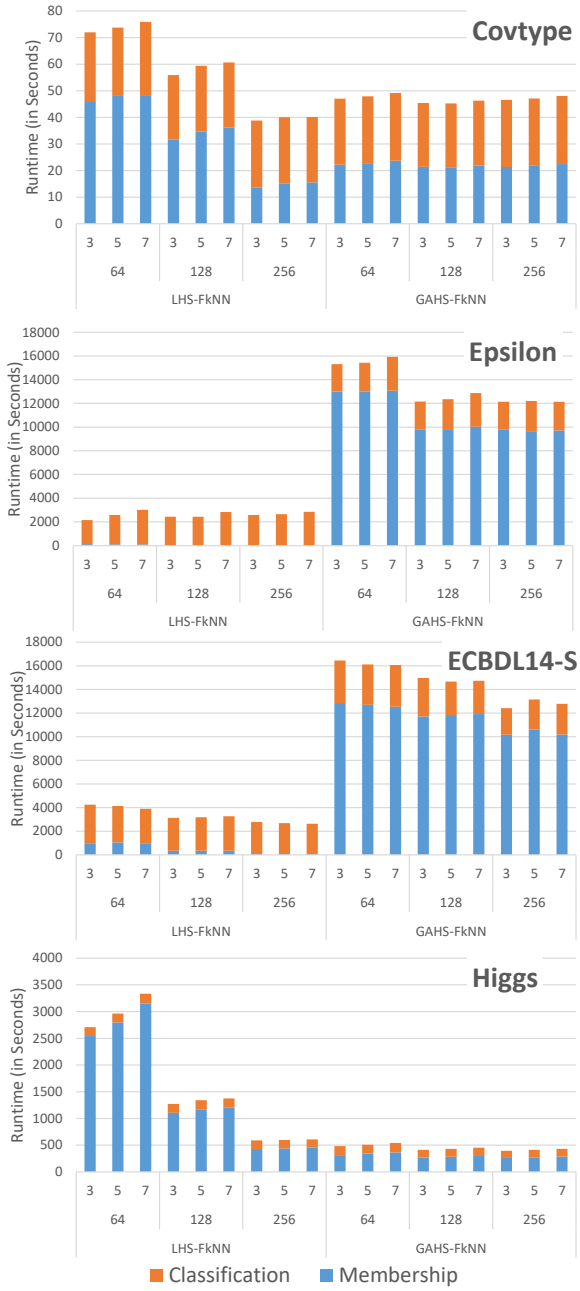
D. Comparison with crisp kNN algorithms

As the inuence of the number of maps has already been analyzed and not considered significant, this experiment has been set to 128 maps in order to focus on the comparative study of crisp vs fuzzy.

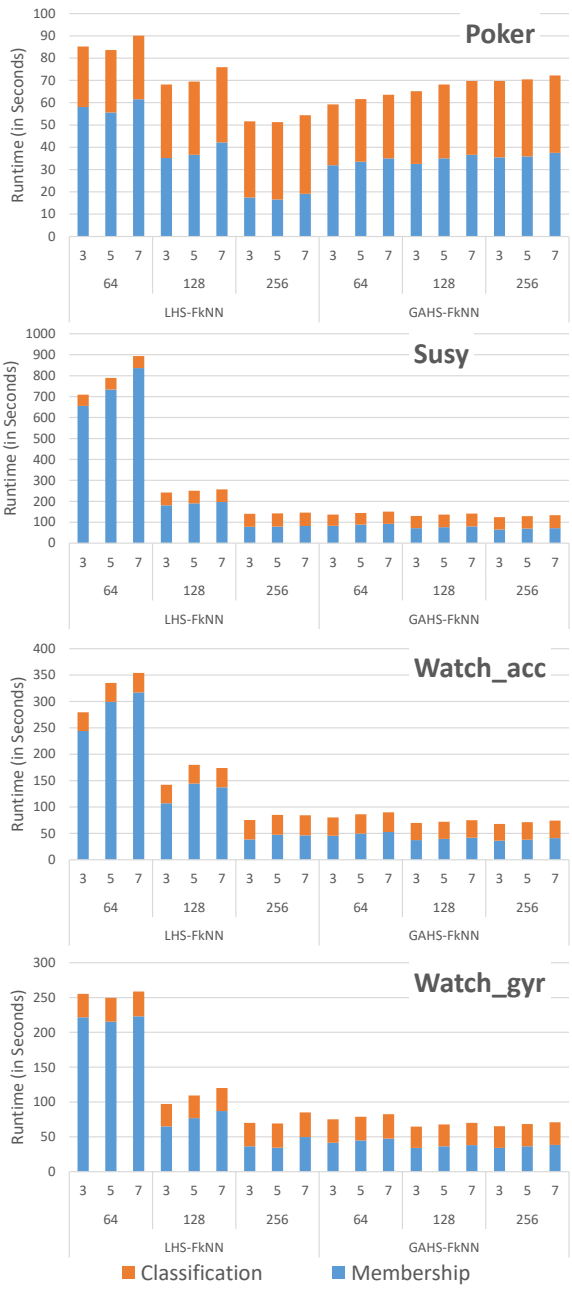
Figure 10 shows the total runtime for the algorithms kNN-IS, HS-kNN, GAHS-FkNN and LHS-kNN. To facilitate the study of the runtime, it is presented only with the value of k = 5. The results are shown for two figures due to the differences in the scales of the total runtime of each dataset.

