Un enfoque aproximado para acelerar el algoritmo de clasificación Fuzzy kNN para Big Data

Jesus Maillo, Julián Luengo, Salvador García, Francisco Herrera
Dept. de Ciencias de la Computación e Inteligencía Artificial
Universidad de Granada. 18071, Granada, España
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

***Resumen*—El clasificador difuso de los k vecinos más cercanos (Fuzzy k Nearest Neighbor - Fuzzy kNN) es reconocido por su eficacia en problemas de aprendizaje supervisado. El algoritmo kNN clasifica comparando una nueva instancia con los datos etiquetados mediante una función de similitud. Su versión difusa se compone de dos etapas. La primera etapa calcula el grado de pertenencia a cada clase para cada instancia del problema, con el objetivo de detectar con mayor precisión las fronteras entre clases. La segunda etapa clasifica siguiendo el mismo esquema que el algoritmo kNN, aunque haciendo uso del grado de pertenencia a clase previamente calculado. La propuesta existente de Fuzzy kNN para afrontar conjuntos de datos muy grandes no es completamente escalable, debido a que replica el comportamiento de Fuzzy kNN original. Con el objetivo de agilizar este algoritmo, en esta contribución se propone un enfoque aproximado que acelera los tiempos de ejecución sin perder calidad en la clasificación, mediante el uso de árboles e implementado en Spark. En la experimentación realizada se comparan varios enfoques de Fuzzy kNN para big data, con conjuntos de datos de hasta 11 millones de instancias. Los resultados muestran una mejora en tiempo de ejecución y precisión con respecto al modelo de la literatura.**

***Palabras Clave*—Conjuntos difusos, K-vecinos más cercanos, Clasificación, MapReduce, Apache Spark, Big Data**

I. Introducción

El algoritmo de los k vecinos más cercanos (*k Nearest Neighbor* - kNN) [1] es catalogado como clasificador basado en instancias. Para clasificar, compara las instancias no vistas con aquellas etiquetadas del conjunto de entrenamiento utilizando una función de similitud. Generalmente la similitud es medida mediante una función de distancia (Euclídea o Manhattan). A pesar de su simplicidad, el clasificador kNN es uno de los diez algoritmos de clasificación más relevantes [2].

Sin embargo, kNN suponiendo todos los vecinos igual de importantes en la clasificación, considerando que las fronteras entre clases están perfectamente definidas. Existen diferentes propuestas basadas en la teoría de los conjuntos difusos que abordan este problema. En [3], el algoritmo clásico Fuzzy kNN [4] destaca como uno de los enfoques más eficaces. Fuzzy kNN se compone de dos etapas: cálculo de grado de pertenencia y clasificación. La primera etapa, cambia la etiqueta de la clase por un grado de pertenencia a cada clase, de acuerdo a las instancias más cercanas. La segunda etapa, calcula el kNN con la información del grado de pertenencia. Así, se consigue detectar con mayor precisión las fronteras, viéndose menos afectado por el ruido y mejorando al kNN en la mayoría de los problemas de clasificación.

En el ámbito del big data, tanto el algoritmo kNN como Fuzzy kNN encuentran problemas para manejar grandes conjuntos de datos con respecto al tiempo de ejecución y al consumo de memoria. Existe una propuesta exacta del algoritmo kNN para abordar problemas big data y se denomina *k Nearest Neighbor - Iterative Spark* (kNN-IS) [5]. Además de esta versión exacta, también existen versiones aproximadas que reducen los tiempos de ejecución: Metric-Tree [6] y Spill-Tree. En [7], los autores estudian estos modelos y proponen el modelo Hybrid Spill-Tree (HS) como hibridación de ambos con el objetivo de mejorar el tiempo de ejecución sin afectar a los resultados debido a la redundancia de datos implícita en big data. Por otro lado, existe una solución exacta para aplicar Fuzzy kNN en big data, *Global Exact Fuzzy k Nearest Neighbors* (GE-FkNN) [8]. Aunque es capaz de escalar hasta conjuntos de datos muy grandes, los tiempos de ejecución de la primera etapa son sustancialmente altos, provocando un cuello de botella. Sin embargo, no se han aplicado enfoques aproximados para el algoritmos Fuzzy kNN en big data.

