CIMAT

Centro de Investigación en Matemáticas, A.C.

Reducción de instancias para grandes datos

T E S I S

Que para obtener el grado de

Maestro en Cómputo Estadístico

Presenta

Victor Manuel Martinez Santiago

Director de Tesis:

Dr. Alejandro Rosales Pérez

Co-director de Tesis:

Dr. Edgar Jiménez Peña

Autorización de la versión final

Monterrey, Nuevo León, Junio de 2022

Dedicatoria ...

Agradecimientos

Al Centro de Investigación en Matemáticas por la oportunidad de ser uno de sus alumnos, al Laboratorio de Supercómputo del Bajío (proyecto CONACyT 300832) y al Laboratorio de Computo Unidad Monterrey por las facilidades otorgadas para la elaboración de la tesis.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT) por la beca que me otorgó para la realización de este programa de maestría con número de becario y número de CVU: 1076711.

Resumen

En el mundo existe una gran variedad de problemas en los cuales se quieren aprender modelos para llevar a cabo la tarea de segmentación y cada vez es más común tener grandes volúmenes de información, por lo que el almacenamiento y la velocidad con la cual analizar la información disponible se vuelve relevante. La selección de instancias es una opción para reducir tiempos de entrenamiento y predicción además de reducir el espacio de almacenamiento de la información.

Existe una investigación relativamente extensa para realizar la selección en conjuntos de datos pequeños, pero no es el caso cuando tenemos miles o millones de observaciones.

Las soluciones de ultima generación enfocadas a realizar la selección con grandes bases de datos plantean realizar la selección pensando en una distancia en un espacio euclideano. Esta Tesis, propone el uso de distancias en el espacio kernel como alternativa con el objetivo de controlar el numero de instancias seleccionadas así como para tener un mejor control de la métrica de interés. Para ello se retoman propuestas de ultima generación y se propone modificaciones a las mismas.

El estudio experimental considera escenarios donde se tienen conjuntos de datos balanceados y no balanceados, se realiza una extensa evaluación considerando diferentes clasificadores y un reporte con diferentes métricas.

Los resultados experimentales dan evidencia que el uso de las distancia en el espacio kernel puede ser usada como un parámetro para controlar el numero de instancias retenidas y para mejorar métricas de interés.

Palabras clave: Selección de instancias, Tarea de Clasificación, Segmentación, *MapReduce* , *Spark*, Distancias, Vecinos más cercanos, Clasificadores.

Índice general

Índice de figuras

Capítulo 1

Introducción

Constantemente nuevas propuestas enfocadas a la toma de decisiones a partir de datos estructurados son exploradas y puestas en practica, por lo que hablar de aprendizaje supervisado invita a mencionar tareas de regresión y clasificación, en ambos escenarios se tienen datos etiquetados. Si pensamos que en ambos ejercicios el objetivo es aprender una función que me permite obtener un valor de salida a partir de ciertos valores de entrada, la diferencia entre ambas tareas radica en el tipo de salida que puede ser una etiqueta de clase discreta o bien un número real continuo.

Idea 1 A la par de estas nuevas ideas, la disponibilidad de grandes volúmenes de información es más común cada día y uno de los mayores retos es administrar dicha información de manera eficiente. Una posible solución ante tal situación es la reducción del conjunto de datos, la selección de instancias es un conjunto de técnicas efectivas que permiten reducir el volumen de la información y es tema de interés en esta Tesis, particularmente para la tarea de clasificación.

Idea 2 A la par de estas nuevas propuestas de aprendizaje maquina, la disponibilidad de grandes volúmenes de información es más común cada día, por lo que cuando hay un grado muy alto de información redundante irrelevante y datos ruidosos el proceso de descubrimiento de conocimiento es muy difícil de llevar a cabo Zhang, Zhang y Yang (2003). Una posible solución ante tal situación es utilizar técnicas de preprocesamiento como lo son los métodos de reducción de datos, la selección de instancias es un conjunto de técnicas efectivas que permiten sobre llevar las proble-

máticas comentadas y es tema de interés en esta Tesis específicamente para la tarea de de clasificación.

1.1. Problema

Si consideramos datos tabulados, entenderemos por instancia a un elemento de nuestro conjunto de datos que tiene p características, sea x una instancia tal que $x \in \mathbb{R}^p$. Por lo que se refiere a selección de instancias, (Olvera-López, Carrasco-Ochoa, Martínez-Trinidad, y Kittler, 2010) señala que: "El objetivo de un método de selección de instancias es obtener un subconjunto $S \in T$ tal que S no contenga instancias superfluas y $Acc(S) \cong Acc(T)$ donde $Acc(X)$ es la exactitud de clasificación obtenida usando X como conjunto de entrenamiento".

Una problemática vigente en la selección de instancias, es la dificultad de no perder rendimiento en la métrica de interés, esto una vez que el conjunto S es seleccionado, además, es común que en este tipo de algoritmos se tenga una alta demanda computacional, puesto que la mayoría de ellos son de orden $O(n^2)$ y $O(n^3)$, lo cual es un obstáculo y vuelve inviable el aplicar los algoritmos a grandes volúmenes de datos.

1.2. Justificación

En la practica, cada vez es más común encontrarse con conjuntos de datos grandes, por lo que entrenar un modelo con millones de instancias puede ser un problema difícil en marcos de trabajo tradicionales, no todos los usuarios tienen acceso directo a la infraestructura computacional adecuada para *big data* o la capacidad de trabajar con marcos dedicados como Apache Spark y la renta de un servicio externo en la nube podría resultar en un costo excesivo. Además, cabe señalar que existen algoritmos de aprendizaje automático de última generación que aún no han sido desarrollados para ser escalables. En ese sentido, la importancia que la reducción del conjunto de datos conlleva en la etapa de preprocesamiento es reducir los requisitos de espacio

del sistema así como el tiempo de procesamiento de las tareas de aprendizaje. En muchas ocasiones los datos suelen contener instancias ruidosas y valores atípicos, por lo que es común que ciertos algoritmos de selección de instancias tengan la intención de solucionar en parte dicha problemática.

A pesar de que la selección de instancias para la tarea de clasificación ha sido ampliamente investigada en las últimas décadas, no es común encontrar literatura disponible donde se realice una comparación del desempeño de las metodologías en grandes volúmenes de datos, particularmente en el *framework* de trabajo *Spark*.

De la misma manera en que el algoritmo *kernel* KNN pretende mejorar el desempeño del algoritmo convencional KNN Yu, Ji, y Zhang (2002), la idea es utilizar los espacios de Hilbert para realizar la selección de instancias, esta hipótesis de obtener una mejora pudiera ser verdadera pues la mayoría de las metodologías están intrínsecamente relacionadas con el modelo de clasificación KNN.

El uso de marcos de trabajo como *Spark* y el paradigma de programación que implica su utilización resulta no ser la primera opción de usuarios interesados en la ciencia de datos, en consecuencia a pesar de tener una gran comunidad la falta de algoritmos y metodologías es evidente, por lo que queremos contribuir a la existencia y disponibilidad de dichos algoritmos, en este caso para llevar a cabo la selección de instancias.

1.3. Objetivos

Desarrollar un método que permita seleccionar las instancias más relevantes de un conjunto de datos, considerando una función kernel, bajo el paradigma de *Spark* con la finalidad de que sea escalable a grandes volúmenes de datos.

Para lograr el objetivo general, proponemos los siguientes objetivos especificos

1. Realizar la implementación de algoritmos de selección de instancias ya existentes en un entorno que permita que sean escalables y por ende aplicable a grandes volúmenes de datos.

2. Estructurar una propuesta que permita mejorar los algoritmos ya existentes de selección de instancias utilizando espacios de Hilbert.

3. Evaluar y comparar el rendimiento de diferentes algoritmos de selección de instancias en *Big Data*.

1.4. Organización de la tesis

El documento se estructura de la siguiente forma:

- El Capitulo 2 presenta los fundamentos teóricos relacionados con el problema de selección de instancias, el marco de trabajo Spark, y los algoritmos de ultima generación utilizados para tratar la selección de instancias con grandes volúmenes de información.

- En el Capitulo 3 se presentan los elementos necesarios para el calculo de la distancia en el espacio kernel y se realiza la descripción de las variaciones realizadas en los algoritmos de ultima generación para adoptar nuestra propuesta.

- En el Capitulo 4 se describe el diseño experimental y se exponen los resultados obtenidos.

- El Capitulo 5 contiene las conclusiones así como posibles extensiones al trabajo presentado.

Capítulo 2

Marco teórico

Como se mencionó en el capítulo anterior, en ocasiones tener una gran cantidad de datos no es lo más ideal por lo que ante tal planteamiento del problema han surgido diversas investigaciones para tratar de dar una solución.

En este capítulo se comentan ciertos trabajos relevantes que se han reportado y que están relacionados con nuestro tema de investigación. Primero se aborda los antecedentes de la selección de instancias, una posible taxonomía de las mismas y se nombran y describen las metodologías mas relevantes, posteriormente se desarrolla los antecedentes de *Big Data*, su definición así como su importancia en la actualidad, se toma uno de los marcos de trabajo más importantes hoy en día y se desarrolla de manera breve la historia del mismo. Por último, comentaremos los trabajos relacionados a selección de instancias en *Big Data* con el propósito de conocer las bases existentes de las cuales partir para desarrollar nuestro trabajo.

2.1. Selección de instancias

Como ya hemos comentado en la Sección (1.1), la selección de instancias tiene como objetivo obtener un subconjunto de observaciones, conocidos como representantes, que puede describir de manera eficiente y confiable todo el conjunto de datos y es un tema que cobró interés a finales de los años 60 y principios de los 70, entre los primeros trabajos podemos mencionar: *Condensed Nearest Neighbor*(CNN)

(Hart, 1968), *Reduced Nearest Neighbor*(ENN) (D. L. Wilson, 1972) y *Selective Nearest Neighbor*(SNN) (Ritter, Woodruff, Lowry, y Isenhour, 1975). La mayoría de los algoritmos existentes se enfocan en reducir los requisitos de espacio y tiempo para la regla *Nearest Neighbor*(NN), estas metodologías son consideradas como una tarea dentro del preprocesamiento de datos, por lo que las aplicaciones no son únicamente para la regla NN, son algoritmos de una alta complejidad complejidad computacional, al menos log-lineal (González, 2018).

Publicaciones como las realizadas por (Olvera-López *et al.*, 2010) y (García *et al.*, 2012) tienen como objetivo recopilar y resumir diversas técnicas de selección de instancias. Para el año 2012 García reporta que existen más de 50 métodos de selección de instancias, donde *Condensed Nearest Neighbor*(CNN), *Edited Nearest Neighbor* (ENN), *Iterative Case Filtering* (ICF) y *Decremental Reduction Optimization* (DROP) son denominados como los algoritmos más influyentes (García, Luengo, y Herrera, 2015).