Table IV shows a comparison between the result obtained by the two proposed algorithms and the two crisp-alternatives, exploring the values of k 3, 5 and 7.

According to the table and figure presented, it can be seen that the best results are obtained by the proposed algorithms. Although HS-kNN improves with respect to the kNN-IS algorithm, it is always less accurate than the FkNN models, without the runtime being excessively increased due to the optimization carried out in the classification stage of the GAHS-FkNN algorithm. kNN-IS wins in the Epsilon, Watch-acc and Watch-gyr datasets, possibly because it is a dataset with clearly differentiated boundaries and low noise, where the classification problem is simpler than in the other datasets. Despite this, on average the proposed fuzzy models are clearly better.



(a) Datasets: Covtype, Epsilon, ECBDL14-S and Higgs

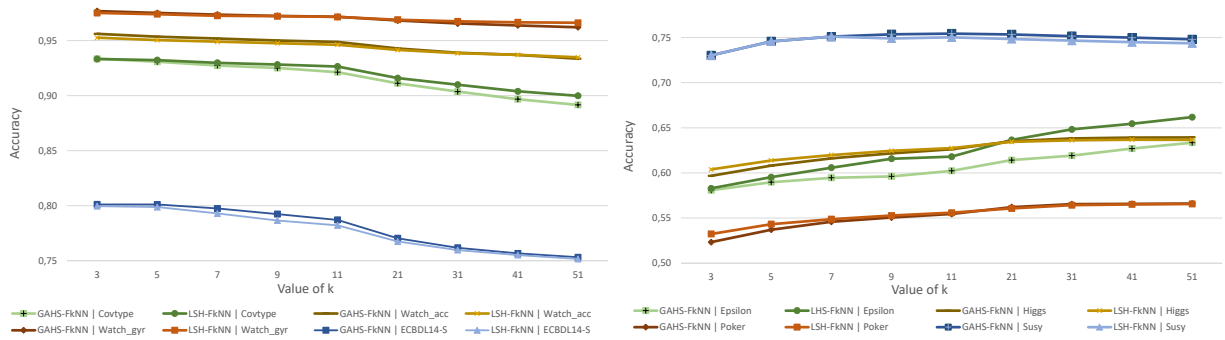


(b) Datasets: Poker, Susy, Watch-acc and Watch-gyr

Fig. 8: Scalability comparison between GAHS-FkNN and LHS-FkNN

TABLE IV: Crisp vs Fuzzy models: accuracy

Algoritm	k	Covtype	Epsilon	ECBDL14-S	Higgs	Poker	Susy	Watch-acc	Watch-gyr	Average
kNN-IS	3	0.9371	0.5864	0.7833	0.5454	0.4758	0.6675	**0.9647**	**0.9845**	0.7431
	5	0.9367	0.6007	0.7797	0.5458	0.4952	0.6784	0.9613	0.9811	0.747
	7	0.9326	**0.6110**	0.7683	0.5559	0.4937	0.6861	0.9582	0.9788	0.7481
HS-kNN	3	0.9308	0.5847	0.8020	0.5885	0.5201	0.7223	0.9542	0.9755	0.7598
	5	0.9232	0.5981	0.8017	0.5936	0.5305	0.7360	0.9478	0.9698	0.7626
	7	0.9161	0.6086	0.7986	0.5981	0.5369	0.7431	0.9423	0.9651	0.7636
GAHS-FkNN	3	**0.9375**	0.5808	**0.8054**	0.5969	0.5237	0.7298	0.9601	0.9808	0.7644
	5	0.9347	0.5897	0.8046	0.6084	0.5368	0.7461	0.9576	0.9790	0.7696
	7	0.9313	0.5946	0.8013	0.6163	0.5451	**0.7514**	0.9558	0.9776	0.7717
LHS-FkNN	3	0.9372	0.5838	0.8034	0.6047	0.5333	0.7331	0.9566	0.9790	0.7664