En este trabajo se propone una variación aproximada del algoritmo Fuzzy kNN desarrollado en Spark. Para aliviar el cuello de botella de la primera etapa del algoritmo Fuzzy kNN, se estudiarán dos enfoques: local y global. El enfoque local consiste en dividir el conjunto de datos en diferentes partes y calcular el grado de pertenencia a clase internamente en cada partición (operación map), sin considerar otras particiones. El enfoque global está basado en el modelo HS. Éste genera un árbol con las instancias del conjunto de entrenamiento y lo distribuye entre todos los nodos de cómputo, considerando todas las instancias para el cálculo del grado de pertenencia. La segunda etapa es responsable de la clasificación y común para las dos propuestas. En ésta, el esquema de trabajo es igual a la versión global de cálculo de pertenencia, pero considera la pertenencia previamente calculada clase para predecir. Para comprobar el rendimiento de este modelo, los experimentos realizados se han llevado a cabo en 4 conjuntos de datos con hasta 11 millones de instancias. El estudio experimental analiza la precisión y tiempo de ejecución realizando una comparativa con otros algoritmos de la literatura.

El documento está estructurado de la siguiente manera. La

sección II introduce el estado del arte en los algoritmos Fuzzy kNN y Hybrid Spill-Tree. La sección III detalla la propuesta aproximada del algoritmo Fuzzy kNN. El estudio experimental se describe en la Sección IV. La sección V concluye el documento y esboza el trabajo futuro.

II. Preliminares

Esta sección proporciona el conocimiento básico del algoritmo Fuzzy kNN (Sección II-A), el Hybrid Spill-Tree (Sección II-B) y las tecnologías big data utilizadas (Sección II-C).

II-A. k vecinos más cercanos difuso y su complejidad

El algoritmo Fuzzy kNN [4] se propone como una mejora del algoritmo kNN, llegando a mejorarlo en términos de precisión para la mayoría de los problemas de clasificación. Necesita una etapa de pre-cómputo en el conjunto de entrenamiento, que calcula el grado de pertenencia a la clase. Posteriormente, calcula kNN para cada instancia no vista y decide la clase predicha con el grado de pertenencia. Una notación formal para el algoritmo Fuzzy kNN es la siguiente:

Sea CE un Conjunto de Entrenamiento y CP un Conjunto de Prueba, compuestos de **n** y **t** instancias respectivamente. Cada instancia $\mathbf{x}_i$ es un vector $(\mathbf{x}_{i1}, \mathbf{x}_{i2}, \mathbf{x}_{i3}, \ldots, \mathbf{x}_{ij})$, donde $\mathbf{x}_{ij}$ es el valor de la i-ésima instancia y j-ésima característica. Para cada instancia de CE se conoce su clase ω. Sin embargo, para las instancias de CP la clase es desconocida.

Fuzzy kNN posee dos etapas: cálculo de pertenencia y clasificación. La primera etapa calcula los k vecinos más cercanos para cada instancia del CE, manteniendo un esquema *leave-one-out* seleccionando las k instancias con una distancia menor. Finalmente, calcula el grado de pertenencia de acuerdo a la Ecuación 1. El resultado de la primera etapa será CE modificando la etiqueta de clase ω, por un vector de pertenencia a cada clase (ω_1, ω_2, …, ω_l) donde l es el número de clases. Este nuevo conjunto se llamará Conjunto de Entrenamiento Difuso, CED.

$$u_j(x) = \begin{cases} 0{,}51 + (n_j/k_{pert}) \cdot 0{,}49 & if \quad j = i \\ (n_j/k_{pert}) \cdot 0{,}49 & if \quad j \neq i \end{cases} \tag{1}$$

La etapa de clasificación calcula para cada instancia de CP sus kNN como se describe en la primera etapa, para cada instancias de CP en el CED y se obtiene el grado de pertenencia a clase. Posteriormente, obtiene la clase resultante de acuerdo a la Ecuación 2.

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

La etapa extra respecto a kNN y el aumento de la complejidad computacional generan dos problemas big data:

- Tiempo de ejecución: La complejidad de calcular kNN para una instancia es $mathcalO(n \cdot c)$, donde n es el número de instancias de CE y c el número de características. Para más de un vecino, aumenta a $\mathcal{O}(n \cdot log(N))$. Además, Fuzzy kNN tiene una etapa extra de cómputo para el cálculo de grado de pertenencia.
- Consumo de memoria: Para acelerar el cálculo, se requiere disponer de los conjuntos CE y CP almacenados en memoria principal. Cuando ambos conjuntos son grandes, es fácil exceder la memoria principal disponible.

