A preliminary study on Hybrid Spill-Tree Fuzzy k-Nearest Neighbors for big data classification

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 Fuzzy k Nearest Neighbor (Fuzzy kNN) classifier is well known for its effectiveness in supervised learning problems. kNN classifies by comparing new incoming examples with a similarity function using the samples of the training set. The fuzzy version of the kNN accounts for the underlying uncertainty in the class labels, and it is composed of two different stages. The first one is responsible for calculating the fuzzy membership degree for each sample of the problem in order to obtain smoother boundaries between classes. The second stage classifies similarly to the standard kNN algorithm but uses the previously calculated class membership degree. To deal with very large datasets, distributed versions of the Fuzzy kNN algorithm have been proposed. However, existing approaches remain not fully scalable as they aim to replicate the exact behavior of the Fuzzy kNN. In this work, we present an approximate and distributed Fuzzy kNN approach based on Hybrid Spill-Tree implemented under Apache Spark. The aim of this model is to alleviate the scalability problems and to deal with big datasets maintaining high accuracy. In our experiments, we compare in precision and runtime with the Fuzzy kNN for big data problems existing in the literature, running with datasets of up to 11 million instances. The results show an improvement in the runtime and accuracy with respect to the previous exact model.**

I. Introduction

The algorithm of the k Nearest Neighbors (kNN) [1] belongs to the family of instance-based classifiers. To classify, it compares unseen samples with the labeled instances of the training set. The kNN algorithm does not create a classification model, and it postpones the computation until the classification stage. It classifies by similarity, usually with a distance function (which used to be the Euclidean or Manhattan distance). In addition to its simplicity, the kNN classifier stands as one of the top ten most relevant classification algorithms [2].

However, kNN does not distinguish between the selected neighbors, affecting all the neighbors equally in the classification, accepting that all boundaries are perfectly defined between the classes. There are different proposals that alleviate this problem through the use of fuzzy sets. In [3], the classical Fuzzy kNN algorithm [4] highlights as one of the best performing approaches. Notice a novel Evolutionary Fuzzy kNN algorithm [5], which improves on the previous ones as an evolutionary proposal for small groups. We are going to focus on classical Fuzzy kNN algorithm because of its computational efficiency. This algorithm performs two different stages. First, it switches the class label for a class membership degree regarding of the k nearest neighbors. After that, it computes the kNN with the membership degree knowledge. Thus, it get softer edges getting better at kNN in most classification problems. In addition, Fuzzy kNN is present in current applications in different fields such as medicine [6] or prediction of landslides [7].

The kNN and Fuzzy kNN algorithms face two main problems to handle big datasets because of postponing the whole computation until the classification stage. These are the runtime and the memory consumption. Fuzzy kNN is even more affected, since the membership calculation stage requires the double of main memory than kNN because it calculates kNN with the training set against itself.

To alleviate these two problems we can make use of cloud-based technologies. Specifically, the MapReduce [8] paradigm and the Spark framework [9]. It outstand by his distributed file system and the fault tolerance mechanism.

Based on MapReduce and Spark, we currently find an exact proposal of the kNN algorithm that can deal with large-scale problems, reporting the same results as the original kNN, and it is called k Nearest Neighbor - Iterative Spark (kNN-IS) [10]. Its workflow has two stages: the map stage and the reduce stage. The first stage splits the training set and calculates the kNN of each test example. In the reduce stage, all candidates are grouped and after that, the k closest according to their distance are selected. Thus, it is allowed to address large training and test sets, obtaining good execution times while maintaining the results of the original kNN algorithm.

However, in the big data environment, it is interesting to pursue an approximation of the algorithm to accelerate the execution times, since the high number of samples usually causes redundancy in the data, making the approximate algorithms less affected than in classic problems. As examples of approximated kNN we find the LSH algorithm [11], Metric-Tree [12] or Spill-Tree. In [13], the authors study these models and show that Hybrid Spill-Tree (HST) is the best for their runtime and results obtained.

Regarding Fuzzy kNN to tackle big data problems, we can find a proposal of the algorithm that obtains the same result as the original version. It is called Exact Fuzzy k Nearest Neighbors (EF-kNN) [14], which has two phases: class membership degree phase and classification phase. The first calculates the class membership degree. It computes the kNN with the same MapReduce-scheme that kNN-IS does.

Once we have the kNN, the class label is changed by a class membership degree vector. The second stage classifies the unseen samples with the class membership degree information, also following the kNN-IS workflow. Although it is capable of scaling up to very large data sets, the execution times for calculating the class membership degree are substantially high.