2.1.1. Taxonomía de los métodos de selección de instancias

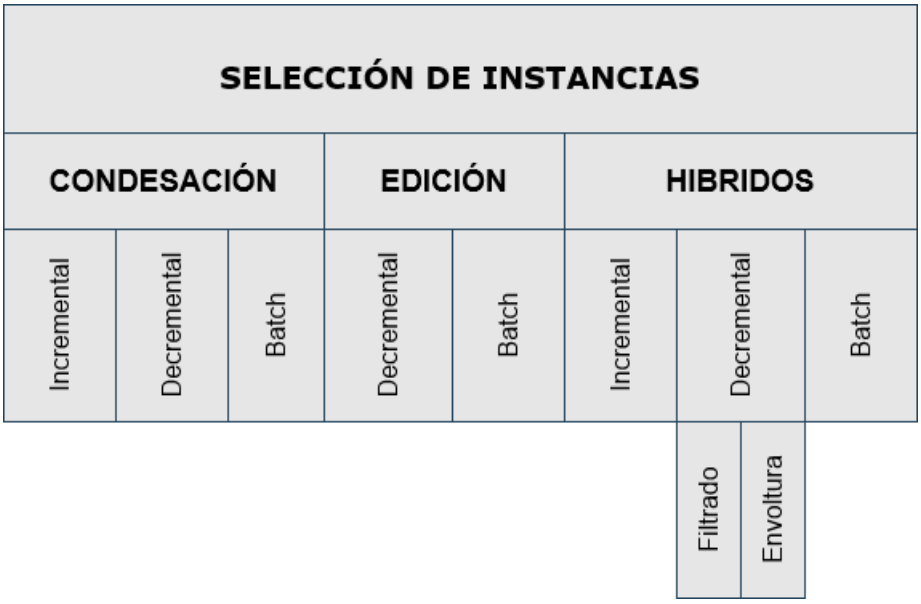

Figura 2.1: Taxonomia de Seleccion de Instancias, Figura basada en (García *et al.*, 2012)

La Figura (2.1) esta basada en el diagrama presentado por (García *et al.*, 2012) e ilustra lo que seria la una posible taxonomía de los métodos de selección de instancias, en los siguientes párrafos describimos de manera breve en que consiste cada uno de ellos así como ejemplos de metodologías de los mismos, notese que por lo general cualquier metodología puede ser clasificada de más de una manera.

- Métodos de envoltura y filtrado

De acuerdo con (Olvera-López *et al.*, 2010) las técnicas de selección de instancias se pueden agrupar como técnicas de envoltura y filtrado, en las técnicas de envoltura tenemos una estrecha relación entre alguna métrica y el clasificador, por otra parte, las decisiones del segundo grupo no están asociadas a un clasificador y en principio pudieran ser usadas con cualquier modelo de aprendizaje supervisado. Como ya hemos mencionado, la mayoría de los métodos propuestos están basados en KNN, por lo que las metodologías CNN, SNN y *Generalized Condensed Nearest Neighbor rule*(GCNN) serían métodos de envoltura para el clasificador KNN y metodologías como *Pattern by Ordered Projections*(POP) o *Clustering instance selection method (CLU)* son técnicas de filtrado.

- Métodos de condensación y edición

Los algoritmos de selección podrían prestar un poco más de atención a puntos fronterizos o a puntos centrales, los métodos de condensación son aquellos que buscan retener los puntos en las fronteras y eliminar los puntos internos, mientras que los métodos de edición buscan eliminar puntos fronterizos ruidosos y no buscan la eliminación de puntos internos, también podemos mencionar una combinación entre los dos métodos obteniendo un método híbrido que permite la eliminación de puntos fronterizos y puntos internos (García *et al.*, 2012). Ejemplos de métodos que se centran en puntos fronterizos son CNN, *Fast Condensed Nearest Neighbor*(FCNN) y GCNN, ejemplos de métodos de edición son ENN y Multiedit, entre los métodos híbridos encontramos a *Decremental Reduction Optimization Procedure*(DROP) o *Instance Based* (IB).

- Métodos incrementales y decrementales

Los métodos incrementales se caracterizan por ir añadiendo instancias al conjunto S, ejemplos de métodos incrementales incluyen CNN, FCNN, *Instance Based 3*(IB3). Los métodos decrementales inicialmente consideran todas las instancias T y la idea es ir eliminando instancias hasta obtener el conjunto S, ENN, Multiedit son ejemplos de métodos decrementales (García *et al.*, 2012).

- Métodos Heurísticos

Los algoritmos evolutivos, como los algoritmos genéticos, son métodos adaptativos basados en la evolución natural, por lo que a la población de individuos que intenta modelar el problema, es común someterlos a operadores probabilísticos como mutación, selección y recombinación para evolucionar hacia valores de aptitud cada vez mejores (Bäck, Fogel, y Michalewicz, 1997). *Generational Genetic Algorithm*(GGA), *Steady-State Genetic Algorithm*(SGA) y *Adaptive Search Algorithm*(CHC) son ejemplos de estos algoritmos (Cano, Herrera, y Lozano, 2004).

2.1.2. CNN (*Condensed Nearest Neighbor*)

Subconjunto consistente

Es un subconjunto de observaciones que cuando se usa como conjunto de referencia para realizar la clasificación bajo la regla NN clasifica correctamente todos los puntos restantes en el conjunto de muestra (Hart, 1968). En ese sentido ,todo conjunto tiene un subconjunto consistente, ya que todo conjunto es trivialmente un subconjunto consistente de sí mismo.

Subconjunto consistente mínimo

Es un subconjunto consistente con un número mínimo de elementos, así todo conjunto finito tiene un subconjunto consistente mínimo, aunque el tamaño mínimo, en general, no se consigue de manera única.

El algoritmo CNN descrito por Hart (1968) no encontrará un subconjunto consistente mínimo, aun así, es considerado como el primer algoritmo de referencia que introduce dicho concepto, el algoritmo procede de la siguiente manera.

1. Sea S un subconjunto inicialmente vació, una primera muestra se agrega a S
2. En cada iteración del algoritmo se verificara que los puntos restantes estén bien clasificados de lo contrario el i-ésimo punto mal clasificado al usar S es seleccionado y agregado a S.
3. El algoritmo finaliza cuando no hay puntos mal clasificados en T

Algoritmo 1 CNN (Condensed Nearest Neighbour)

Entrada: Conjunto de entrenamiento $T = \{(x_1, y_1), ..., (x_n, y_n)\}$
Salida: S

```
S = random(x_i)
Para cada x_i ∈ (T − S) hacer:
    Si x_i no esta bien clasificado usando S entonces
        S := S ∪ x_i
    Fin Si
Fin Para cada
```

Debido a la aleatoriedad es poco probable que el algoritmo seleccione puntos cercanos a la frontera de decisión además el método podría requerir muchas iteraciones para converger. Con el propósito de acelerar el criterio de convergencia existen variantes del algoritmo como lo es la regla MCNN donde en cada iteración del algoritmo a lo más c puntos correspondientes a las c diferentes clases del problema son agregados en cada iteración.

2.1.3. ENN (*Edited Nearest Neighbor*)

(D. L. Wilson, 1972) propuso una versión modificada de la regla k-NN: la regla del vecino más cercano editado (ENN por sus siglas en ingles), el procedimiento propuesto es como sigue:

Se realiza una clasificación de cada elemento x_i en la muestra utilizando la regla k-NN con el conjunto de entrenamiento (se recomienda un valor impar para k). Las

muestras mal clasificadas se eliminan del conjunto de T mientras que las muestras restantes constituyen el conjunto de referencia editado S.

Algoritmo 2 ENN (Edited Nearest Neighbour)

Entrada: Conjunto de entrenamiento $T = \{(x_1, y_1), ..., (x_n, y_n)\}$
Salida: S

1: $T = \{(x_1, y_1), ..., (x_n, y_n)\}$
2: **Para cada** $x_i \in T$ **hacer:**
3: Encontrar los k-vecinos más cercanos a x_i de entre $x_1, ..., x_{i-1}, x_{i+1}, ..., x_n$
4: Encontrar la clase c por votación entre los K-vecinos más cercanos, rompiendo los empates aleatoriamente cuando ocurran
5: **Fin Para cada**
6: **Para cada** $x_i \in T$ **hacer:**
7: Eliminar del conjunto T aquellos x_i que no concuerden con la votación realizada en el paso anterior.
8: **Fin Para cada**

El procedimiento tiende a suavizar la frontera de decisión eliminando puntos fronterizos y valores atípicos, en la practica el algoritmo no logra una gran reducción del conjunto original T y por el contrario no degrada la métrica de interés.

2.1.4. FCNN (*Fast Nearest Neighbor Condensation*)

Angiulli (2005) presenta FCNN como un algoritmo para ser utilizado en grandes conjuntos de datos, la complejidad temporal en el peor de los casos es cuadrática, el algoritmo tiene propiedades interesantes como que requiere pocas iteraciones para converger, es probable que seleccione puntos cercanos al limite de decisión además de obtener un subconjunto consistente.

El algoritmo original se inicializa considerando como puntos iniciales del conjunto S a los centroides de cada clase, es un algoritmo incremental y utiliza el concepto de espacios de Voronoi (por ejemplo, el espacio de Voronoi de la instancia $p \in S$ es el número de prototipos que están más cerca de p que de otros prototipos en el conjunto S), luego, en cada iteración se analiza la posibilidad de que para cada uno de los elementos en S, una instancia $q \in T$ de etiqueta diferente se selecciona y sea añadida al conjunto S, el algoritmo finaliza cuando el conjunto T es clasificado correctamente por S.

Para entender un poco mejor el funcionamiento de FCNN y FCNN1 introduciremos la siguiente notación:

Sea p un elemento del conjunto S, luego $l(p)$ es la clase asociada al elemento p. Denotaremos por $nn(p, S)$ al vecino más cercano de p en S, por lo que es claro que si p esta en S, el vecino más cercano es el mismo. Si S es subconjunto de T y $q \in T$, denotaremos como $Vor(p, S, T)$ al conjunto de puntos q en T que tienen a p como elemento más cercano bajo alguna métrica de distancia que cualquier otro elemento de S, esto es $\{q \in T | p = nn(q, S)\}$, por otra parte $Voren(p, S, T)$ denotara al conjunto de puntos q en T que tienen una clase diferente a p en S es decir $\{q \in Vor(p, S, T)\}$. Recordemos que en cada iteración un cierto numero de observaciones son agregadas al conjunto S, por lo que el elemento representativo de $Voren(p, S, T)$ a ser agregado lo denotaremos como $rep(p, Voren(p, S, T))$.

De esta manera, el algoritmo (3), muestra la estructura general de como funciona FCNN.

Algoritmo 3 FCNN (Fast Condensed Nearest Neighbour)

Entrada: Conjunto de entrenamiento $T = \{(x_1, y_1), ..., (x_n, y_n)\}$
Salida: Conjunto consistente S de T

$S = \emptyset$
$S = centroides(T)$
Mientras $\Delta S \neq \emptyset$ **hacer:**
    $S = S \cup \Delta S$
    $\Delta S = \emptyset$
    Para cada $x \in S$ **hacer:**
        $\Delta S = \Delta S \cup \{rep(x, Voren(p, S, T))\}$
    Fin Para cada
Fin Mientras

Existen variaciones relacionadas a como elegir al representante de etiqueta diferente a ser agregado, por lo que se crean variaciones en el algoritmo como lo son FCNN1, FCNN2, FCNN3 y FCNN4, en el presente trabajo utilizamos FCNN1, pues de acuerdo a ensayos experimentales realizados en (Angiulli, 2005), es un algoritmo que muestra un buen rendimiento en cuanto a velocidad.