Para aliviar estas dificultades, trabajamos en el diseño de un modelo aproximado basado en Hybrid Spill-Tree desarrollado bajo las tecnologías big data de MapReduce y Spark.

II-B. Hybrid Spill-Tree

El algoritmo Hybrid Spill-Tree (HS) [9] es una propuesta aproximada para calcular el algoritmo kNN de forma distribuida. HS se compone de dos estructuras: Metric-Tree y Spill-Tree. La estructura de datos Metric-Tree (MT) organiza el conjunto de datos en una jerarquía espacial. Es un árbol binario cuya raíz contiene todos los elementos, y donde cada hijo representa un subconjunto de elementos. La Figura 1 ilustra como dividir los elementos entre los dos hijos, seleccionando cada hijo como las instancias más lejanas posibles (representados por ◯). Los nodos hijos de un MT no contienen instancias repetidas (representadas por $+$). El árbol tendrá una profundidad de $\mathcal{O}log(N))$. Para realizar la búsqueda de la instancia más cercana, se conserva al candidato con menor distancia C y su distancia d. Si la distancia a una rama es superior a d, la poda y continúa la búsqueda. Una vez no existe una rama del árbol con distancia menor que d, se finaliza la búsqueda y se devuelve C y d. Sin embargo, invierte tiempo y cómputo en asegurar que C es el más cercano, volviendo atrás en el árbol si fuese necesario. Con el objetivo de reducir el tiempo de búsqueda emerge Spill-Tree.

La estructura de datos Spill-Tree (SP) es una variación de MT. La diferencia principal consiste en permitir instancias compartidas entre los nodos hijos. La Figura 1 presenta cómo se dividen los datos con el mismo procedimiento que MT, permitiendo un conjunto de instancias duplicadas en los nodos hijos. El área de solapamiento es dependiente del parámetro τ. Cuando τ es 0, no se comparten instancias, sería un MT. Si τ es demasiado alto, el solapamiento es alto y la profundidad del árbol tiende a infinito. SP reduce los tiempos de búsqueda respecto a MT sacrificando la *vuelta atrás* en el árbol para comprobar que C es el más cercano. Debido al solapamiento entre hijos, la instancia que encuentra, aunque aproximada, es muy representativa del problema.

HS aparece con el objetivo de obtener un balance entre precisión y tiempo de ejecución. Para ello, fusiona los modelos MT y SP. Para construir la estructura hibrida de HS, se construye un SP, y si el número de instancias en el área de solapamiento es menor que el Umbral de Equilibrio (UE), se mantiene como SP. Si las instancias repetidas superan el UE, se reconstruye como MT. Para calcular kNN sobre esta estructura, se realizará *vuelta atrás* para asegurar el más cercano si la rama es MT, mientras que los nodos SP no harán esta comprobación. Destacar la implementación

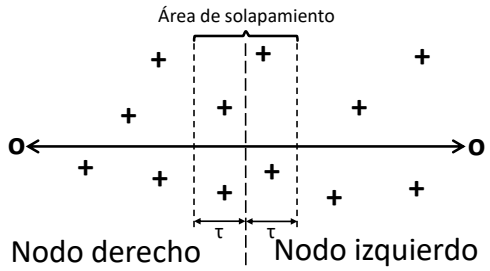

Figura 1: Construcción de Metric-Tree y Spill-Tree

disponible en el repositorio de software libre spark-package[1], que es el punto de partida en la etapa de desarrollo del presente trabajo.

II-C. Paradigma de programación MapReduce: Spark

Para el desarrollo del algoritmo propuesto en este trabajo se utilizará el paradigma de programación MapReduce [10]. MapReduce tiene como objetivo procesar grandes conjuntos de datos a través de la distribución del almacenamiento de datos y ejecución a través de un conjunto de máquinas.

La implementación de MapReduce seleccionada es Apache Spark [11]. Spark paraleliza el cálculo de forma transparente a través de una estructura de datos distribuidos llamada *Resilient Distributed Datasets* (RDD). Los RDDs permiten persistir y reutilizar estructuras de datos almacenadas en memoria principal. Adicionalmente, Spark fue desarrollado para cooperar con el sistema de archivos distribuidos de Hadoop[2] (Hadoop Distributed File System - HDFS). Con esta configuración, se puede aprovechar la división de instancias, la tolerancia a fallos y la comunicación de trabajos que proporciona Spark.

Spark incluye una librería de aprendizaje automático llamada MLlib[3]. Dispone de multitud de algoritmos de aprendizaje automático y técnicas estadísticas de diferentes áreas como clasificación, regresión o preprocesamiento de datos.