In this preliminary contribution, we propose a fuzzy variation of the Hybrid Spill-Tree-based Nearest Neighbor Search and we will call through the text Hybrid Spill-Tree Fuzzy kNN (HSTF-kNN). It has been developed in Spark, taking advantage of its in-memory primitive to address large datasets by dividing the data and distribute the computation between the different compute nodes. As we explained briefly, Fuzzy kNN has 2 stages: class membership degree and classification. The first stage starts partitioning the data and generating the HST structure. Then, it computes the kNN and changes the label with a vector of membership degree. The second stage is responsible for the classification. It performs the same steps as the first one, with the difference that it will calculate the k nearest neighbors for each sample of the test set according to the membership degree. Finally, it will decide the predicted class according to the membership degree of each neighbor.

In summary, the main contributions of this work are as follow:

- Design and develop an approximate model of Fuzzy kNN based on Hybrid Spill-Tree taking advantage of the Apache Spark framework.
- A experimental study of the scalability and accuracy of this model and a comparison against EF-kNN algorithm.

The remainder of this contribution is organized as follows. Section II introduces the state-of-art in Fuzzy kNN, the Hybrid Spill-Tree algorithm and big data technologies. Then, Section III details the proposed Hybrid Spill-Tree 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 Hybrid Spill-Tree (Section II-B). The big data technologies used are described in Section II-C.

A. *Fuzzy k-Nearest Neighbors algorithm and complexity*

The Fuzzy kNN algorithm [4] is proposed as an improvement of the kNN algorithm. It has been shown to improve kNN and other Fuzzy approaches in terms of accuracy. It needs a pre-calculation stage on the training set, which consists of calculating the class membership degree. Then, it calculates kNN for each unseen sample. A formal notation for the Fuzzy kNN algorithm is the following:

Let TR be a training set and TS a test set, composed of several instances $\mathbf{n}$ and $\mathbf{t}$ respectively. Each instance $\mathbf{x}_i$ is a vector $(\mathbf{x}_{i1}, \mathbf{x}_{i2}, \mathbf{x}_{i3}, \dots, \mathbf{x}_{ij})$, where $\mathbf{x}_{ij}$ is the value it takes for the i-th instance and j-th feature. Every sample of TR belong to a known class ω. However, for TS it is unknown.

Fuzzy kNN has two stages: class membership degree and classification. The first stage calculates the k nearest neighbors of TR against itself, maintaining a leave-one-out scheme. For that, calculate the distance of each sample and select the k closest. Then, create the class membership following the Equation 1. The output of the first stage will be TR changing the class label ω, for the class membership (ω_1, ω_2, …, ω_l) where l is the number of classes. This new set will be called Fuzzy Training Set, FTR.

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

The classification stage computes the kNN as described in the first stage. However, the kNN algorithm is computed for each sample of TS in the FTR and it is obtained the class membership degree. After that, it decides the resulting class as the Equation 2.

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

The Fuzzy kNN method improves in accuracy the kNN algorithm in most classification problems. However, this supposes an increase of the algorithmic complexity which involves two problems to be addressed in the area of big data:

- Runtime: The complexity of calculating kNN from a single sample is $\mathcal{O}(n \cdot F)$ where n is the number of training instances and F the number of features. For multiple neighbors, the algorithmic complexity increases to $\mathcal{O}(n \cdot log(N))$. In addition, we must remember that the first stage executes kNN on TR against itself. In the second stage calculates kNN of the TS versus FTR.
- Memory consumption: Fuzzy kNN needs to store in the main memory the TR and the TS to accelerate the computation. If both sets are large, they might easily exceed the available main memory.

These difficulties lead us to design an approximate model based on Hybrid Spill Tree, obtaining a distributed model of Fuzzy kNN and using the MapReduce paradigm and the Spark platform as big data technologies.

B. *Hybrid Spill-Tree search*

The Hybrid Spill-Tree algorithm [15] is an approximate proposal to calculate the kNN algorithm in a distributed way. It partitions the search space with two types of tree structures: Metric-Tree and Spill-Tree. Therefore, it is necessary to briefly describe these models.