	5	0.9362	0.5957	0.8025	0.6145	0.5446	0.7446	0.9544	0.9779	0.7713
	7	0.9338	0.6066	0.7968	**0.6214**	**0.5502**	0.7480	0.9530	0.9765	**0.7733**



(a) Datasets: Covtype, ECBDL14-S, Watch-acc and Watch-gyr

(b) Datasets: Epsilon, Higgs, Poker and Susy

Fig. 9: Accuracy comparison with higher values of k: GAHS-FkNN vs LHS-kNN

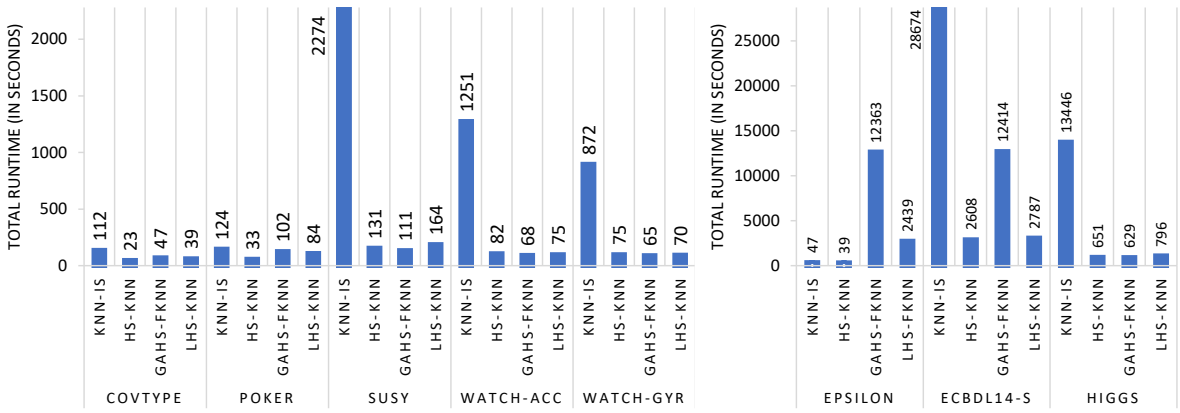


Fig. 10: Total Runtime comparison between Crisp and Fuzzy models

VI. Conclusions and further work

In this paper, two MapReduce approaches have been proposed to speed up the FkNN algorithm in Big Data problems. Because of the design and use of big data technologies, it is possible to execute with very large datasets. In order to study any possible improvements, the proposed model has also been compared with Fuzzy and Crisp versions of the literature. The GAHS-FkNN and LHS-FkNN algorithms achieve statistically equal results in terms of accuracy. On the one hand, the LHS-FkNN algorithm demonstrates very high scalability depending on the hardware facilities available, as well as high accuracy results. On the other hand, the GAHS-FkNN algorithm is less dependent on hardware resources but is more affected by a high number of features.

Thus, the use of LHS-FkNN is recommended when we have powerful hardware according to the problem we want to address, and if the number of features is high. The use of GAHS-FkNN is recommended when the number of features is not too high and we have hardware limitations.

A library has been generated with the algorithms used in this study and is available in the spark-packages platform at https://spark-packages.org/package/JMailloH/HS_FkNN.

As future work, we aim to tackle the class imbalanced problem through evolutionary undersampling techniques [40], capable of handling large datasets.