III. Fuzzy kNN acelerado para Big data

En esta sección se presenta una propuesta aproximada y distribuida del algoritmo Fuzzy kNN basado en el método HS para tratar problemas big data implementado en Spark. La presente propuesta tiene las mismas dos etapas que el modelo Fuzzy kNN: grado de pertenencia a clase (Sección III-A) y clasificación (Sección III-B).

III-A. Etapa de cálculo de grado de pertenencia a clase

Esta sección explica el flujo de trabajo de dos enfoques aproximados para calcular el grado de pertenencia a clase: Local basado en kNN (Sección III-A1) y global basado en HS (Sección III-A2) . El resultado de ambos enfoques es modificar la etiqueta de clase del Conjunto de Entrenamiento (CE) por un vector de pertenencia a clase, formando el Conjunto de Entrenamiento Difuso (CED)

[1]Hybrid Spill-Tree. https://spark-packages.org/package/saurfang/spark-knn
[2]Apache Hadoop. Web: http://hadoop.apache.org/
[3]Machine Learning Library for Spark. Web: http://spark.apache.org/mllib/

Algorithm 1 Etapa de cálculo de grado de pertenencia - Local

Require: CE, k, $\#Maps$
1: $CEP \leftarrow$ repartition(CE, #Maps)
2: $CEPD \leftarrow$ mapPartition(calcularPertenencia(CEP, k))
3: $CED \leftarrow$ unir($CEPD$)
4: **return** CED
5: **COMIENZA calcularPertenencia**
6: **for** y: CEP_i **do**
7: $Vecinos_y \leftarrow$ calcularkNNLocal ($modelo$, k, y)
8: $pertenencia_y \leftarrow$ calcularPertenencia ($Vecinos_y$)
9: $CEPD \leftarrow$ unir(y, $pertenencia_y$)
10: **end for**
11: **return** $CEPD$
12: **FIN calcularPertenencia**

Algorithm 2 Etapa de cálculo de grado de pertenencia - Global

Require: CE, k
1: $muestreas \leftarrow$ muestrear(CE,0, 2 %)
2: $arbolSuperior \leftarrow$ construirMT($muestreas$)
3: $\tau \leftarrow$ estimarTau($TopTree$)
4: $arbol \leftarrow$ repartir(CE, $arbolSuperior$, τ, $UE = 70\,\%$)
5: $modelo \leftarrow$ (difundir($arbolSuperior$),arbol)
6: **for** y: CE **do**
7: $Vecinos_y \leftarrow$ calcularkNN ($modelo$, k, y)
8: $pertenencia_y \leftarrow$ calcularPertenencia ($Vecinos_y$)
9: $resultado_y \leftarrow$ unir(y, $pertenencia_y$)
10: **end for**
11: **return** $resultado$

III-A1. Enfoque local: El enfoque local conjuntamente con la etapa de clasificación recibe el nombre *Local Hybrid Spill-Tree Fuzzy kNN* (L-HSFkNN). El Algoritmo 1 muestra los pasos y operaciones en Spark para el cálculo del grado de pertenencia. Comienza leyendo el CE de HDFS y es dividido en $\#Maps$ partes. Posteriormente, utiliza una operación mapPartition de Spark para calcular de forma distribuida el grado de pertenencia a clase para cada partición CEP_i. Para cada instancia y de cada partición CEP_i, calcula kNN y finalmente obtiene el grado de pertenencia aplicando la Ecuación 1. Una vez obtenida la pertenencia para cada partición, se agrupan los resultados y forman el CED, que será la entrada de la etapa de clasificación.

III-A2. Enfoque global basado en HS: El enfoque global sumado a la etapa de clasificación recibe el nombre *Global Approximate Hybrid Spill-Tree Fuzzy kNN* (GA-HSFkNN). El Algoritmo 1 muestra los pasos y operaciones en Spark para el cálculo del grado de pertenencia.

El algoritmo 2 muestra los pasos de la etapa de cálculo de pertenencia. Las líneas 1-5 corresponden a la etapa de ajuste del modelo, y las restantes al cálculo del grado de pertenencia.

La etapa de ajuste del modelo comienza leyendo el CE de HDFS. En primer lugar, toma un submuestreo aleatorio para construir un MT como se describe en la Sección II-B (Los autores recomiendan un 0,2 %). Este MT recibe el nombre de árbol superior (AS) y es usado para estimar el valor del parámetro τ y particionar todo el CE. La estimación de τ es la distancia media entre las instancias. Para acelerar este cálculo, se realiza con las instancias del AS.