Metric-Tree (MT) is a data structure that organizes a set of points through a spatial hierarchy. It is a binary tree where the root node contains all the elements and each child node represents a set of points. Figure 1 shows how to split the data into the right and left children. Each child represents a subset of samples of the parent node, taken as far apart as possible (at the figure, the sample represented with circles). MT ensures

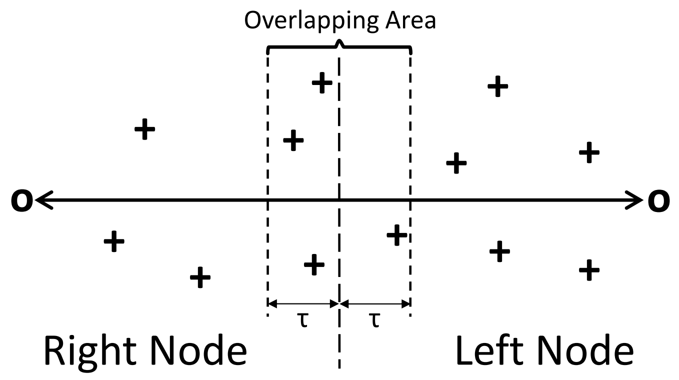


Fig. 1: Partitioning in a Metric-Tree and Spill-Tree

that the children of a node are two disjoint sets, that is, they do not have repeated instances (represented by symbols $+$) in their child nodes. The overlapping area does not exist for MT. Therefore, each leaf node contains very few samples. The tree will take a depth of $\mathcal{O}(log(N))$. To perform the search, it keeps a candidate to be the closest Nearest Neighbor (NN), and its distance d. If the distance to a branch is greater than d, it prunes this branch and continues searching. After analyzing the whole structure, it returns the NN and d. However, the most of the runtime is consumed in making sure that it is selected the NN, performing backtracking if it is necessary. Therefore, Spill-Tree emerges to speed up the computation.

Spill-Tree (SP) is a variant of a Metric-Tree in which the child of one node can "spill over" on another child node and share samples. In other words, the split criterion of SP enables two child nodes to have repeated instances. Figure 1 presents how the data is divided with the same procedure as MT, but an overlapping area according to the distance value τ is allowed. The samples inside the overlapping area are shared for the right and left children. If τ is 0, it is a MT. However, if the parameter τ is too high, the overlap between nodes is high and the depth of the tree tends to $\mathcal{O}(log(\infty))$. Due to the overlapping area, it can sacrifice not having the exact NN. To do this, backtracking is avoided in the search and an approximate NN is obtained with faster runtimes than MT.

Hybrid Spill-Tree appears with the aim of obtaining high accuracy and even lower runtimes. To achieve this, it merges both models. Selecting the use of MT when required to warrant the NN. and in another case, speed up the execution time with SP. It defines a Balance Threshold (BT), which is a percentage that usually takes the value of 70%. To build the tree structure, start by forming a SP, and if the number of samples that are repeated at the nodes is greater than BT, the tree for those nodes is rebuilt as MT and will be marked as non-overlapping. In the other case, the tree for those nodes is kept as SP and will be marked as overlapping. The search for the nearest neighbors is also done in a hybrid way. Backtracking will be done on the nodes marked as non-overlapping, which are of the MT type. Backtracking will not be done on the nodes marked as overlapping, which is of the SP type.

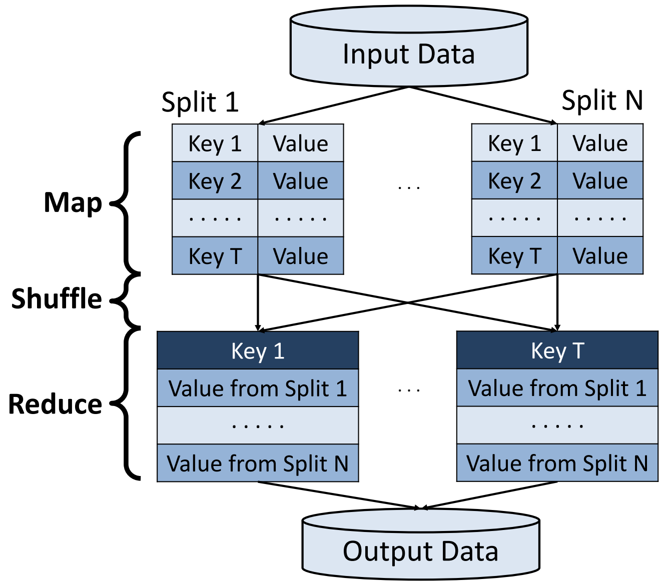


Fig. 2: MapReduce data flow overview

Notice the implementation available in the free software repository accessible in spark-package[1], that is our starting point in the development stage.