Algoritmo 4 FCNN1

Entrada: Conjunto de entrenamiento $T = \{(x_1, y_1), ..., (x_n, y_n)\}$
Salida: Conjunto consistente S de T

Para cada $p \in T$ **hacer:**
    $nearest[p]$ = indefinido
Fin Para cada
$S = \emptyset$
$S = Centroides(T)$
Mientras $\Delta S \neq \emptyset$ **hacer:**
    $S = S \cup \Delta S$
    Para cada $p \in S$ **hacer:** $rep[p]$ = indefinido
    Fin Para cada
    Para cada $q \in (T - S)$ **hacer:**
        Para cada $p \in \Delta S$ **hacer:**
            Si $d(nearest[q], q) > d(p, q)$ **entonces**
                $nearest[q]$ = p
            Fin Si
        Fin Para cada
        Si $(l(q) \neq l(nearest[q])$ && $(d(nearest[q], q) < d(nearest[q], rep[nearest[q]]))$ **entonces**
            $rep[nearest[q]] = q$;
        Fin Si
    Fin Para cada
    $\Delta S = \emptyset$
    Para cada $p \in S$ **hacer:**
        Si $rep[p]$ esta definido **entonces**
            $\Delta S = \Delta S \cup rep[p]$
        Fin Si
    Fin Para cada
Fin Mientras

En el algoritmo (4) el representante $(rep(p, Voren(p, S, T)))$ es simplemente la observación más cercana a p de etiqueta diferente en el espacio de Voronoi creado por p, en cuanto al pseudocódigo descrito, el vector *nearest* sera el encargado de almacenar el punto más cercano p en S de cada uno de los puntos del conjunto T, mientras que el vector *rep* sera el encargado de almacenar el punto mal clasificado más cercano a p en el espacio de voronoi formado por el punto p.

2.1.5. DROP3 (*Decremental Reduction Optimization Procedure*)

DROP 1-3 son metodologías de selección de instancias decrementales, de acuerdo con (D. R. Wilson y Martinez, 2000), DROP3 fue introducido originalmente bajo el nombre de RT3 en el trabajo publicado por (D. R. Wilson y Martinez, 1997). Para realizar la eliminación de instancias, el algoritmo se auxilia previamente de un filtrado ENN con el objetivo de remover ruido. Para explicar DROP3 utilizaremos los conceptos de vecinos más cercanos, asociados y enemigos más cercanos.

Para explicar el concepto de asociados es importante tener presente que cada instancia x tiene k vecinos más cercanos, por lo que los asociados de una instancia x son aquellas observaciones que tienen a x como uno de sus k vecinos más cercanos, por otra parte, utilizaremos la palabra enemigo más cercano cuando nos referimos a aquella instancia más cercana a x que tenga una etiqueta diferente.

La regla utilizada por el algoritmo es eliminar una instancia x si al menos tantos de sus asociados en S se clasificaron correctamente sin él, la regla de eliminación también se puede explicar bajo la siguiente analogía; si la instancia x es removida, ahora cada asociado buscará un $k+1$ vecino más cercano y si este $k+1$ vecino más cercano, clasifica bien al asociado de x, x no hará falta por lo que puede ser removido.

El algoritmo (5) bosqueja el funcionamiento de la metodología, los elementos importantes del algoritmo DROP3 y que de acuerdo con estudios realizados por (D. R. Wilson y Martinez, 2000), lo vuelven más atractivo de usar frente a DROP1 y DROP2 son:

- La eliminación de ruido al usar una regla similar a ENN, en donde cualquier instancia mal clasificada por sus k vecinos más cercanos se elimina, con ello eliminamos instancias ruidosas, puntos fronterizos cercanos, y suavizamos ligeramente el límite de decisión.

- La idea de ordenar las observaciones es que aquellas instancias que tienen a sus enemigos a una mayor distancia jueguen un rol menos importantes, por lo que

serán las primeras observaciones a considerar y posiblemente ser removidas, en un intento de eliminar los puntos centrales antes que los puntos fronterizos.

Algoritmo 5 DROP3

Entrada: Conjunto de entrenamiento $T = \{(x_1, y_1), ..., (x_n, y_n)\}$
Salida: Conjunto S de T

1: Eliminar ruido : Remover de X cualquier instancia mal clasificada por k vecinos más cercanos
2: S= X
3: Ordenar las instancias en S por la distancia al enemigo más cercano de mayor a menor
4: **Para cada** $x \in S$ **hacer:**
5: Sea CON = # de asociados de x correctamente clasificados por x
6: Sea SIN = # de asociados de x correctamente clasificados sin x
7: **Si** $SIN \geq CON$ **entonces**
8: Remover x de S
9: **Para cada** asociado **as**$'$ de x **hacer:**
10: Remover x de la lista de **as**$'$ como vecinos más cercano
11: Encontrar un nuevo vecino más cercano para **as**$'$
12: Añadir **as**$'$, a los asociados que corresponden a **as**$'$
13: **Fin Para cada**
14: **Fin Si**
15: **Fin Para cada**

2.2. *Big Data*

Un par de definiciones breves que se puede encontrar sobre lo que es *Big Data* son las siguientes:

- "*Big Data* quiere decir grandes volúmenes, gran velocidad y gran variedad"
  Stonebraker (como se citó en (Curry, 2016))

- "Cuando el tamaño de los datos llega a formar parte del problema y las tecnicas tradicionales para trabajar con datos no son tan eficientes"
  Loukides (como se citó en (Curry, 2016))

Al día de hoy, se han desarrollado varias distribuciones y tecnologías para sobrellevar lo que implica el *Big Data*, nosotros haremos énfasis en Apache *Spark*, abordaremos sus características y mencionaremos los inicios del marco de trabajo.

Botta (como se citó en (Oussous, Benjelloun, Lahcen, y Belfkih, 2018)) menciona que en la actualidad se genera un gran volumen de datos de diferentes fuentes, tales como el gobierno, el sector salud, las redes sociales, el sector financiero, así como de diferentes empresas del sector privado. Cuando grandes volúmenes de datos comenzaron a aparecer, las personas tuvieron que buscar los recursos que permitieran almacenar y acceder a los datos de manera eficiente y segura. Para darle valor a los datos, era necesario computar grandes volúmenes de información y ello debería hacerse posiblemente bajo otro paradigma de programación, ya que la mayoría de algoritmos de aprendizaje maquina o bien metodologías estadísticas para procesar información no estarían pensadas para el orden de cientos de miles o millones de registros.

2.2.1. Marco de trabajo *Spark*

El paradigma de programación *MapReduce* cobró relevancia en 2004, año en que *Google* propuso una simple pero potente interfaz que permite la paralelización automática y distribución de cálculos a gran escala utilizando funciones primitivas para resolver su problema de procesamiento e indexación de la información de Internet (Dean y Ghemawat, 2004).

Aunque *Google* ha mantenido en secreto el funcionamiento interno de *MapReduce* y el software de administración de archivos relacionado, las pocas publicaciones que realizó sobre técnicas subyacentes y el modelo de programación *MapReduce* fue suficiente para que Doug Cutting creara su propia versión de la tecnología, llamada *Hadoop* (Vance, 2009).

La tecnología *Hadoop* propone un sistema tolerante a fallos, es capaz de crear un *cluster* grande o pequeño y es relativamente fácil de escalar. *Hadoop* consiste de dos componentes: el almacenamiento y el procesamiento, donde para el primero se hace uso del Hadoop Distributed File System (HDFS) y para el procesamiento se hace uso del modelo *MapReduce* (Beakta, 2015).

Ahora bien, *Haddop* tiene dificultades en cuanto a la velocidad de consultas y ejecución de programas, por lo que *Spark* fue introducido con el propósito de ayudar con dicha problemática. En el 2009, en la Universidad de California en *Berkeley* dio inicio el proyecto Apache *Spark*, inicialmente el proyecto llevaba por nombre *Spark* y al frente de la investigación estaba Matei Zaharia, para el momento de ser lanzado como código abierto ya había logrado acelerar el proceso de software de cálculo computacional *Hadoop*, años después el proyecto fue donado a la empresa *Apache Software Foundation* lanzándolo como *Apache Spark*, a pesar de que tiene un modelo de programación similar al *MapReduce* agrega una abstracción de intercambio de datos llamada *Resilient Distributed Datasets* (RDDs)(Zaharia *et al.*, 2016).

Una de las ventajas por las cuales optar por Apache *Spark* como marco de trabajo hoy en día es la relativa facilidad que ofrece para ser usado de manera local o conectarse a un grupo de computadoras, expone API's para *JAVA*, *PYTHON*, R y *SCALA*, además cuenta con otros proyectos que lo vuelve una herramienta muy completa tal que sigue siendo actualizada y utilizada para el año 2022.

2.2.2. Paradigma de programación *MapReduce*

Como hemos comentado en la Sección anterior, el paradigma de programación *MapReduce* tiene un rol muy importante en el marco de trabajo *Spark* y se puede dividir en dos etapas:

Map: partiendo de que los datos se leen y están distribuidos en lo que es el cluster, los *mappers* son tareas individuales que aplican un procesamiento específico a cada partición lo que resulta comúnmente pero no necesariamente en pares llave-valor.

Reduce: usualmente la función *reduce* se auxilia previamente de una función *map*, quien es la encargada de asignar los pares llave-valor por lo que para este punto reducir es el encargado de realizar la fusión y en ocasiones algún cálculo que se aplica a los datos con la misma clave.

Implícitamente en el paso *reduce* los datos son barajeados tal que se combinan todos los valores en una colección asociada a la misma clave, además se realiza un ordenamiento de las claves antes de ser presentadas a reducer.

La figura (2.2) ilustra un ejemplo de como realizar el cálculo de promedios bajo el paradigma *MapReduce*.

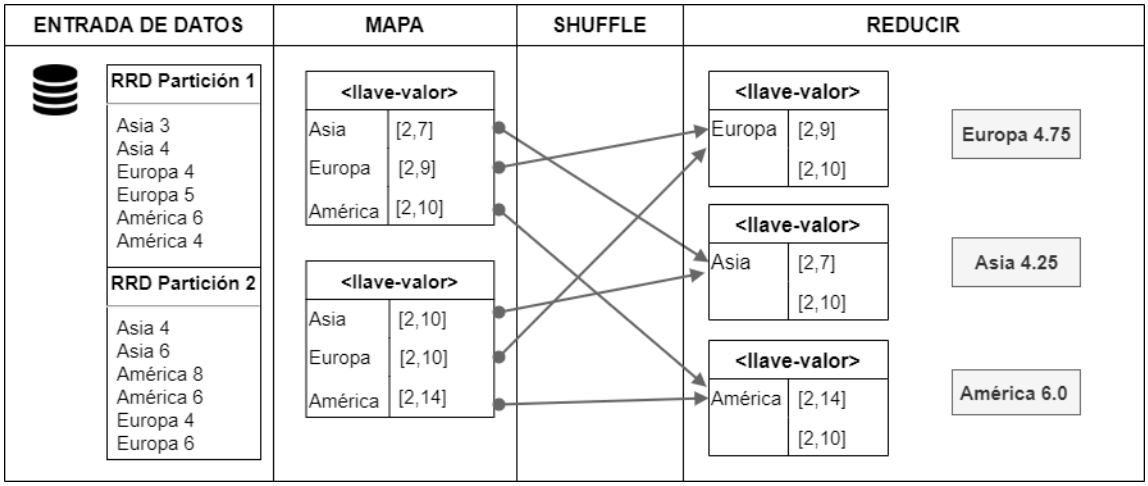

Figura 2.2: Ejemplo de calculo de promedios bajo el paradigma *MapReduce*. Imagen Adaptada (Im *et al.*, 2013)

Es importante considerar el hecho de que las funciones *map* corren independientemente, mientras que las operaciones *reduce* comienzan una vez que los respectivos *mappers* han terminado. Cuando en los experimentos se tiene un mayor número de *mapperss* definidos y se supera el número de esclavos disponibles en el *cluster*, el sistema pone en cola las tareas restantes y se envían tan pronto cualquier tarea *map* haya terminado su procesamiento.