El siguiente paso es repartir el CE. Para ello, a partir

Algorithm 3 Calcular kNN

Require: $modelo$, k, x
1: $Indices \leftarrow$ x.flatMap (buscaIndices($modelo.arbol$))
2: $Vecinos \leftarrow$ consulta($modelo.arbol$, $Indices$, k)
3: **return** $Vecinos$
4:
5: **COMIENZA buscaIndices**
6: $distIzq \leftarrow$ nodoIzq.dist(x)
7: $distDch \leftarrow$ nodoDch.dist(x)
8: **if** $nodo! = HOJA$ **then**
9: **if** $distIzq < distDch$ **then**
10: buscaIndices($nodoIzq$, ID)
11: **else**
12: buscaIndices($nodoDch$, $ID + hijoIzq$)
13: **end if**
14: **else**
15: **return** $Indices$
16: **end if**
17: **FIN buscaIndices**

Algorithm 4 Etapa de clasificación

Require: CED, CP, k
1: $muestreas \leftarrow$ muestrear(CED,0, 2 %)
2: $arbolSuperior \leftarrow$ construirMT($muestreas$)
3: $\tau \leftarrow$ estimarTau($TopTree$)
4: $arbol \leftarrow$ repartir(CED, $arbolSuperior$, τ, $UE = 70\,\%$)
5: $modelo \leftarrow$ (difundir($arbolSuperior$),arbol)
6: **for** y: CE **do**
7: $VecinosP_y \leftarrow$ calcularFuzzykNN ($modelo$, k, y)
8: $prediccion_y \leftarrow$ calcularPredicción ($Vecinos_y$)
9: **end for**
10: **return** $prediccion_y$

del AS se distribuyen las instancias en el espacio. El valor de τ define el área de solapamiento. Comienza construyendo un SP, y comprueba si el número de instancias en el área de solapamiento es menor que el 70 %. En otro caso, se reconstruye un MT. Al realizar la búsqueda, aquellas ramas SP realizan una búsqueda más rápida al no tener que volver atrás en el árbol. Sin embargo, las construidas como MT realizan *vuelta atrás* para asegurar el más cercano. La etapa de construcción del modelo termina distribuyendo el AS y el árbol asociado al CE.

La etapa de cálculo de grado de pertenencia se muestra en las líneas 6-10. Para cada instancia de CE, se calcula kNN siguiendo el modelo generado. El Algoritmo 3 describe como se realiza la búsqueda con operaciones nativas de Spark. Mediante una operación flatMap, se computan y obtienen los índices de las instancias más cercanas del CE. Para ello, se calcula la distancia hasta el nodo derecho e izquierdo, y continúa por el nodo con una distancia menor. Cuando alcanza un nodo hoja, devuelve el índice de la instancia seleccionada.

Dados los vecinos, se calcula el grado de pertenencia siguiendo la Ecuación 1. (Línea 8). El resultado de esta fase es el CED, pasando a ser la entrada de la etapa de clasificación.

III-B. Clasificación con Hybrid Spill-Tree

Esta sección describe la etapa de clasificación, la cual recibe como entrada el CED calculado en la etapa previa. La etapa de clasificación está basada en HS y sigue la misma estructura que la aproximación global de la primera etapa (Sección III-A2). Tiene dos fases diferenciadas: creación del modelo y clasificación. La primera construye el árbol y divide las instancias entre los nodos de cómputo. La segunda busca las k instancias más cercanas de CED calculado en la primera etapa, y devuelve como salida la clase predicha de acuerdo al vector de grado de pertenencia.

El Algoritmo 4 muestra los pasos de la etapa de clasificación. Se especificarán las diferencias con respecto a la sección III-A2, ya que el esquema principal es el mismo y sólo se ven afectados pequeños detalles de la estructura de datos.

La primera diferencia se encuentra en los conjuntos de datos de entrada. En este caso, se utilizarán CED y CP. La fase de ajuste del modelo no se ve afectada pues no se modifican las características, manteniendo las distancias de las instancias. Así, se construye el modelo con la misma metodología.

El cálculo de Fuzzy kNN se hace de la misma manera, con la diferencia de que en lugar de devolver la etiqueta de clase para cada vecino, se devuelve el vector de grado de pertenencia a la clase (Línea 7). La línea 8 calcula la clase predicha aplicando la Ecuación 2. El resultado final es la clase prevista para cada instancia de CP.