C. MapReduce programming model: Apache Spark

For the development of the proposed algorithm in this paper, the programming paradigm MapReduce [16] will be used. This paradigm was designed by Google in 2003 and it is a scalable data processing tool. Its aim is to process big datasets by distributing storage and execution through a group of machines.

The MapReduce paradigm has three stages: Map, Shuffle and Reduce. Map stage reads the dataset in form of <key-value> pairs, and it distributes through the nodes for parallel computation. The Shuffle is responsible for merging all the values with the same key. Finally, the Reduce stage combines those coincident pairs and it aggregates it into smaller <key-value> pairs. The MapReduce scheme of this process is represented in Figure 2. In [17], authors expose an exhaustive review of this framework and other distributed paradigms.

Apache Spark [9] is a novel implementation of MapReduce that parallelizes the computations in a transparent way through a distributed data structure called Resilient Distributed Datasets (RDDs). In addition, RDDs allow us to persist and reuse data, cached in memory. Recently, Spark has incorporated a new distributed data structure called DataFrame. It maintains the advantages of the RDDs, incorporating optimization improvements and allowing SQL queries on the data. We incorporate DataFrame in our implementation. Moreover, Spark was developed to cooperate with the distributed file

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

Algorithm 1 Membership Degree Stage

```
Require: $TR$, $k$
 1: $sampled \leftarrow$ sample($TR$,0.2%)
 2: $TopTree \leftarrow$ buildMetricTree($sampled$)
 3: $\tau \leftarrow$ estimateTau($TopTree$)
 4: $trees \leftarrow$ repartition($TR$, $TopTree$, $\tau$, $BT = 70\%$)
 5: $model \leftarrow$ (Broadcast($TopTree$),trees)
 6: for $y$: $TR$ do
 7:     $Neigbors_y \leftarrow$ computekNN ($model$, $k$, $y$)
 8:     $Membership_y \leftarrow$ computeMembership ($Neigbors_y$)
 9:     $result_y \leftarrow$ join($y$, $Membership_y$)
10: end for
11: return $result$
```

system of Apache Hadoop[2] (Hadoop Distributed File System). With this configuration, it can take advantage of the data splitting, fault-tolerance and job communication provided by the Spark framework.

Spark includes a scalable machine learning library called MLlib[3]. It has a multitude of machine learning algorithms and statistical techniques from different areas of KDD such as classification, regression, clustering or data preprocessing.

III. HSTF-kNN: Hybrid Spill-Tree Fuzzy-kNN for Big Data Classification

In this section, we present an approximate and distributed proposal of the Fuzzy k Nearest Neighbor algorithm based on the structure and search of the Hybrid Spill-Tree method to address big data problems using Spark. We will make use of the speed and effectiveness of the Hybrid Spill-Tree algorithm in comparison with the exact and distributed kNN algorithm (kNN-IS). The goal is to design and develop a fuzzy variant that improves Hybrid Spill-Tree accuracy and EF-kNN runtimes. The general scheme of our proposal has the same two stages as the Fuzzy kNN model: Membership stage (Section III-A) and Classification stage (Section III-B).

A. Class membership degree stage

This section explains the workflow process that calculates the class membership degree. Figure 3 shows the flowchart of the HSTF-kNN, dividing the computation into two parts: Model Fit phase and Membership phase. The first forms the tree-based structure and divides the data between the compute nodes. The second searches for the kNN of each TR sample and computes the class membership degree. The output is the TR changing the class label by the class membership degree vector. It will be called Fuzzy Training Set (FTR).

Algorithm 1 outlines the steps of the membership stage. In it, Lines 1-5 correspond to the Model Fit stage, and the remaining lines describe the calculation of the membership phase.

The Model fit phase starts by reading the TR from HDFS as a DataFrame data type from Spark. First of all, it gets a random sub-sample (the authors recommend the 0.2% of samples) to build a MetricTree, as described in Section II-B.