Las operaciones en Spark se pueden clasificar en dos tipos: transformaciones y acciones. Las transformaciones, generan un nuevo rdd o *data frame* otorgándole la propiedad de inmutabilidad. Dicho de otra manera, estas operaciones no cambiaran el objeto original; en su lugar, devolverán los resultados transformados de la operación como un nuevo *data frame* o rdd. Todas las transformaciones se evalúan perezosamente, es decir, sus resultados no se computan inmediatamente, sino que se registran o recuerdan y la ejecución es retrasada hasta que se invoque una acción (Damji, Wenig, Das, y Lee, 2020).

El cuadro (2.1) contiene una muestra de transformaciones y acciones, si se desea tener información más detallada de la clasificación de las funciones sugerimos revisar (W. Feng, 2019).

Transformaciones	Acciones
filter()	*count()*
sample()	*save()*
join()	*take()*
randomsplit()	*collect()*

Tabla 2.1: Ejemplos de transformaciones y acciones en MapReduce

2.2.3. Enfoque Global y Local

Como ya hemos mencionado, el paradigma de programación *MapReduce* distribuye la información en particiones con el objetivo de poder procesar la información, ahora bien, dichas particiones pueden compartir información entre ellas o bien actuar independientemente. Un enfoque local es aquel que no considera la información relacionada de otras particiones por lo que los algoritmos diseñados bajo este enfoque son aproximaciones, por el contrario un enfoque global conoce la información de cada partición.

Enfoque Global y Local en Selección de instancias

Tal y como se menciona en (Triguero, Peralta, Bacardit, García, y Herrera, 2015), los métodos de selección de instancias necesitan almacenar en la memoria principal cálculos parciales, soluciones intermedias, en ocasiones el conjunto de entrenamiento completo, por lo que se motiva el uso de procedimientos de particionamiento distribuido es decir el uso de un enfoque local sobre el uso de un enfoque global. Sin embargo, las limitaciones que podría presentar un enfoque local son el hecho de que no se puede mantener la proporción de ejemplos por clase de T dentro de cada subconjunto. Otra limitación no menos importante, es el hecho de que al unir los conjuntos reducidos localmente pudiéramos tener ejemplos redundantes y/o ruidosos.

Otras observaciones a considerar bajo un enfoque local y que son reportadas por Triguero es el hecho de que cuando el número de instancias por partición supera las veinte mil instancias, la ejecución de ciertos algoritmos de selección de instancias no es factible en el tiempo, en contraparte, cuando el número de particiones aumenta y pocas instancias se encuentran en cada partición, el rendimiento de la métrica exactitud podría disminuir y aunque esto depende del problema en especifico que se aborde, pero la afirmación podría aplicar a la mayoría de los problemas.

2.3. Selección de instancias y *Big Data*

Para el año 2007 ya se comentaba y visualizaba la dificultad de aprovechar por completo los datos con los que se contaba debido al aumento drástico en las capacidades de recopilación de información, por lo que uno de los enfoques más comunes en esos días era el de dividir el conjunto de datos T en subconjuntos pequeños, pues versiones paralelas o distribuidas de algoritmos no se encontraban disponibles, para realizar la implementación de un algoritmo distribuido se tenía que proponer arquitecturas nuevas y auxiliarse de paqueterías muy especializadas de cómputo paralelo y distribuido (Angiulli y Folino, 2007).

2.3.1. MPPR

MapReduce para Reducción de Prototipos (MRPR, por sus siglas en ingles) es una propuesta realizada por Triguero *et al.* (2015) quien introduce el paradigma de programación *MapReduce* como una posible solución para tratar el problema de selección de instancias en grandes volúmenes de datos, se plantea utilizar la fase mapeo para llevar a cabo la división de los datos en conjuntos disjuntos y aplicar una técnica de reducción de instancias, en la etapa de reducción se plantea utilizar un filtrado o fusión de instancias con el propósito de evitar la introducción de prototipos redundantes o que confundan la verdadera selección de instancias.

Algoritmo 6 MRPR

Entrada: Conjunto de entrenamiento T, Número de particiones j, Tipo De Reducción
Salida: S

```
- - - - Fase de Mapeo
Formar T_j conjuntos j = 1, ..., m
Para cada T_j hacer:
    S_j = ReducciónDeInstancias (T_j)
Fin Para cada
- - - - Fase de Reducción
Para cada Sj hacer:
    S = S ∪ S_j
Fin Para cada
Si Tipo de reducción == 'Filtro' entonces
    S = Filtro(S)
Fin Si
Si Tipo de reducción == 'Fusión' entonces
    S = Fusión(S)
Fin Si
Return S
```

La idea de MRPR es dividir el conjunto de datos T en m particiones disjuntas T_j, $j = 1, ..., m$, donde m bajo el escenario ideal es igual al numero de *mappers* disponibles en la maquina o conjunto de maquinas, después de aplicar la metodología de selección de instancias en la fase de reducción se plantean 3 opciones para unir los conjuntos reducidos.

- Unir: plantea simplemente concatenar los conjuntos disjuntos reducidos
- Filtrar: plantea concatenar los conjuntos reducidos para posteriormente aplicar un suavizador de selección de instancias, por ejemplo ENN
- Fusión: después de concatenar los conjuntos reducidos plantea generar observaciones basadas en el centroide a partir de las observaciones existentes y que sustituyen a las mismas.

El algoritmo (6) así como la figura (2.3) bosqueja el planteamiento realizado, el trabajo se plantea originalmente en el marco de trabajo *Mahout*, es de interés particular evaluar el desempeño así como el comportamiento del clasificador KNN y se presta particular atención a la metodología de reducción de instancias SSMA-SFLSDE.

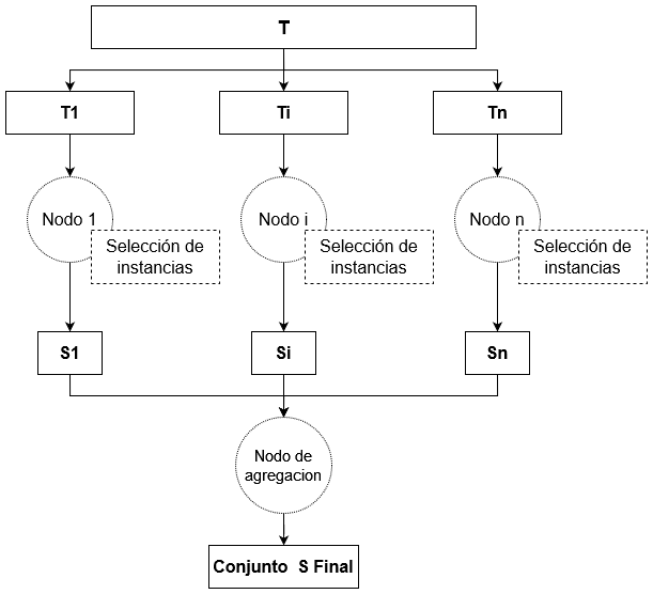

Figura 2.3: Flujo de trabajo MRPR. Imagen adaptada (Si *et al.*, 2017)

2.3.2. DS-RNGE

Entre los primeros trabajos enfocados a reducción de instancias para grandes volúmenes de datos, específicamente utilizando el marco de trabajo *Spark* podemos encontrar el que lleva por nombre 'Clasificación de vecinos más cercanos de alta velocidad para datos en *streaming* usando Spark' (Ramírez-Gallego *et al.*, 2017), entre las

aportaciones del autor destaca la modificación al clasificador KNN con el propósito de mejorar el rendimiento del mismo, el trabajo se enfoca a lograr el procesamiento de grandes secuencias de instancias, pensando que se tienen observaciones potencialmente ilimitadas y ordenadas que llegan con el tiempo, para conseguir el objetivo mejorar constantemente el rendimiento a través del tiempo se propone hacer uso del algoritmo de selección de instancias *Relative neighborhood graph edition* (RNGE).

La propuesta consta de 3 partes, la primera consiste en particionar la base datos e inicializar una base de casos distribuida, posteriormente se realiza el proceso de edición/actualización que involucra directamente al método RNGE y por ultimo tenemos la fase de predicción para nuevas instancias.

La construcción de arboles métricos (*M-trees*) en la primera etapa es importante porque son utilizados para disminuir la complejidad de la búsqueda del clasificador k-NN, se logran búsquedas más eficientes saltándose una gran cantidad de comparaciones. Del primer lote de datos, se toma una muestra de instancias para construir el árbol principal en la maquina maestra, mientras que el resto de instancias se utiliza para construir arboles locales en las maquinas esclavas siempre consultando al árbol superior.

La segunda etapa de edición y sustitución, se ejecuta cuando un nuevo lote de datos ingresa, el algoritmo calcula en qué subárbol cae cada elemento nuevo, por lo que una vez que todas las instancias son barajadas en subárboles, se inicia una búsqueda local de vecinos más cercanos para cada elemento en los subárboles correspondientes, se forman grupos, donde cada uno está formado por él nuevo elemento y sus vecinos. Luego, se aplica RNGE de manera local a cada grupo formado, es decir, la idea es construir un grafo local alrededor de cada grupo y decidir qué instancias deben insertarse, eliminarse o dejarse intacta, el algoritmo RNGE permite controlar el número de vecinos para la construcción de los grafos(grupos) a través de un parámetro, en general se cumple que a un mayor numero de vecinos más precisa y lenta será la eliminación.

Omitimos el pseudocódigo relacionado al proceso de partición inicial y al proceso

de predicción, lo relacionado al proceso de edición y actualización se presenta en el pseudo algoritmo (7).

Algoritmo 7 Proceso de edición y actualización en DS-RNGE

Entrada: *Lote*, ks (numero de vecinos a utilizar en la fase de selección)
Salida: conjunto de instancias actualizado

1: Sea e una instancia
2: **Para cada** $e \in Lote$ **hacer:**
3: Encontrar el nodo de hoja más cercano a e en las hojas del árbol principal y retornar una tupla con el ID del árbol (clave) y e (valor) $< ID, e >$. (***Map***)
4: La tupla se envía a su subárbol correspondiente de acuerdo a su clave (***Shuffle***)
5: grupo = para cada instancia se retorna una tupla con e (clave) y una lista de sus ks vecinos (valor) $< e, vecinos\ cercanos >$ (***Reduce***)
6: editado = aplicar de manera local el algoritmo RNGE a cada tupla del paso anterior
7: **Fin Para cada**
8: Actualizar el conjunto de instancias y con ello los arboles métricos.

La ultima etapa del algoritmo se inicializa cuando llegan nuevos datos sin etiquetar, el algoritmo se auxilia del árbol principal y subárboles creados en la primera etapa para recuperar los k vecinos de cada nueva observación, finalmente el algoritmo predice la clase del elemento aplicando el esquema de votación mayoritaria.

La aplicación realizada del algoritmo RNGE es una aproximación en el sentido de que cada grafo creado tiene solo una vista estrecha de la base de datos, la propuesta sugerida por el autor es limitar el conjunto de vecinos a ser eliminados a aquellos que comparten una arista(borde) con el nuevo elemento, además, la propuesta contempla controlar la eliminación de ejemplos antiguos para cada lote de datos nuevos.