Acknowledgments

This contribution has been supported by the Spanish National Research Project TIN2017-89517-P. J. Maillo hold a FPU scholarship from the Spanish Ministry of Education.

References

[1] J. M. Keller, M. R. Gray, and J. A. Givens, "A fuzzy k-nearest neighbor algorithm," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-15, no. 4, pp. 580–585, July 1985.

[2] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[3] J. Derrac, S. García, and F. Herrera, "Fuzzy nearest neighbor algorithms: Taxonomy, experimental analysis and prospects," *Information Sciences*, vol. 260, pp. 98 – 119, 2014.

[4] Z. Cai, J. Gu, C. Wen, D. Zhao, C. Huang, H. Huang, C. Tong, J. Li, and H. Chen, "An intelligent parkinsons disease diagnostic system based on a chaotic bacterial foraging optimization enhanced fuzzy knn approach," *Computational and Mathematical Methods in Medicine*, vol. 2018, p. 24, 2018.

[5] J. Qin, L. Wang, and R. Huang, "Research on fault diagnosis method of spacecraft solar array based on f-knn algorithm," in *2017 Prognostics and System Health Management Conference (PHM-Harbin)*, July 2017, pp. 1–4.

[6] S. García, J. Luengo, and F. Herrera, *Data preprocessing in data mining*. Springer, 2015.

[7] J. M. Cadenas, M. C. Garrido, R. Martínez, E. Muñoz, and P. P. Bonissone, "A fuzzy k-nearest neighbor classifier to deal with imperfect data," *Soft Computing*, vol. 22, no. 10, pp. 3313–3330, May 2018.

[8] S. Ezghari, A. Zahi, and K. Zenkouar, "A new nearest neighbor classification method based on fuzzy set theory and aggregation operators," *Expert Systems with Applications*, vol. 80, pp. 58 – 74, 2017.

[9] I. Banerjee, S. S. Mullick, and S. Das, "On convergence of the class membership estimator in fuzzy k-nearest neighbor classifier," *IEEE Transactions on Fuzzy Systems*, pp. 1–1, 2018.

[10] H. Nikdel, Y. Forghani, and S. Mohammad Hosein Moattar, "Increasing the speed of fuzzy k-nearest neighbours algorithm," *Expert Systems*, vol. 35, no. 3, p. e12254, 2018.

[11] J. Derrac, F. Chiclana, S. García, and F. Herrera, "Evolutionary fuzzy k-nearest neighbors algorithm using interval-valued fuzzy sets," *Information Sciences*, vol. 329, pp. 144 – 163, 2016.

[12] P. H. Kassani, A. B. J. Teoh, and E. Kim, "Evolutionary-modified fuzzy nearest-neighbor rule for pattern classification," *Expert Systems with Applications*, vol. 88, pp. 258 – 269, 2017.

[13] N. Biswas, S. Chakraborty, S. S. Mullick, and S. Das, "A parameter independent fuzzy weighted k-nearest neighbor classifier," *Pattern Recognition Letters*, vol. 101, pp. 80 – 87, 2018.

[14] S. John Walker, *Big data: A revolution that will transform how we live, work, and think*. Taylor & Francis, 2014.

[15] O. U. Lenz, D. Peralta, and C. Cornelis, "A scalable approach to fuzzy rough nearest neighbour classification with ordered weighted averaging operators," in *International Joint Conference on Rough Sets, June*, June 2019.

[16] R. Talavera-Llames, R. Pérez-Chacón, A. Troncoso, and F. Martínez-Álvarez, "Big data time series forecasting based on nearest neighbours distributed computing with spark," *Knowledge-Based Systems*, vol. 161, pp. 12–25, 2018.

[17] I. Triguero, D. García-Gil, J. Maillo, J. Luengo, S. García, and F. Herrera, "Transforming big data into smart data: An insight on the use of the k-nearest neighbors algorithm to obtain quality data," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 9, no. 2, p. e1289, 2019.

[18] J. Maillo, S. Ramírez, I. Triguero, and F. Herrera, "kNN-IS: an iterative spark-based design of the k-Nearest Neighbors classifier for big data," *Knowledge-Based Systems*, vol. 117, no. Supplement C, pp. 3 – 15, 2017, volume, Variety and Velocity in Data Science.