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

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

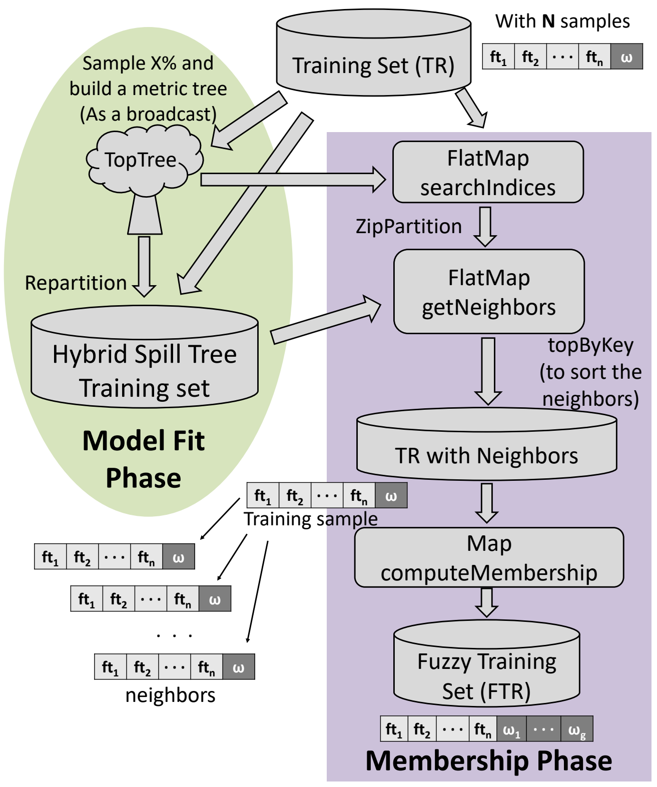


Fig. 3: Flowchart of the class membership phase

This MetricTree is the TopTree (TT) and is used to estimate the value of the parameter τ and split all the TR. A good estimate of the value of τ should get the nearest neighbors with high accuracy. Therefore, the appropriate value is the average distance between the samples. To accelerate the estimate of τ, it will be done only with the sample of TT, so as not to make the calculation with the whole TR.

The next step is to repartition the TR, taking TT as a reference to distributing the instances in the space. τ will be key to define the overlapping area of instances between the nodes. The authors recommend a value of 70% ($BT < 70\%$) to discern if a node will be Metric-Tree or Spill-Tree. It starts by forming a SP, and it checks if $BT < 70\%$, it is marked as overlapping, and the search will not perform backtracking. In another case, a MT is re-built and marked as non-overlapping, performing backtracking in the search. Performing backtracking involves going back into the tree to make sure that another branch does not contain any closer instances. If the closest instance of that node is larger than the one we have already found, this child is pruned. The Model Fit stage ends by broadcasting TT and distributing the HST structure.

The Membership stage is shown from lines 6 to 10. For each sample of TR, the k nearest neighbors are calculated using the HST. Algorithm 2 describes how the search of the neighbors is done through the tree structures. With a flatMap operation, the indexes of the k nearest neighbors for each element of the

Algorithm 2 Compute kNN

```
Require: $model$, $k$, $x$
 1: $Indexes \leftarrow$ x.flatMap ( searchIndexes($model.tree$) )
 2: $Neighbor \leftarrow$ Query($model.tree$, $Indexes$, $k$)
 3: return $Neighbors$
 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($nodeRight$, $ID + leftChild$)
13:     end if
14: else
15:     return $Indexes$
16: end if
17: END searchIndexes
```

Algorithm 3 Classification Stage

```
Require: $FTR$, $TS$, $k$
 1: $sampled \leftarrow$ sample($FTR$,0.2%)
 2: $TopTree \leftarrow$ buildMetricTree($sampled$)
 3: $\tau \leftarrow$ estimateTau($TopTree$)
 4: $trees \leftarrow$ repartition($FTR$, $TopTree$, $\tau$, $BT = 70\%$)
 5: $model \leftarrow$ (Broadcast($TopTree$),trees)
 6: for $x$: $TS$ do
 7:     $NeigborsMemb_x \leftarrow$ computeFuzzykNN ($model$, $k$, $x$)
 8:     $Prediction_x \leftarrow$ computePrediction ($NeigborsMemb_x$)
 9:     $result_x \leftarrow$ join($x$, $Prediction_x$)
10: end for
11: return $result$
```

TR are calculated. To do this, it calculates the distance to the right and left nodes, and continue through the node with the shortest distance. When it reaches a leaf node, it returns the index of the neighbor found.

Once we have the neighbors, we change the class label with the vector of class membership degree by applying the Equation 1 (Line 8). The output of this phase is the TR, changing the class label by a class membership degree vector, receiving the name of Fuzzy Training Set, FTR.

B. Classification stage

This section describes the steps to calculate fuzzy kNN. Figure 4 presents the flowchart of the classification stage, which is also divided into two phases: Model Fit phase and Classification phase. The workflow of the Class membership degree and Classification stages are very similar. For this reason, this section will be focused on presenting the main differences between them.

Algorithm 2 describes the steps during the classification stage. We will specify the differences with respect to the previous stage since the main scheme is the same and only slight details of the data structure and the final result are affected.