2.3.3. MR-DIS

Otro antecedente que se tienen sobre selección de instancias en *Big Data* corresponde al trabajo realizado por González entre los años 2016-2018, González (2018) realizó una investigación en selección de instancias para la tarea de regresión, haciendo énfasis en que la mayoría de los métodos implementados en la actualidad son para la tarea de clasificación, aun así, el investigador destaca que la mayoría de los

métodos de selección incluso para la tarea de clasificación no se pueden usar con conjuntos de datos grandes, por lo que también aborda este problema con propuestas como *MapReduce Democratic Instance Selection* (MR-DIS).

MR-DIS es una adaptación al paradigma de programación *MapReduce* del algoritmo Selección Democrática de Instancias (DIS por sus siglas en ingles) propuesto por García-Osorio, de Haro Garcia, y García-Pedrajas (2010). La ejecución del algoritmo consiste en completar un número de rondas r, en cada una de las cuales el conjunto original T se divide en una serie de particiones y a cada una se le aplica un algoritmo de selección de instancias de forma independiente, las instancias seleccionadas a ser eliminadas por el algoritmo reciben un voto, el proceso se repite durante un número de rondas predefinidas por el usuario y las instancias con un número de votos superior a un umbral calculado son eliminadas para obtener el conjunto S final.

Como hemos comentado, es una metodología que tiene sus bases en el principio de divide y vencerás, la determinación del umbral con el propósito de identificar el número óptimo de votos a partir de los cuales eliminar instancias es independiente para cada conjunto de datos, omitiremos los detalles del mismo pero comentamos que se auxilia de una validación cruzada a un subconjunto pequeño de los datos.

Por ultimo, mencionamos que la propuesta original de los autores para llevar a cabo MR-DIS se ejecuta con el algoritmo CNN y se deja en claro que la propuesta es fácil de extender a otros algoritmos de selección de estancias (Arnaiz-González *et al.*, 2017).

El bosquejo de cómo se realiza el cálculo se puede ver en el pseudocódigo (8) y la figura (2.4) complementa la explicación con un ejemplo.

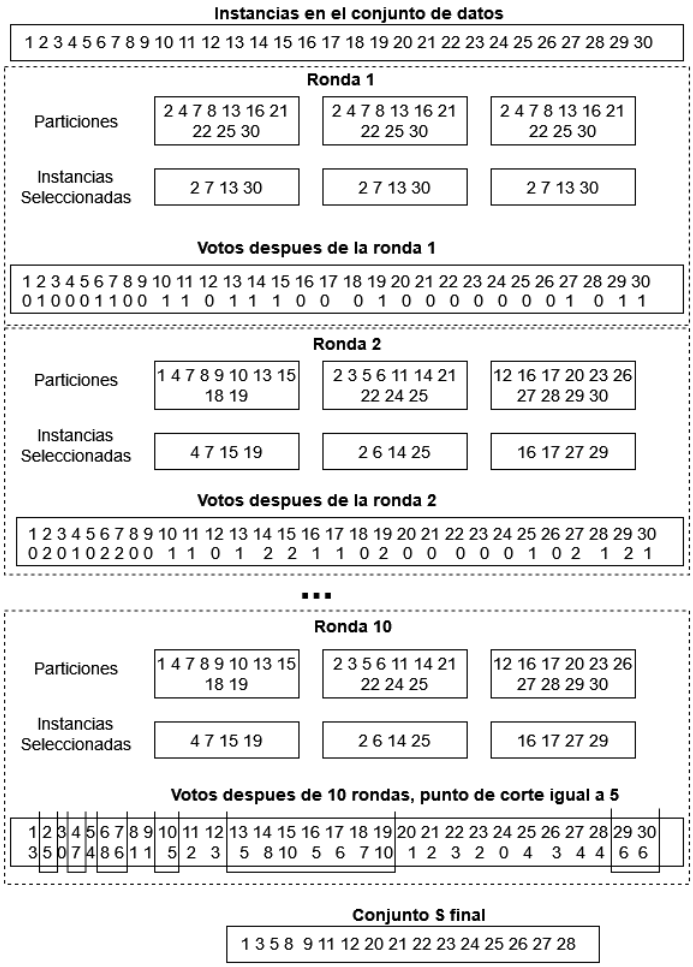

Figura 2.4: Ejemplo de ejecución de un algoritmo DIS en un conjunto de 30 instancias. Imagen Adaptada (Arnaiz-González *et al.*, 2017)

Algoritmo 8 Selección Democrática de Instancias *MapReduce* (MR-DIS)

Entrada: Conjunto de entrenamiento T
Salida: Conjunto reducido S

1: **Para cada** $k = 1...r$ **hacer:**
2: Dividir el conjunto T en n subconjuntos disjuntos $t_i : \bigcup_{i=1}^{n} t_i = T$
3: **Para cada** $i = 1...n$ **hacer:**
4: Aplicar Selección de instancias a t_i
5: Asignar votos a las instancias eliminadas de t_i
6: **Fin Para cada**
7: **Fin Para cada**
8: Obtener el punto de corte de los votos v
9: S = T
10: Remover de S todas las instancias con un numero de votos mayor o igual a v

2.3.4. FCNN_MR

FCNN_MR (Fast Condensed Nearest Neighbor *MapReduce*) es un método de selección de instancias de ultima generación propuesto por Si *et al.* (2017), la metodología retoma las bases de divide y vencerás propuestas en MRPR (Ver Subsección 2.3.1) y a diferencia del mismo, busca eliminar las instancias ruidosas y redundantes que pudieran resultar de considerar subconjuntos disjuntos, la propuesta hace posible la comunicación entre nodos en un marco de trabajo como *Spark* por lo que FCNN-MR utiliza todo el conjunto de entrenamiento y busca paralelizar el procedimiento de búsqueda, tal que el resultado del conjunto reducido es el mismo que en la versión secuencial de FCNN. La carga de trabajo en los nodos se reduce a medida que el numero de *mappers* aumenta por lo que se espera un buen rendimiento de escalabilidad.

Para lograr lo anterior plantea un procedimiento iterativo tal que en cada iteración el conjunto reducido S es actualizado, el costo computacional es evidentemente mayor pues cada iteración comprende la ejecución de la fase *map* y *reduce* (Véase Figura (2.5)).

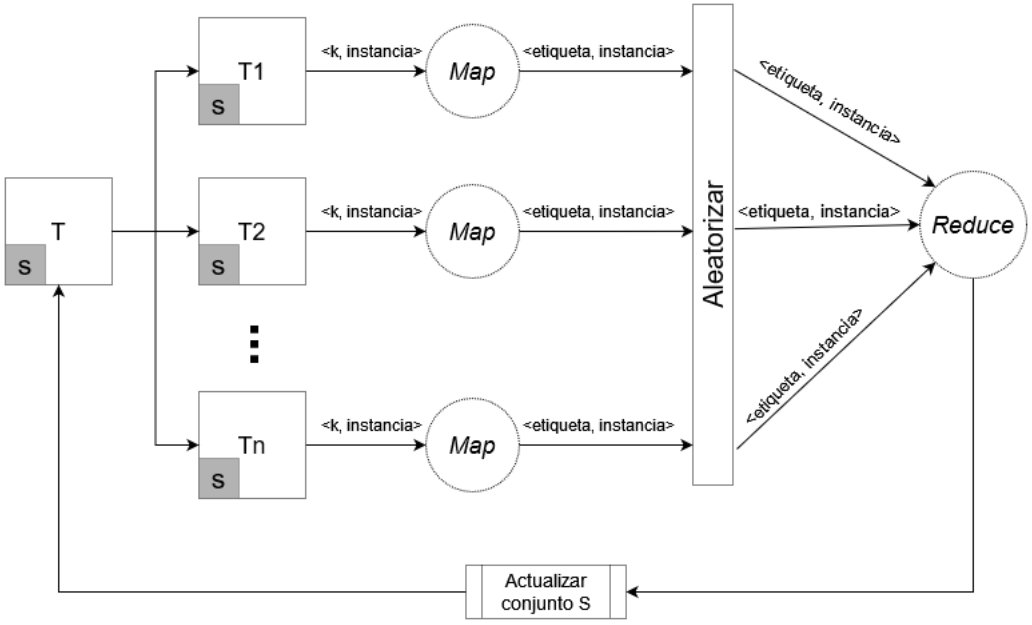

Figura 2.5: Flujo de trabajo del algoritmo FCNN_MR. Imagen Adaptada (Si *et al.*, 2017)

El resultado de la propuesta MRPR es subconjunto S aproximado, en dado caso se

intente considerar los filtros propuestos que tienen como propósito mitigar instancias ruidosas o aumentar la compresión del conjunto de datos; sea cual sea la elección, el resultado desemboca en una aproximación que estará condicionado al numero de particiones.

El mecanismo de votación MR-DIS (Véase Subsección 2.3.3) aborda de cierta manera el problema de considerar subconjuntos disjuntos pero no se implementa el algoritmo tradicional de selección de instancias de forma paralela y de acuerdo con Si *et al.* (2017) el numero de instancias retenidas aumenta a medida que se consideran un mayor numero de particiones

El algoritmo 9 bosqueja el funcionamiento de FCNN_MR. El algoritmo se inicializa igual que en FCNN, esto es con los centroides de las clases etiquetadas, posteriormente en cada *mapper* se computa la distancia así como el vecino más cercano de cada elemento del conjunto S, tal que para este punto podamos tener a los nuevos candidatos a agregar en la siguiente iteración, los candidatos serán los enemigos más cercanos del conjunto S, más adelante, en la fase *reduce* se elije a un único candidato de entre todos los candidatos de cada *mapper*.

Algoritmo 9 *FCNN_MR*

Entrada: Conjunto de entrenamiento T
Salida: Conjunto reducido S

1: $S = \emptyset$
2: $\Delta S = \emptyset$
3: Agregar los centroides de cada clase a ΔS
4: **Mientras** $\Delta S \neq \emptyset$ **hacer:**
5: $S = S \cup \Delta S$
6: $\Delta S = \emptyset$
7: **Para cada** instancia $I \in S$ **hacer:**
8: Ejecutar una función *mapper* a procesar (Algoritmo 10)
9: **Fin Para cada**
10: Unir todos los resultados intermedios e invocar una función *reducer* (Algoritmo 11) para generar ΔS.
11: **Fin Mientras**

Algoritmo 10 *mapper FCNN_MR*

Entrada: $< k, instancia >$, subconjunto S
Salida: null o $< etiqueta, instancia >$
1: Obtener la clase c de la instancia I
2: Predecir la clase c^* de I, por el vecino más cercano en S
3: **Si** $c \neq c^*$ **entonces**
4: Emitir el par $< etiqueta, instancia >$
5: **Fin Si**

Algoritmo 11 *reduce FCNN_MR*

Entrada: $< etiqueta, [instancia] >$, subconjunto S
Salida: Conjunto temporal ΔS
1: $\Delta S = \emptyset$
2: **Para cada** $< etiqueta, [instancia] >$ **hacer:**
3: Agregar el enemigo más cercano a ΔS
4: **Fin Para cada**

PROPUESTA DEL ALUMNO EN EL PSEUDOCÓDIGO

Algoritmo 12 *FCNN_MR*

Entrada: Conjunto de entrenamiento T
Salida: Conjunto reducido S
1: Crear $T_i, i = 1, .., n$ particiones disjuntas de T
2: $S = \emptyset$
3: $\Delta S = \emptyset$
4: Agregar los centroides de cada clase a ΔS
5: **Mientras** $\Delta S \neq \emptyset$ **hacer:**
6: $S = S \cup \Delta S$
7: **Para cada** partición T_i **hacer:**
8: Ejecutar función *mapper* (Algoritmo 13)
9: **Fin Para cada**
10: Unir todos los resultados intermedios (*Shuffle*)
11: Ejecutar la función *reducer* (Algoritmo 14) para generar ΔS.
12: **Fin Mientras**

Algoritmo 13 *mapper FCNN_MR*

Entrada: T_i, subconjunto S
Salida: null o $< idx, instancia >$

1: **Para cada** $instancia \in T_i$ **hacer:**
2: Obtener la clase c de la instancia I
3: Predecir la clase c^* de la instancia, por el vecino más cercano en S
4: **Si** $c \neq c^*$ **entonces**
5: **Retornar** $< idx, instancia >$, donde idx es el índice de S asociado a c^*
6: **Fin Si**
7: **Fin Para cada**

Algoritmo 14 *reduce FCNN_MR*

Entrada: $< idx, instancia >$, Subconjunto S
Salida: Conjunto temporal ΔS

1: $\Delta S = \emptyset$
2: **Para cada** $< idx, instancia >$ **hacer:**
3: Agregar el enemigo más cercano a ΔS
4: **Fin Para cada**

Capítulo 3

Espacios *kernel*

La presente Sección introduce conceptos y notación importante relacionada a los espacios de Hilbert, además, se exponen las ideas de como realizar la selección de instancias utilizando *kernels*.