IV. Estudio experimental

En esta sección se presentan las cuestiones planteadas en el estudio experimental. La Sección IV-A presenta los algoritmos de comparación utilizados para la experimentación. La Sección IV-B determina el marco de los experimentos. Finalmente, la Sección IV-C expone y analiza los resultados obtenidos.

IV-A. Algoritmos de comparación

Esta sección expone los algoritmos de comparación utilizados en los experimentos y sus acrónimos. Se compara con otras propuestas de Fuzzy kNN y sus análogos no *Fuzzy* siguiendo dos enfoques: local y global. El enfoque local divide las instancias y los distribuye entre los nodos de cómputo, ejecutando de forma independiente en cada partición, obteniendo resultados aproximados. El enfoque global si considera todas las instancias, aunque realice el cómputo de forma distribuida y a su vez presenta dos posibilidades: exacto y aproximado. El exacto invierte cálculo en asegurar que el resultado es igual que la variante secuencial. El aproximado, no asegura el mismo resultado acelerando el tiempo de ejecución.

Algoritmos utilizados en la experimentación para realizar la comparativa:

- *Global Exact Fuzzy kNN (GE-FkNN)* [8]: modelo exacto del algoritmo Fuzzy kNN para abordar problemas big data, obteniendo los mismos resultados que el Fuzzy kNN original. Sus dos etapas son globales y exactas.
- *Local Fuzzy kNN (L-FkNN)*: propuesta desarrollada del algoritmo Fuzzy kNN. La primera etapa es la descrita en la Sección III-A1 y su segunda etapa sería global y exacta, idéntica a la de GE-FkNN.

- *k Nearest Neighbor - Iterative Spark (kNN-IS)* [5]: propuesta exacta de kNN para abordar problemas big data, obteniendo los mismos resultados que el kNN original.
- *Hybrid Spill-Tree kNN (HS-kNN)* [9]: propuesta aproximada de kNN para big data. Aunque aproximada, considera todas las instancias en la búsqueda.

IV-B. Marco experimental

Para el estudio experimental, se han seleccionado cuatro conjuntos de datos de un elevado número de instancias. El conjunto de datos ECBDL14 se extrae de la competición [12]. A pesar de que tiene una relación de desbalanceo superior a 45, hemos seleccionado este conjunto de datos por su número relativamente alto de características. Sin embargo, en este documento no abordamos el problema de la clasificación desequilibrada, por lo que se ha submuestreado dejándolo en una relación de desbalanceo de dos. Los otros tres conjuntos han sido extraídos del repositorio UCI [13]. La Tabla I presenta el número de instancias, características y clases ($\#\omega$). Se seguirá el esquema de validación cruzada en 5 particiones.

Tabla I: Descripción de los conjuntos de datos

Conjunto de datos	#Instancias	# Características	$\#\omega$
Poker	1.025.010	10	10
ECBDL14	2.063.187	631	2
Susy	5.000.000	18	2
Higgs	11.000.000	28	2

Se tendrán en cuenta las siguientes medidas para evaluar el rendimiento de la técnica propuesta:

- *Precisión:* Contabiliza el número de clasificaciones correctas en relación con el número total de instancias.
- *Tiempo de ejecución:* Tiempo consumido considerando lecturas y comunicaciones por red de Spark.

El parámetro más conocido para el algoritmo original Fuzzy kNN es el número de vecinos (k). k podría ser diferente en cada etapa, pero mantendremos ambos iguales por simplicidad. En nuestros experimentos, el parámetro k está ajustado a 3, 5 y 7. Se necesita un parámetro adicional debido a su comportamiento distribuido. Este es el número de particiones del conjunto de entrenamiento. Para exprimir la componente distribuida, se utilizará el máximo posible para la configuración del cluster, 256 operaciones maps/particiones.

Los algoritmos basados en HS tienen dos parámetros adicionales: el número de instancias para construir el árbol principal AP y el umbral de equilibrio UE. Los valores recomendados por los autores son AP igual al 0,2 % y UE igual al 70 %.

Todos los experimentos se han realizado en un cluster compuesto por 14 nodos de cálculo gestionados por un nodo maestro. Todos los nodos tienen la misma configuración. 2 procesadores Intel Xeon CPU E5-2620, 6 núcleos (12 hilos) por procesador, 2 GHz y 64 GB de RAM. Red Infiniband 40Gb/s. Con la configuración actual, se disponen de 256 tareas map como máximo. La versión de Spark es 2.2.1.