The first difference is found in the input sets. In this case, FTR and TS will be used. The use of FTR does not affect the Model Fit phase, since the change with respect to TR affects the class label, which has been replaced by a class membership degree vector. This does not disturb the distances of the samples and the model is built with the same method.

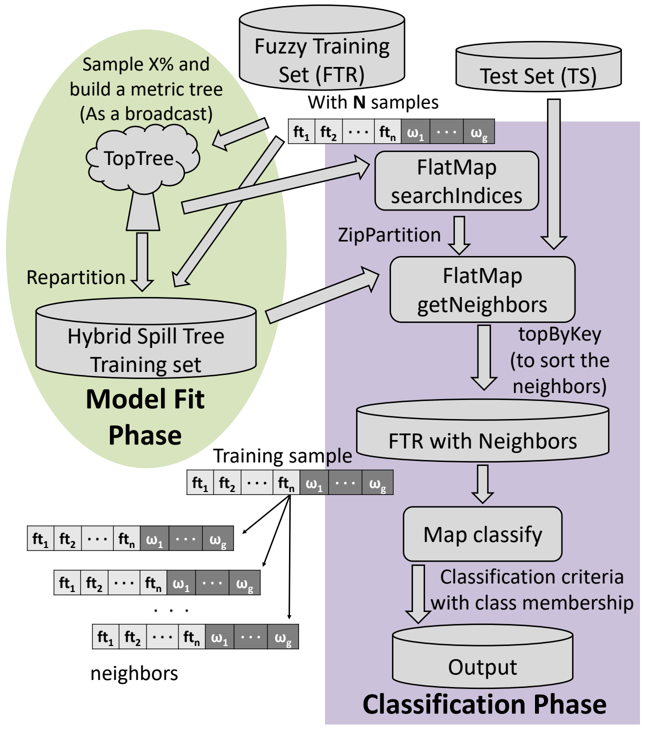


Fig. 4: Flowchart of the classification phase

The calculation of Fuzzy kNN is done in the same way, with the difference that instead of returning the class label for each neighbor, class membership degree is returned (Line 7). Line 8 presents a difference with respect to the previous stage. It performs the Compute Prediction operation, which will apply the Equation 2 to decide the predicted class label. The final output of the HSTF-kNN algorithm is the predicted class for each sample of TS.

IV. Experimental Framework and Preliminary 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.

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

A. Experimental set-up

For this experimental study, we have selected three large datasets from the UCI machine learning repository [18] to evaluate our model: PokerHand, Susy and Higgs. Table I presents the number of samples, features, and classes ($\#\omega$). In our experiments, we follow a 5 fold cross-validation scheme.

TABLE II: Influence of the k value

Dataset	k	Membership Runtime (in seconds)		Classification Runtime (in seconds)		Accuracy	
		EF-kNN	HSTF-kNN	EF-kNN	HSTF-kNN	EF-kNN	HSTF-kNN
Poker	3	397.1361	35.4365	114.0275	34.3035	0.5257	0.5233
	5	444.1799	35.9263	128.8338	34.5083	0.5316	0.5371
	7	503.1163	37.5005	143.8841	34.6973	0.5338	0.5459
Susy	3	11521.7479	64.8935	3956.3381	59.1125	0.7301	0.7302
	5	15853.2564	69.4081	4023.5556	59.8742	0.7306	0.7457
	7	16934.5319	72.0451	4026.3059	61.2118	0.7268	0.7510
Higgs	3	-	261.6500	-	131.5073	-	0.5968
	5	-	273.5462	-	137.8146	-	0.6081
	7	234431.8347	288.7636	17137.3216	140.5870	0.5904	0.6162

TABLE III: Influence of the number of maps

Dataset	#Maps	Membership Runtime (in seconds)		Classification Runtime (in seconds)		Accuracy	
		EF-kNN	HSTF-kNN	EF-kNN	HSTF-kNN	EF-kNN	HSTF-kNN
Poker	128	623.4500	36.6495	166.2626	33.1551	0.5371	0.5451
	256	503.1163	37.5005	143.8841	34.6973	0.5338	0.5459
Susy	128	20071.0198	80.2774	6446.3527	61.2445	0.7320	0.7514
	256	16934.5319	72.0451	4026.3059	61.2118	0.7268	0.7510
Higgs	128	-	296.8639	-	154.6719	-	0.6163
	256	234431.8347	288.7636	17137.3216	140.5870	0.5904	0.6162

This means that each partition includes 80% of training samples and the remaining samples will from the test set.