La utilización de espacios de Hilbert se ha encontrado increíblemente útil en la comunidad de aprendizaje automático, la teoría existe desde hace tiempo y se ha utilizado en la literatura estadística durante al menos veinte años. Más recientemente, su aplicación a algoritmos de estilo perceptrón, así como a algoritmos de máquinas de soporte vectorial en donde se aprovecha el "truco del kernel" para realizar aprendizaje lineal en espacios no lineales.

3.1. Definiciones

Daremos por hecho que el lector esta familiarizado con los conceptos de Campo y Espacio Vectorial, para más detalles se puede consultar (Daumé III, 2004).

Definición 1: Norma

Sea V un espacio vectorial, una norma es una función en un espacio vectorial que satisface las siguientes propiedades para todo $u, v \in V$ y todo $a \in \mathbb{R}$:

- No negativa $\|u\| > 0$

- Homogeneidad $\|au\| = |a|\|u\|$
- Desigualdad del triangulo $\|u + v\| \geq \|u\| + \|v\|$

Definición 2: Producto Escalar

Sean x, y vectores en $\mathbb{R}^n$, el producto escalar es definido como:

$$\langle x, y \rangle = \sum_{i=1}^{n} x_i y_i$$

Sea $d : X \times X \to \mathbb{R}$ una función que toma dos elementos de el campo X y produce una "distancia" entre ellos.

Definición 3: Espacio de Banach

Un espacio de Banach es un espacio vectorial completo V dotado de un método para calcular el "tamaño"de los vectores en V, donde al tamaño se le llamara norma y la norma de $v \in V$ se denotara por $\|v\|$.

Definición 4: Espacio de Hilbert

Sea $\mathcal{H}$ un espacio de Hilbert, un espacio de Hilbert es un espacio de Banach y que posee una operación de producto escalar.

Definición 5: Matriz de Gram

Dada una función $k : X \times X \to \mathbb{R}$ y $\phi(\cdot) : X \to \mathcal{H}$, donde las observaciones $x_1, ..., x_m \in X$, la matriz K de $m \times m$ con elementos

$$K_{ij} = k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{H}}$$

se conoce como matriz de Gram o matriz *kernel* de k con respecto a $x_1, ..., x_n$ (Forsbach, 2005).

De la ultima definición, tenemos que dada una función K, existe una función $\phi(.)$, tal que la evaluación del *kernel* en los puntos x_i y x_j es equivalente a tomar el producto escalar entre $\phi(x_i)$ y $\phi(x_j)$ en algún espacio de Hilbert, a esta última propiedad es a

lo que se le suele denominar el truco del *kernel*.

Ejemplos de una función kernel son:

- kernel Polinomial

$$k(x, y) = (1 + \langle x, y \rangle^p)$$

- kernel Gaussiano o kernel Función de Base Radial (RFB, por sus siglas en ingles)

$$k(x, y) = exp\{-\frac{||x - y||^2}{2\sigma^2}\} = exp\{-\gamma||x - y||^2\}$$

- kernel Sigmoidal

$$k(x, y) = tanh(\alpha \langle x, y \rangle + \beta)$$

3.2. *Kernel* KNN

El algoritmo KNN es simple e intuitivo, por ende puede tener muchas variaciones, una estas variaciones es por ejemplo el uso del *kernel* a la par de KNN, por si solo el algoritmo KNN ha demostrado un buen desempeño en problemas no lineales, pero cuando la distribución de muestra es arbitraria perderá potencia, por lo que KNN se recomienda en problemas donde los datos tienen una distribución similar a una esfera, ahora bien, el mapeo no lineal que implica el uso de un kernel puede cambiar la distribución de la muestra por lo que el desempeño del clasificador puede mejorar su rendimiento.

Una propiedad importante que se puede mostrar analíticamente es el hecho que la distancia en el nuevo espacio de características se puede calcular utilizando una función kernel (Véase ecuación 3.1). Además, si consideramos el algoritmo Kernel KNN, bajo un kernel adecuado se puede garantizar que los resultados no sean siempre peores que los del algoritmo del vecino más cercano convencional (Yu *et al.*, 2002).

La distancia, entre dos vectores x y y es denotado y calculado como:

$$d(x, y) = \|x - y\|$$

El cuadrado de la distancia anterior en el espacio de Hilbert puede ser expresado en términos de productos internos de la siguiente manera:

$$\begin{aligned} d^2(\phi(x), \phi(y)) &= \|\phi(x) - \phi(y)\|^2 \\ &= \langle \phi(x) - \phi(y), \phi(x) - \phi(y) \rangle \\ &= \langle \phi(x), \phi(x) \rangle + \langle \phi(y), \phi(y) \rangle - 2 \langle \phi(x), \phi(y) \rangle \\ &= K(x, x) - 2K(x, y) + K(y, y) \end{aligned} \tag{3.1}$$

3.2.1. Distancia utilizando el kernel RBF

Para el caso particular del kernel RBF, $K(x, x) = 1 = K(y, y)$, por lo que la ecuación (3.1) se reduce a

$$d^2(\phi(x), \phi(y)) = 2 - 2K(x, y) \tag{3.2}$$

La función kernel RBF calcula la similitud entre dos puntos, esto es que tan cerca están entre los dos

$$K(x, y) = exp\{-\frac{\|x - y\|^2}{2\sigma^2}\} = exp\{-\gamma ||x - y||^2\}$$

Sea $d_{xy} = \|x - y\|^2$, entonces

$$K(x, y) = exp\{-\frac{d_{12}}{2\sigma^2}\} = exp\{-\gamma \ \ d_{12}\}$$

Si $x = y$, entonces $d_{12} = 0$, en tal situación el valor máximo del kernel es 1, en contra parte si los puntos están separados, el valor del núcleo es menor que 1 y cercano a 0 lo que significa que los puntos son diferentes. Es importante encontrar el valor

correcto de σ para decidir que puntos deben considerarse similares.

Por ejemplo para un σ pequeño, es decir un γ con un valor grande la mayoría de los puntos caerán en la zona de disimilaridad por lo que de acuerdo con (3.2) las distancias serán iguales a 2.

3.3. Propuesta de uso de los espacios de Hilbert para realizar la selección de instancias

Incorporación de la distancia en el espacio de Hilbert a los algoritmos

La primera propuesta de cómo utilizar los espacios kernel para realizar la selección de instancias consiste en utilizar el truco del *kernel* y no visitar explícitamente el espacio de características, es decir, calcularemos la distancia entre observaciones en el espacio de Hilbert utilizando solo productos internos y con dichas distancias realizar la selección de instancias para obtener un conjunto reducido S. El bosquejo gráfico de la propuesta puede visualizarse en la Figura 3.1.

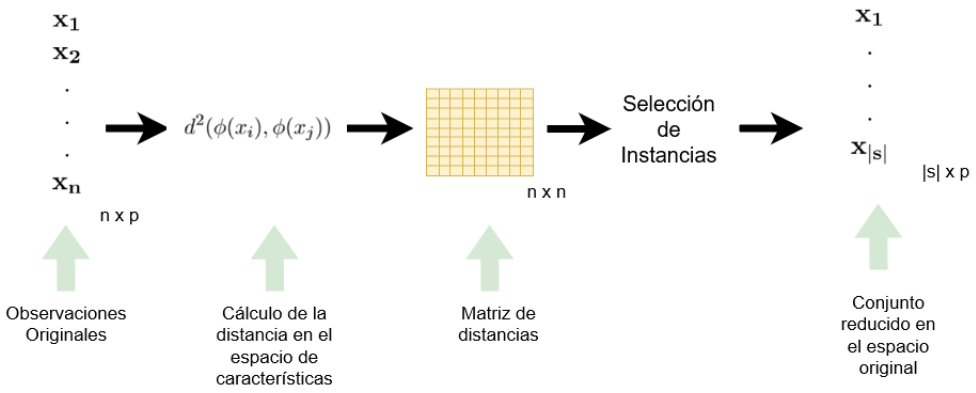

Figura 3.1: Uso de la distancia en el espacio de características

Uso de *Kernel* PCA para realizar la selección de instancias

La segunda propuesta consiste en utilizar la técnica *Kernel* PCA, la cual es una generalización de análisis de componentes principales, el objetivo es que nuestros datos tabulares originales, adopten una estructura diferente a partir de la matriz de Gram y la estructura que adoptará dependerá en gran medida de los p' componentes principales que se decidan utilizar, la selección de instancias se realizará con esta nueva estructura de los datos y finalmente volveremos al espacio original.

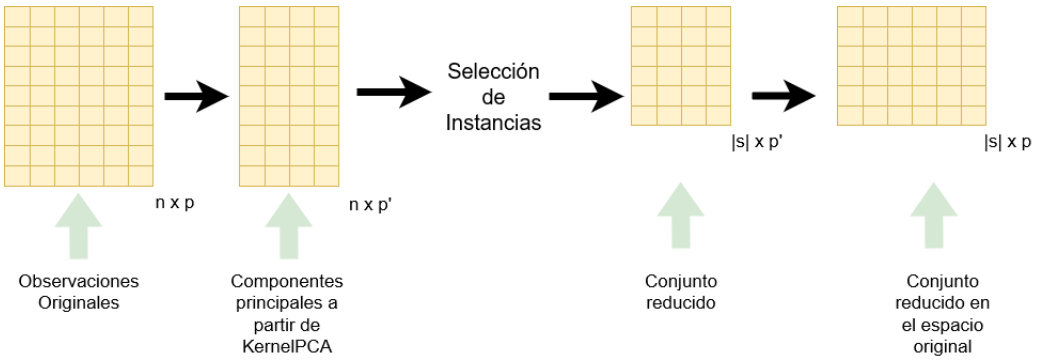

Figura 3.2: Adaptación de PCA para realizar la selección de instancias

3.3.1. Modificación del algoritmo FCNN1

KFCNN

La modificación del algoritmo para utilizar la distancia en el espacio kernel es cómo se describe a continuación, recordamos del algoritmo (4) que hay dos etapas claves para seleccionar las instancias del conjunto T a ser agregadas en S, la primera ocurre cuando para cada instancia en T se busca la instancia más cercana que esta en S, para ello se utiliza un criterio de distancia euclidiana, la segunda etapa es valorar para una instancia $p \in S$ de entre todos los posibles candidatos $q \in T$ aquella que tenga una etiqueta diferente y tenga la distancia euclidiana mínima a p, por lo que la modificación consiste en utilizar la distancia en el espacio de características en ambas etapas, llamaremos a dicha modificación kFCNN.