[19] J. K. Uhlmann, "Satisfying general proximity / similarity queries with metric trees," *Information Processing Letters*, vol. 40, no. 4, pp. 175 – 179, 1991.

[20] T. Liu, A. W. Moore, K. Yang, and A. G. Gray, "An investigation of practical approximate nearest neighbor algorithms," in *Advances in neural information processing systems*, 2005, pp. 825–832.

[21] T. Liu, C. J. Rosenberg, and H. A. Rowley, "Performing a parallel nearest-neighbor matching operation using a parallel hybrid spill tree," Jan. 6 2009, uS Patent 7,475,071.

[22] J. Maillo, J. Luengo, S. García, F. Herrera, and I. Triguero, "Exact fuzzy k-nearest neighbor classification for big datasets," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, July 2017, pp. 1–6.

[23] J. Maillo, J. Luengo, S. Garca, F. Herrera, and I. Triguero, "A preliminary study on hybrid spill-tree fuzzy k-nearest neighbors for big data classification," in *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, July 2018, pp. 1–8.

[24] J. Dean and S. Ghemawat, "Mapreduce: A flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[25] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and 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*. USENIX Association, 2012, pp. 1–14.

[26] S. Dutta and A. K. Ghosh, "On some transformations of high dimension, low sample size data for nearest neighbor classification," *Machine Learning*, vol. 102, no. 1, pp. 57–83, Jan 2016.

[27] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, Oct 2006, pp. 459–468.

[28] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning spark: lightning-fast big data analysis*. " O'Reilly Media, Inc.", 2015.

[29] T. White, *Hadoop: The Definitive Guide*, 3rd ed. O'Reilly Media, Inc., 2012.

[30] "Apache Hadoop Project," Accessed on Dec. 2018. [online]. [Online]. Available: http://hadoop.apache.org/

[31] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, "Mllib: Machine learning in apache spark," *Journal of Machine Learning Research*, vol. 17, no. 34, pp. 1–7, 2016.

[32] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[33] E. Alpaydin, *Introduction to Machine Learning*, 3rd ed. The MIT Press, 2014.

[34] A. Benavoli, G. Corani, J. Demšar, and M. Zaffalon, "Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 2653–2688, 2017.
[35] J. Carrasco, S. García, M. del Mar Rueda, and F. Herrera, "rNPBST: an r package covering non-parametric and bayesian statistical tests," in *Hybrid Artificial Intelligent Systems*. Springer, 2017, pp. 281–292.
[36] "ECBDL14 dataset: Protein structure prediction and contact map for the ECBDL2014 big data competition," 2014. [Online]. Available: http://cruncher.ncl.ac.uk/bdcomp/
[37] "Epsilon in the LIBSVM website, 2019," Accessed on May. 2019. [online]. [Online]. Available: http://www.csie.ntu.edu.tw/cjlin/libsvmtools/datasets/binary.html#epsilon
[38] "Pascal large scale learning challenge," 2008. [Online]. Available: http://largescale.ml.tu-berlin.de
[39] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml
[40] S. García and F. Herrera, "Evolutionary undersampling for classification with imbalanced datasets: Proposals and taxonomy," *Evolutionary computation*, vol. 17, no. 3, pp. 275–306, 2009.

Jesus Maillo received the B.Sc. and M.Sc. degrees in computer science from the University of Granada, Granada, Spain, in 2014 and 2015. He is currently pursuing the Ph.D. degree in the Department of Computer Science and Artificial Intelligence in the University of Granada.

His research interests include data mining, data preprocessing and big data.

Salvador García received the B.S. and Ph.D. degrees in Computer Science from the University of Granada, Granada, Spain, in 2004 and 2008, respectively. He is currently an Associate Professor in the Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain. Dr. García has published more than 90 papers in international journals (more than 60 in Q1), h-index 44, over 60 papers in international conference proceedings (data from Web of Science). He has organized several special sessions and workshops related to data preprocessing and evolutionary learning in conferences such as Hybrid Intelligent Systems, Intelligent Systems Design and Applications and International Joint-Conference of Neural Networks. He has been associated with the international program committees and organizing committees of several regular international conferences including IEEE CEC, ICPR, ICDM, IJCAI, etc.