To evaluate the efficiency and scalability of the proposed algorithms, we used the following measures:

- *Accuracy:* It represents the number of unseen samples correctly classified against the total number of them. This metric shows the performance of the algorithm and is the measure most commonly used for assessing the performance of classification problems [19].
- *Total Runtime:* This measure collects the total time spent by the algorithms. The runtime includes reading and distributing the dataset over the compute cluster and also the two stages of the Fuzzy kNN.

The most well-known parameter for the original Fuzzy-kNN algorithm is the number of neighbors (k) to be considered for classifying. k could be different at each stage, but we will keep both equal for simplicity. In our experiments, parameter k is set to 3, 5 and 7. An extra parameter is needed due to its distributed behavior. This is the number of partitions of the training set, which matches the number of map tasks. In this preliminary experimentation, we will run with 128 and 256 map operations.

HSTF-kNN has two extra parameters: Sample to build the Top Tree and the Balance Threshold. The first one is the number of samples to build the TT, used to split and distribute the data. The second is the BT. It is a percentage that specifies the intersection to decide if the tree constructed for each split will be a Spill-Tree or a Metric-Tree. For this preliminary study, we will focus on the recommended values, TT equal to 0.2% and BT equal to 70%.

All the experiments have been run on a cluster composed of 14 computing nodes managed by the master node. All the nodes have the same configuration. They have 2 Intel Xeon CPU E5-2620 processor, 6 cores (12 threads) per processor, 2 GHz and 64 GB of RAM. The network is Infiniband 40Gb/s. This hardware was configured providing a maximum number of current tasks to 256. 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 2.2.1.

B. Results and discussion

This study compares the results obtained by our preliminary HSTF-kNN approach and the EF-kNN model. We will explore different aspects of the proposed method, such as the influence of the k parameter, how it affects the number of maps operations and its behavior compared to the exact version. For this purpose, the three datasets (Poker, Susy and Higgs), described in the experimental setup will be used. Due to the large number of instances given by the Higgs datasets, the EF-kNN algorithm cannot be run in a bounded time. For this reason, the experiment in this dataset will be limited to k = 7 and 256 map tasks.

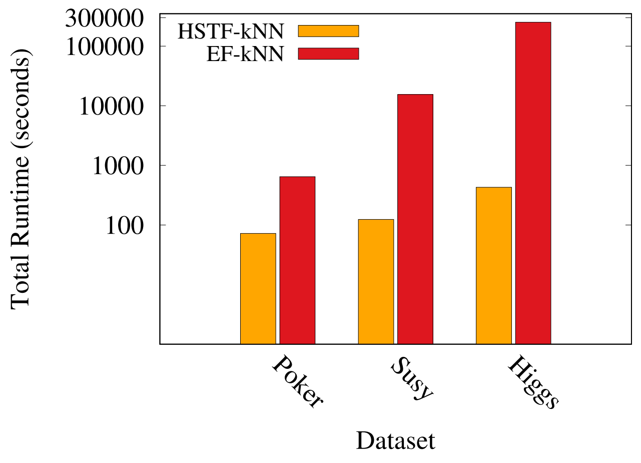
Table II shows the Membership and Classification Runtime (in seconds) and the Accuracy, with the Poker and Susy datasets. The objective is to analyze the influence of the value of k in HSTF-kNN algorithm and compare it with EF-kNN. Thus, the number of maps will be set to 256.

Table III presents the Accuracy and Membership and Classification Runtime (in seconds), with Poker and Susy datasets. We select the same value of k equal to 7, since with this value we obtain the best accuracy, enabling us to focus on the analysis of the influence of the number of maps (#Maps).

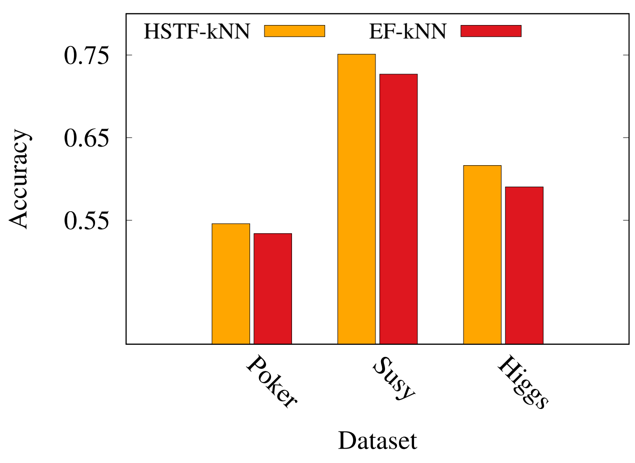
Figure 5 shows a comparison with Poker, Susy and Higgs. The Figures 5a and 5b present the Total Runtime (in seconds) and the Accuracy respectively. The number of maps is set to 256 and the number of neighbor to 7.