KPCAFCNN

Consiste en proyectar nuestra matriz de datos a un espacio de mayor dispersión para posteriormente realizar PCA en dicha dimensión, la modificación entonces consiste en utilizar esta ultima matriz para realizar la selección de instancias de acuerdo con el algoritmo (4), en este punto volvemos a retomar las distancias bajo un espacio euclidiano, bajo la suposición de que las características de interés ya han sido resumidas en las columnas de los componentes principales.

3.3.2. Modificación del algoritmo DROP3

KDROP3

En la metodología DROP3 (Véase 5), el calculo de asociados y de vecinos más cercanos a una determinada instancia se realiza considerando la distancia euclidiana, en este caso lo que planteamos es encontrar los vecinos más cercanos así como los asociados utilizando la distancia en el espacio de características.

KPCADROP3

Una vez teniendo la matriz PCA que se obtiene de un espacio de características de alta dimensión, la idea es encontrar a los asociados y vecinos más cercanos utilizando dichos datos tabulares.

3.4. Datos no balanceados

En la practica es común encontrarse con datos no balanceados, podemos encontrar estos datos cuando tenemos eventos relativamente raros o eventos que no ocurren en la misma proporción, por ejemplo el numero de transacciones bancarias no legitimas suele ser menor al numero de transacciones verídicas, en la medicina es común que ciertas enfermedades patológicas solo llegan a afectar a una pequeña parte de la población total, por lo que intuitivamente será más difícil recopilar información de la clase minoritaria.

La implementación ingenua de un método de clasificación a menudo conduce a

resultados de predicción insatisfactorios en los datos de prueba cuando tenemos la existencia de clases no equilibradas, hoy en día, se han propuesto múltiples técnicas de remuestreo para abordar los problemas (Y. Feng, Zhou, y Tong, 2020). Nuestra modificación al algoritmo FCNN busca ser una alternativa para lidiar con estas situaciones.

MODIFICACIÓN AL ALGORITMO FCNN

Cuando los tamaños de muestra de diferentes clases están desequilibrados en los datos de entrenamiento, proponemos inicializar el conjunto S considerando todos los elementos de la clase minoritaria, tal que los nuevos elementos de la clase mayoritaria serán elegidos por una lógica basada en distancias ademas, se podra seguir considerando la propuesta de distancias en el espacio de hilbert planteada en la Sección 3.3.

Capítulo 4

Diseño experimental y análisis de resultados

4.1. Estudio Experimental

En la presente Sección describimos la configuración usada en los diferentes algoritmos de Selección de Instancias (IS por sus siglas en Ingles), cada uno de los experimentos se realizaron en el marco de trabajo Apache Spark a través de la API en el lenguaje de programación *Python*.

Para medir el desempeño de los diferentes algoritmos I.S., se consideran varias métricas tales cómo: la exactitud, la métrica macro f1, tiempos de ejecución y la reducción del conjunto de datos(véase Sección A). Con el propósito de tener una mejor idea del rendimiento esperado en nuestras metodologías, se realiza una validación cruzada k-pliegues con k igual a 5 (Véase Apéndice A.2).

Los métodos FCNN y DROP3 que se toman de referencia son implementaciones que están basadas en los respectivos trabajos de (Angiulli, 2005) y (D. R. Wilson y Martinez, 1997), los métodos que plantean utilizar la distancia en el espacio de características llevaran el prefijo K y KPCA, tal que las propuestas comentadas en el Capítulo 3 serán nombradas como KFCNN, KPCAFCNN, KDROP3 y KPCA-

DROP3. Para estas metodologías se utiliza la propuesta MR-PR (Véase Subsección 2.3.1) idea basada en un enfoque de divide y vencerás, particularmente en nuestros experimentos decidimos utilizar *mappers* de tamaño 2000 , es decir el numero de *mappers* estará en función del numero de datos en el conjunto T. Por ultimo, para contrastar el desempeño entre un enfoque exacto y un enfoque aproximado, se realiza la evaluación de la metodología FCNN_MR (Véase Subsección 2.3.4)

El conjunto reducido S que se obtiene de la etapa anterior, es evaluado en 6 clasificadores: K Vecinos más cercanos, Bosques Aleatorios, Regresión logística, Redes Neuronales, Maquinas de Soporte Vectorial utilizando el kernel RBF y el kernel lineal (Véase B), la configuración usada para cada clasificador se resumen en la Tabla (4.1),

CLASIFICADOR	**Parámetro**	**Valor**
KNN	k vecinos	3
Bosques Aleatorios	Profundidad Maxima Criterio	5 “gini”
Regresion Logistica	Maximo de iteraciones Optimizador	5000 “lbfgs”
Red Neuronal	Capas intermedias Numero de neuronas Numero Máximo de iteraciones	1 100 200
SVM Kernel(“RBF”)	γ Parámetro de regularización : C	[0.0005,0.005,0.05] 1
SVM Kernel(“LINEAL”)	Parámetro de regularización : C	1

Tabla 4.1: Configuración utilizada para los clasificadores

En cuanto a la infraestructura utilizada para realizar los experimentos con Apache Spark atraves de la API en Python, se utilizó el clúster del Centro de Investigación en Matemáticas Unidad Monterrey. La maquina proporcionada consta de 12 núcleos y 64 GB en memoria RAM bajo Linux distribución Ubuntu 20, específicamente decidimos utilizar con Apache Spark 3.0.3 y Haddop 2.7.

Cada uno de los algoritmos descritos en : 4.1.1,4.1.2,4.1.3, 4.1.4,4.1.5,4.1.6 retorna un subconjunto reducido S, en dicho conjunto se incluyen las características así como las etiquetas de clase. En particular el algoritmo (4.1.1) asegura devolver un subconjunto reducido consistente por construcción en el espacio Euclideano

4.1.1. Fast Condensed Nearest Neighbor (FCNN)

```
def FCNN(X,y)
```

- X: Matriz de nxp tal que n es el numero de observaciones y p el numero de características
- y: Vector de etiquetas de longitud n correspondiente a cada una de las observaciones de X.

4.1.2. Kernel Fast Condensed Nearest Neighbor (KFCNN)

```
def KFCNN(X,y,gamma)
```

- X: Matriz de nxp tal que n es el numero de observaciones y p el numero de características
- y: Vector de etiquetas de longitud n correspondiente a cada una de las observaciones de X.
- gamma: parámetro gamma correspondiente al parámetro del kernel RBF descrito en (3), controla el ancho de banda de similaridad entre observaciones.

4.1.3. Kernel PCA Condensed Nearest Neighbor (KPCAFCNN)

```
def KPCAFCNN(X,y,gamma,n_componentes):
```

- X: Matriz de nxp tal que n es el numero de observaciones y p el numero de características

- y: Vector de etiquetas de longitud n correspondiente a cada una de las observaciones de X.
- gamma: parámetro gamma correspondiente al parámetro del kernel RBF descrito en (3), controla el ancho de banda de similaridad entre observaciones.
- n_componentes: parámetro asociado al numero de componentes principales al utilizar la técnica de transformación KERNEL PCA.

4.1.4. Decremental Reduction Optimization Procedures (DROP3)

```
def DROP3(X,y):
```

- X: Matriz de nxp tal que n es el numero de observaciones y p el numero de características.
- y: Vector de etiquetas de longitud n correspondiente a cada una de las observaciones de X.

4.1.5. Kernel Decremental Reduction Optimization Procedures (KDROP3)

```
def KDROP3(X,y,gamma):
```

- X: Matriz de nxp tal que n es el numero de observaciones y p el numero de características.
- y: Vector de etiquetas de longitud n correspondiente a cada una de las observaciones de X.
- gamma: parámetro gamma correspondiente al parámetro del kernel RBF descrito en (3), controla el ancho de banda de similaridad entre observaciones.

4.1.6. KernelPCA Decremental Reduction Optimization Procedures (KPCADROP3)

```
def KPCADROP3(X,y,gamma,n_componentes):
```

- X: Matriz de nxp tal que n es el numero de observaciones y p el numero de características.
- y: Vector de etiquetas de longitud n correspondiente a cada una de las observaciones de X.
- gamma: parámetro gamma correspondiente al parámetro del kernel RBF descrito en (3), controla el ancho de banda de similaridad entre observaciones.
- n_componentes: parámetro asociado al numero de componentes principales al utilizar la técnica de transformación KERNEL PCA.

Los hiperparámetros utilizados en nuestra experimentación fueron los siguientes:

Algoritmo	**Parámetro**	**Valor**
FCNN_MR	k	1
FCNN	k	1
KFCNN	k	1
	kernel	RBF
	gamma	[0.0005,0.005,0.05,0.5,2,5,10,12,20,50,100]
KPCAFCNN	k	1
	Componentes Principales	[10]
	kernel	RBF
	gamma	[0.0005,0.005,0.05,0.5,2,5,10,12,20,50,100]
DROP3	k	3
	Componentes	No se especifica
	kernel	RBF
KDROP3	k	3
	kernel	RBF
	gamma	[0.0005,0.005,0.05,0.5,2,5,10,12,20,50,100]
KPCADROP3	k	3
	Componentes Principales	[10]
	kernel	RBF
	gamma	[0.0005,0.005,0.05,0.5,2,5,10,12,20,50,100]

Tabla 4.2: Hiperparámetros utilizados en nuestra experimentación

4.2. Conjuntos de datos

Se utilizan 5 Conjuntos de datos para evaluar el rendimiento de nuestras propuestas, los datos fueron tomados de distintos repositorios como UCI *Machine Learning Repository* (Dua y Graff, 2017) y del sitio *OpenML* (Vanschoren, van Rijn, Bischl, y Torgo, 2013).

En la tabla (4.3) se presenta una descripción de cada conjunto de datos ordenados alfabéticamente, la tabla presenta el numero de observaciones (Obs), numero de atributos (Atr), numero de clases en el conjunto de datos (Num Clases) y el tamaño del archivo en MB (Tamaño MB).

Para propósitos experimentales con el objetivo de tener un mejor control del parámetro γ que corresponde al kernel RBF los datos son previamente preprocesados para tener media cero y desviación estándar unitaria, se utilizan atributos numéricos y ordinales numéricos mientras que las características ordinales no numéricas y las nominales son descartadas. Los conjuntos de datos son diversos en el número de instancias, número de de variables y clases balanceadas y no balanceadas.

Conjunto de datos	Obs	Atr	Num Clases	Tamaño MB
Airlines	129,488	17	2	50.0
BNG Australian	1,000,000	14	2	82.7
Covtype1_vs_2	497,000	54	2	61.1
Fraud Challengue	150,000	7	2	46.7
Diabetes	249,050	20	2	99.8

Tabla 4.3: Bases de datos utilizadas

BNG Australian

Corresponde a un conjunto de datos que se puede obtener del repositorio OpenML, corresponde a una base datos artificial.

Covtype1_vs_2

El conjunto de datos tiene como objetivo realizar la predicción del tipo de cubierta forestal a partir de variables cartográficas únicamente (sin datos de teledetección). El conjunto original contiene 7 categorias de cubierta forestal, por lo que al referirnos a *Covtype1_vs_2* estaremos contemplando unicamente las categorias 1 y 2.