IV-C. Resultados obtenidos y análisis

Esta sección presenta una comparativa de los modelos propuestos y los detallados en la Sección IV-A, analizando el tiempo de ejecución y la precisión de cada uno de ellos. Para realizar un análisis detallado de los algoritmos, se utilizarán los 4 conjuntos de datos expuestos en la Tabla I. La Sección IV-C1 presenta el tiempo de ejecución y precisión. La Sección IV-C2 compara los algoritmos propuestos con otros algoritmos de la literatura.

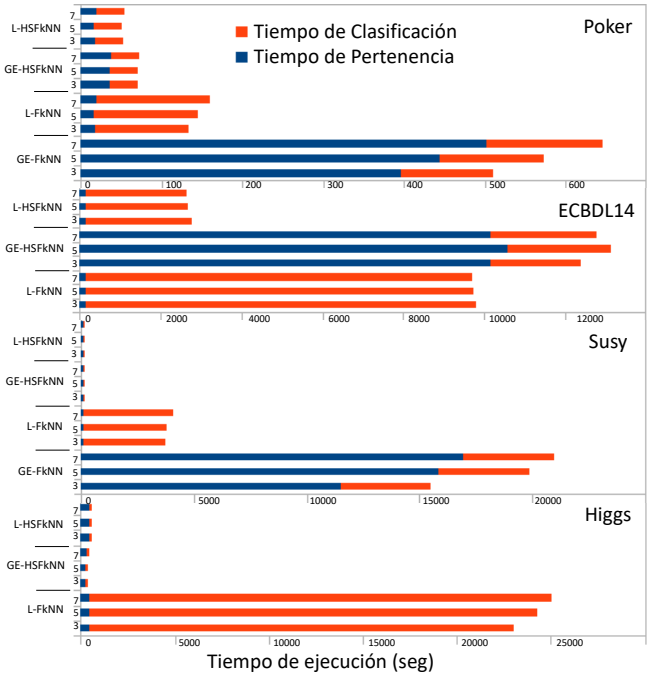

Figura 2: Tiempo de ejecución para cada conjunto y algoritmo

IV-C1. Estudio de tiempo y precisión: Esta sección presenta y estudia el tiempo de ejecución y precisión obtenida para los 4 conjuntos de datos y los diferentes algoritmos.

La Tabla II muestra una comparación de precisión entre los algoritmos en relación al número de vecinos (k) y para todos los conjuntos de datos.

Tabla II: Comparativa entre algoritmos - Precisión

Algoritmo	k	Poker	ECBDL14	Susy	Higgs
GE-FkNN	3	0,5257	-	0,7301	-
	5	0,5316	-	0,7306	-
	7	0,5338	-	0,7268	-
L-FkNN	3	0,5218	0,7636	0,7178	0,5936
	5	0,5243	0,7492	0,7190	0,5953
	7	0,5243	0,7423	0,7179	0,5959
GA-HSFkNN	3	0,5233	**0,8051**	0,7302	0,5968
	5	0,5371	0,8050	0,7457	0,6081
	7	0,5459	0,8014	**0,7510**	0,6162
L-HSFkNN	3	0,5324	0,8035	0,7320	0,6038
	5	0,5432	0,8027	0,7438	0,6139
	3	**0,5487**	0,7969	0,7471	**0,6198**

La Figura 2 muestra el tiempo de la etapa de cálculo de pertenencia y el tiempo de la etapa de clasificación en segundos, para cada conjunto de datos, algoritmo y los valores de k igual a 3, 5 y 7.

De acuerdo con la tabla y figura presentadas, se observa:

Tabla III: Comparativa contra propuestas kNN

Conjunto de datos	Algoritmo	k	Tiempo Total	Precisión
Poker	kNN-IS	3	113,2370	0,4758
		5	123,8311	0,4952
		7	136,5830	0,4937
	HS-kNN	3	32,9661	0,5201
		5	33,4674	0,5305
		7	35,5575	0,5369
	GA-HSFkNN	7	72,1978	0,5459
	L-HSFkNN	7	54,3435	**0,5487**
ECBDL14	kNN-IS	3	27121,0583	0,7833
		5	28673,7015	0,7797
		7	28918,2992	0,7683
	HS-kNN	3	3151,6242	0,8020
		5	2608,1722	0,8017
		7	2691,5595	0,7986
	GA-HSFkNN	3	12413,8265	**0,8051**
	L-HSFkNN	3	2787,1230	0,8035
Susy	kNN-IS	3	2615,0150	0,6675
		5	2273,6377	0,6784
		7	2372,4100	0,6861
	HS-kNN	3	128,1860	0,7223
		5	131,0210	0,7360
		7	133,2428	0,7431
	GA-HSFkNN	7	133,2568	**0,7510**
	L-HSFkNN	7	146,2076	0,7471
Higgs	kNN-IS	3	10262,9178	0,5454
		5	13446,0519	0,5458
		7	14285,6442	0,5559
	HS-kNN	3	641,4434	0,5885
		5	651,1573	0,5936
		7	664,9204	0,5981
	GA-HSFkNN	7	429,3507	0,6162
	L-HSFkNN	7	608,2571	**0,6198**

- El valor de k no afecta significativamente a los tiempos de ejecución en ninguna de las etapas.
- El algoritmo GE-FkNN encuentra su límite de escalabilidad en su primera etapa, no llegando a ejecutar para los datasets ECBDL14 y Higgs.
- Centrándonos en los tiempos de ejecución, el modelo L-FkNN escala mejor en la primera etapa y se ve menos afectado por el número de características. El algoritmo GA-HSFkNN consigue mejores tiempos en la etapa de clasificación. Así, los mejores tiempos de ejecución son obtenidos por el algoritmo L-HSFkNN, que toma la primera etapa del algoritmo L-FkNN y la etapa de clasificación de GA-HSFkNN.
- En precisión los ganadores son GA-HSFkNN y L-HSFkNN con diferencias poco significativas. Teniendo en cuenta que los tiempos de ejecución totales son menores para L-HSFkNN, es nuestra propuesta más escalable y de mayor calidad.

IV-C2. Comparativa con versiones kNN big data: En esta sección se comparan los modelos basados en conjuntos difuso con las versiones clásicas de kNN para big data. La Tabla III muestra una comparativa entre el mejor resultado obtenido por los dos algoritmos propuestos y las dos alternativas no *Fuzzy*, explorando los mismos valores de k. Se muestra para cada conjunto de datos y algoritmo, el tiempo total de ejecución y el valor de k utilizado asociado a la precisión obtenida.

De acuerdo con la tabla presentada, se puede observar que los mejores resultados los obtienen las versiones *Fuzzy*. Además, kNN-IS obtiene resultados significativamente peores que los demás algoritmos. Aunque HS-kNN mejora respecto al algoritmo kNN-IS, siempre queda por debajo de los modelos propuestos, sin verse incrementado en exceso el tiempo de ejecución debido a la optimización realizada en la etapa de clasificación del algoritmo GA-HSFkNN.

V. Conclusiones y trabajo futuro

En esta contribución se ha propuesto un enfoque MapReduce para acelerar el algoritmo Fuzzy kNN en problemas Big data. Gracias al diseño realizado y al uso de tecnologías big data, se consigue ejecutar con conjuntos de datos muy grandes, que de otra forma sería inviable. Los experimentos realizados comparan la propuesta con otros enfoques de comportamiento exacto y aproximado para conseguir un equilibrio entre eficiencia y precisión. Asimismo, se ha comparado con las versiones no *Fuzzy* para estudiar la mejora del modelo propuesto. El algoritmo L-HSFkNN demuestra una escalabilidad muy elevada, así como resultados en precisión de alta calidad. Como trabajo futuro, se plantea adaptar el modelo desarrollado a otros problemas de minería de datos como regresión, o aprendizaje semi-supervisado.

Agradecimientos

Este trabajo se ha sustentado por el proyecto de investigación TIN2017-89517-P. J. Maillo disfruta de una beca FPU del Ministerio de Educación de España.

Referencias

[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. 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.
[6] J. K. Uhlmann, Satisfying general proximity / similarity queries with metric trees, Information Processing Letters 40 (4) (1991) 175 – 179.
[7] 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.
[8] 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.
[9] 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).
[10] J. Dean, S. Ghemawat, Map reduce: A flexible data processing tool, Communications of the ACM 53 (1) (2010) 72–77.
[11] 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.
[12] ECBDL14 dataset: Protein structure prediction and contact map for the ECBDL2014 big data competition (2014).
URL http://cruncher.ncl.ac.uk/bdcomp/
[13] M. Lichman, UCI machine learning repository (2013).
URL http://archive.ics.uci.edu/ml