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

- Regarding k parameter and according to the Table II, the total runtime does not increase too much, despite the increase of the network traffic and the calculation of the neighbors, due to the design performed in both models. Although HSTF-kNN is an approximate version,



(a) Total Runtime



(b) Accuracy

Fig. 5: Total Runtime and Accuracy with all datasets

it obtains better results than EF-kNN and as the value of k increases, an improvement in accuracy is appreciated.

- We can observe in Table III how the number of maps does not affect drastically the runtime in the algorithm HSTF-kNN. However, EF-kNN is highly affected, doubling the time in relation to hardware features that we have. Regarding accuracy, there is a slight change depending on the number of maps in both methods. This is caused by some test examples that are identical to several training data points. When it is a distributed execution, the definitive neighbors will depend on the order in which they arrive from each map output.
- According to Figures II and III, it can be seen how the execution time of the Membership stage is higher than in the Classification stage. EF-kNN presents a clear bottleneck, while HSTF-kNN manages it better, obtaining much shorter times due to its model adjustment phase and the use of Hybrid Spill-Tree.
- Analyzing Figure 5a, the runtime on Higgs dataset is pretty high, and reveals a weakness of the EF-kNN algorithm. However, HSTF-kNN obtains lower runtime that demonstrates its scalability potential. This potential makes us pay special attention to the precision in the figure 5b. In this Figure, we see how HSTF-kNN improves the accuracy of EF-kNN in all the datasets of the experimental study.

V. Conclusions and further work

In this contribution, we have developed a scalable and distributed model for the Fuzzy kNN algorithm based on Hybrid Spill-Tree on the Spark framework. The biggest contribution is to improve the scalability of the existing method, EF-kNN, also adding an improvement in accuracy. In addition, k and the number of maps does not affect drastically the accuracy or the runtime. As future work, we will focus on the bottleneck, the membership calculation stage. We will try to reduce the runtime and the memory consumption, without having a decrement in accuracy.

Acknowledgment

This work has been supported by the projects TIN2017-89517-P, 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. Derrac, S. García, F. Herrera, Fuzzy nearest neighbor algorithms: Taxonomy, experimental analysis and prospects, Information Sciences 260 (2014) 98 – 119.

[4] 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.

[5] J. Derrac, F. Chiclana, S. García, F. Herrera, Evolutionary fuzzy k-nearest neighbors algorithm using interval-valued fuzzy sets, Information Sciences 329 (2016) 144 – 163, special issue on Discovery Science. doi:https://doi.org/10.1016/j.ins.2015.09.007.

[6] 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.

[7] D. Tien Bui, Q. P. Nguyen, N.-D. Hoang, H. Klempe, A novel fuzzy k-nearest neighbor inference model with differential evolution for spatial prediction of rainfall-induced shallow landslides in a tropical hilly area using gis, Landslides 14 (1) (2017) 1–17. doi:10.1007/s10346-016-0708-4.

[8] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Communications of the ACM 51 (1) (2008) 107–113.

[9] 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.

[10] J. Maillo, S. Ramírez, I. Triguero, F. Herrera, kNN-IS: an iterative spark-based design of the k-Nearest Neighbors classifier for big data, Knowledge-Based Systems 117 (Supplement C) (2017) 3 – 15, volume, Variety and Velocity in Data Science. doi:https://doi.org/10.1016/j.knosys.2016.06.012.

[11] A. Andoni, 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), 2006, pp. 459–468. doi:10.1109/FOCS.2006.49.

[12] J. K. Uhlmann, Satisfying general proximity / similarity queries with metric trees, Information Processing Letters 40 (4) (1991) 175 – 179. doi:https://doi.org/10.1016/0020-0190(91)90074-R.

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

[14] J. Maillo, J. Luengo, S. García, F. Herrera, I. Triguero, Exact fuzzy k-nearest neighbor classification for big datasets, in: 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2017, pp. 1–6. doi:10.1109/FUZZ-IEEE.2017.8015686.

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

[16] J. Dean, S. Ghemawat, Map reduce: A flexible data processing tool, Communications of the ACM 53 (1) (2010) 72–77.

[17] 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.

[18] M. Lichman, UCI machine learning repository (2013).
URL http://archive.ics.uci.edu/ml

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