As edited activities, he is an associate editor of Information Fusion (Elsevier), Swarm and Evolutionary Computation (Elsevier) and AI Communications (IOS Press) journals. He is a co-author of the books entitled Data Preprocessing in Data Mining and Learning from Imbalanced Data Sets published by Springer. His research interests include data science, data preprocessing, Big Data, evolutionary learning, Deep Learning, metaheuristics and biometrics.

Dr. García has been given some awards and honors for his personal work or for his publications in and conferences, such as IFSA-EUSFLAT 2015 Best Application Paper Award and IDEAL 2015 Best Paper Award. He belongs to the list of the Highly Cited Researchers in the area of Computer Sciences (2014-2018): http://highlycited.com/ (Clarivate Analytics).

Julián Luengo received the M.S. degree in computer science and the Ph.D. from the University of Granada, Granada, Spain, in 2006 and 2011 respectively.

He currently acts as an Associate Professor in the Department of Computer Science and Artificial Intelligence at the University of Granada, Spain. His research interests include machine learning and data mining, data preparation in knowledge discovery and data mining, missing values, noisy data, data complexity and fuzzy systems.

Dr. Luengo has been given some awards and honors for his personal work or for his publications in and conferences, such as IFSA-EUSFLAT 2009 Best Student Paper Award. He belongs to the list of the Highly Cited Researchers in the area of Computer Sciences (2018): http://highlycited.com/ (Clarivate Analytics).

Francisco Herrera (SM'15) received his M.Sc. in Mathematics in 1988 and Ph.D. in Mathematics in 1991, both from the University of Granada, Spain. He is a Professor in the Department of Computer Science and Artificial Intelligence at the University of Granada and Director of the Andalusian Research Institute in Data Science and Computational Intelligence (DaSCI). He's an academician at the Spanish Royal Academy of Engineering.

He has been the supervisor of 45 Ph.D. students. He has published more than 400 journal papers, receiving more than 73000 citations (Scholar Google, H-index 135). He has been selected as a Highly Cited Researcher http://highlycited.com/ (in the fields of Computer Science and Engineering, respectively, 2014 to present, Clarivate Analytics).

He currently acts as Editor in Chief of the international journal "Information Fusion" (Elsevier). He acts as editorial member of a dozen of journals.

He received the several honors and awards, among others: ECCAI Fellow 2009, IFSA Fellow 2013, 2010 Spanish National Award on Computer Science ARITMEL to the "Spanish Engineer on Computer Science", International Cajastur "Mamdani" Prize for Soft Computing (Fourth Edition, 2010), IEEE Transactions on Fuzzy System Outstanding 2008 and 2012 Paper, 2011 Lotfi A. Zadeh Prize Best paper Award (IFSA Association), 2013 AEPIA Award to a scientific career in Artificial Intelligence, 2014 XV Andaluca Research Prize Maimnides, 2017 Andaluca Medal (by the regional government of Andaluca), 2018 Granada: Science and Innovation City award.

His current research interests include among others, Computational Intelligence (including fuzzy modeling, computing with words, evolutionary algorithms and deep learning), information fusion and decision making, and data science (including data preprocessing, prediction and big data).

Isaac Triguero received his M.Sc. and Ph.D. degrees in Computer Science from the University of Granada, Granada, Spain, in 2009 and 2014, respectively. He is currently an Assistant Professor of Data Science at the University of Nottingham since June 2016. His work is mostly concerned with the research of novel methodologies for big data analytics. Dr Triguero has published more than 65 international publications in the fields of Big Data, Machine Learning and Optimisation (H-index=22 and more than 1900 citations on Google Scholar). He is a Section Editor-in-Chief of the Machine Learning and Knowledge Extraction journal, and an associate editor of the Big Data and Cognitive Computing journal, and the IEEE Access journal. He has acted as Program Co-Chair of the IEEE Conference on Smart Data (2016), the IEEE Conference on Big Data Science and Engineering (2017), and the IEEE International Congress on Big Data (2018). Dr Triguero is currently leading a Knowledge Transfer Partnership project funded by Innovative UK and the energy provider E.ON that investigates Smart Metering data.