Invistico_Airline

El conjunto de datos consta de los detalles de clientes de cierta aerolínea que han volado con ellos, se tienen datos de los clientes en varios contextos así como sus datos de vuelo. El objetivo principal de este conjunto de datos es predecir si un futuro cliente estaría satisfecho con su servicio dados los detalles de los otros valores de parámetros.

Fraud Challengue

Conjunto de datos no balanceados en el numero de observaciones por clase, el 5 % de las observaciones contiene la etiqueta fraude, el 95 % son etiquetas de transacciones legitimas. El conjunto de datos también es utilizado para estudiar la detección de eventos relativamente raros determinar la importancia de las variables y el tratamiento con datos categóricos de alta frecuencia.

Diabetes

El Sistema de Vigilancia de Factores de Riesgo del Comportamiento (BRFSS, por sus siglas en inglés) es una encuesta telefónica relacionada con la salud que los CDC recopilan anualmente, en particular trabajamos con la encuesta del 2015. La variable objetivo Diabetes_0_2 tiene 2 clases. 0 es sin diabetes o solo durante el embarazo y 2 es diabetes. Hay un desequilibrio de clases en este conjunto de datos.

4.3. Análisis de resultados datos balanceados

En esta Sección se muestran los resultados a partir de las metodologías FCNN_MR, FCNN, KFCNN, KPCAFCNN, DROP3, KDROP3 y KPCADROP3.

Presentamos las evaluaciones priorizando la métrica *f1-score-macro* sobre las demás métricas (Véase Sección A) y los resultados se desglosan para cada uno de los clasificadores.

4.3.1. K Vecinos Más Cercanos

Tabla 4.4: Desempeño en el clasificador KNN

Metodología	**Parámetros**	**Métrica F1**	$1 - \frac{\|S\|}{\|T\|}$	**Tiempo S.I.**	Tiempo Evaluación
Airlines					
FCNN_MR	NA	0.88638(0.00190)	78.43 %	1773.35	9.02
FCNN	NA	0.88993(0.00200)	69.94 %	22.88	12.88
KFCNN	$\gamma = 10$	0.90547(0.00220)	26.45 %	169.91	30.23
KPCAFCNN	$comp = 10, \gamma = 10$	0.90233(0.00196)	44.10 %	147.02	22.51
DROP3	NA	0.89642(0.00172)	56.09 %	252.49	17.81
KDROP3	$\gamma = 5$	0.89869(0.00200)	20.65 %	92.15	31.85
KPCADROP3	$comp = 10, \gamma = 0.0001$	0.89291(0.00200)	56.31 %	266.71	17.78
	T	**0.90737**(0.00152)	0 %	-	**39.72**
Covtype					
FCNN_MR	NA	0.91101(0.00200)	86.03 %	39937.84	81.05
FCNN	NA	0.93067(0.00102)	51.00 %	73.14	294.50
KFCNN	$\gamma = 500$	0.94382(0.00200)	14.51 %	177.16	591.87
KPCAFCNN	$com = 10\ \gamma = 0.01$	0.930715(0.00200)	48.72 %	132.04	308.12
DROP3	NA	0.92045(0.00200)	57.20 %	3296.64	261.45
KDROP3	$\gamma = 500$	0.93527(0.00200)	16.23 %	101.75	785.21
KPCADROP3	$comp = 10, \gamma = 10$	0.92258(0.00200)	50.29 %	2240.01	296.02
	T	**0.94792(0.00058)**	0 %	-	**712.11**
BNG-Australian					
FCNN	NA	**0.86257**(0.00053)	40.43 %	95.22	2.47
KFCNN	$\gamma = 5$	0.86281(0.00069)	66.91 %	164.35	4.42
KPCAFCNN	$ncom = 14\ \gamma = 10$	0.86243(0.00066)	58.35 %	280.90	3.65
DROP3	NA				
KDROP3	NA				
KPCADROP3	NA				
	T	**0.85796**(0.00193)	-	-	6.5

4.3.2. Bosques Aleatorios

Tabla 4.5: Desempeño en el clasificador BOSQUE ALEATORIO

Metodología	**Parámetros**	**Métrica F1**	$1 - \frac{\|S\|}{\|T\|}$	**Tiempo S.I.**	Tiempo Evaluación
Airlines					
FCNN_MR	NA	0.88137(0.00200)	78.43 %	1773.35	0.30
FCNN	NA	0.88397(0.00244)	69.94 %	22.88	0.31
KFCNN	$\gamma = 100$	0.89009(0.00200)	10.88 %	187.22	0.5
KPCAFCNN	$comp = 10, \gamma = 5$	0.88853 (0.00196)	45.54 %	186.01	0.37
DROP3	NA	0.88872(0.00172)	56.09 %	252.49	0.34
KDROP3	$\gamma = 0.01$	0.88686(0.00200)	56.31 %	302.44	0.35
KPCADROP3	$comp = 10, \gamma = 0.1$	0.88566(0.00200)	63.75 %	288.29	0.31
	T	**0.89060**(0.00200)	0 %	-	**0.56**
Covtype					
FCNN_MR	NA	0.76216(0.00200)	86.03 %	39937.84	0.48
FCNN	NA	0.76672(0.00102)	51.00 %	73.14	1.36
KFCNN	$\gamma = 0.01$	0.76814(0.00200)	50.99 %	73.14	1.36
KPCAFCNN	$com = 10\ \gamma = 0.005$	0.769704(0.00200)	49.15 %	132.04	1.39
DROP3	NA	0.76319(0.00200)	57.20 %	3296.64	1.19
KDROP3	$\gamma = 0.5$	0.76734(0.00200)	48.25 %	101.75	1.44
KPCADROP3	$comp = 10, \gamma = 0.0005$	0.76734(0.00200)	60.94 %	3311.74	1.07
	T	**0.76175(0.00058)**	0 %	-	**2.88**
BNG-Australian					
FCNN_MR	NA	()	%		
FCNN	NA	( )	%		
KFCNN	$\gamma = 10$	()	%		
KPCAFCNN	$comp = 10, \gamma = 10$	()	%		
DROP3	NA	()	%		
KDROP3	$\gamma = 100$	( )	%		
KPCAFCNN	$comp = 10, \gamma = 0.0001$	( )	%		
	T	()	%	-	

4.3.3. Regresión Logística

Tabla 4.6: Desempeño en el clasificador REGRESIÓN LOGÍSTICA

Metodología	**Parámetros**	**Métrica F1**	$1 - \frac{\|S\|}{\|T\|}$	**Tiempo S.I.**	Tiempo Evaluación
Airlines					
FCNN_MR	NA	0.79741(0.00340)	78.43 %	1773.35	0.23
FCNN	NA	0.78641(0.00356)	69.94 %	22.88	0.25
KFCNN	$\gamma = 100$	0.81111(0.00448)	10.88 %	187.22	0.60
KPCAFCNN	$comp = 10, \gamma = 10$	0.80765 (0.00196)	44.10 %	147.77	0.41
DROP3	NA	0.80998(0.00172)	56.09 %	252.49	0.30
KDROP3	$\gamma = 0.01$	0.81037(0.00200)	15.25 %	42.09	0.60
KPCADROP3	$comp = 10, \gamma = 0.01$	0.81066(0.00200)	58.86 %	277.131	0.3
	T	**0.80550**(0.00152)	0 %	-	**0.66**
Covtype					
FCNN_MR	NA	0.76584(0.00200)	86.03 %	39937.84	2.88
FCNN	NA	0.76552(0.00200)	51.00 %	73.14	5.86
KFCNN	$\gamma = 10$	0.77030(0.00200)	32.68 %	142.55	10.42
KPCAFCNN	$com = 10, \gamma = 0.005$	0.77016(0.00200)	40.23 %	178.13	10.98
DROP3	NA	0.77115(0.00200)	57.20 %	3296.64	10.13
KDROP3	$\gamma = 0.05$	0.77124(0.00200)	58.32 %	3739.39	9.86
KPCADROP3	$comp = 10, \gamma = 1$	0.76931(0.00200)	61.85 %	2824.46	7.66
	T	**0.76934(0.00058)**	0 %	-	**21.32**
BNG-Australian					
FCNN_MR	NA	()	%		
FCNN	NA	()	%		
KFCNN	$\gamma = 10$	()	%		
KPCAFCNN	$comp = 10, \gamma = 10$	()	%		
DROP3	NA	()	%		
KDROP3	$\gamma = 100$	()	%		
KPCAFCNN	$comp = 10, \gamma = 0.0001$	()	%		
	T	()	%	-	

4.3.4. Red Neuronal Múltiples Capas

Tabla 4.7: Desempeño en el clasificador MLP

Metodología	**Parámetros**	**Métrica F1**	$1 - \frac{\|S\|}{\|T\|}$	**Tiempo S.I.**	Tiempo Evaluación
Airlines					
FCNN_MR	NA	0.92897(0.00162)	78.43 %	1773.35	12.27
FCNN	NA	0.93087(0.00266)	69.94 %	22.88	16.93
KFCNN	$\gamma = 100$	0.938863(0.00399)	10.88 %	187.22	48.45
KPCAFCNN	$comp = 10, \gamma = 5$	0.93528(0.00200)	45.54 %	186.01	30.59
DROP3	NA	0.92963(0.00172)	56.09 %	252.49	24.83
KDROP3	$\gamma = 10$	0.93412(0.00200)	15.25 %	34.07	43.63
KPCADROP3	$comp = 10, \gamma = 0.0001$	0.92941(0.00200)	56.31 %	266.71	24.57
	T	**0.93906**(0.00200)	0 %	-	**50.67**
Covtype					
FCNN_MR	NA	0.85590(0.00200)	86.03 %	39937.84	38.65
FCNN	NA	0.88871(0.00200)	51.00 %	73.14	127.07
KFCNN	$\gamma = 500$	0.90009(0.00200)	14.51 %	142.55	232.44
KPCAFCNN	$com = 10\ \gamma = 0.005$	0.89333(0.00200)	40.80 %	185.14	158.03
DROP3	NA	0.88479(0.00200)	57.20 %	3296.64	111.87
KDROP3	$\gamma = 50$	0.895246(0.00200)	17.14 %	184.30	227.27
KPCADROP3	$comp = 10, \gamma = 10$	0.88653(0.00200)	50.29 %	2240.74	138.23
	T	**0.89891(0.00200)**	0 %	-	**274.92**
BNG-Australian					
FCNN_MR	NA	()	%		
FCNN	NA	( )	%		
KFCNN	$\gamma = 10$	()	%		
KPCAFCNN	$comp = 10, \gamma = 10$	()	%		
DROP3	NA	()	%		
KDROP3	$\gamma = 100$	( )	%		
KPCAFCNN	$comp = 10, \gamma = 0.0001$	( )	%		
	T	()	%	-	

4.3.5. Maquina de Soporte Vectorial con kernel RBF

Tabla 4.8: Desempeño en el clasificador SVM(Kernel = 'RBF')

Metodología	**Parámetros**	**Métrica F1**	$1 - \frac{\|S\|}{\|T\|}$	**Tiempo S.I.**	Tiempo Evaluación
Airlines					
FCNN_MR	NA	0.93011(0.00240)	78.43 %	1773.35	28.29
FCNN	NA	0.93171(0.00127)	69.94 %	22.88	41.12
KFCNN	$\gamma = 10$	0.93419(0.00399)	26.45 %	169.91	110.34
KPCAFCNN	$comp = 10, \gamma = 0.05$	0.93217(0.00200)	65.20 %	88.40	47.28
DROP3	NA	0.921032(0.00172)	56.09 %	252.49	30.39
KDROP3	$\gamma = 100$	0.92514(0.00200)	12.24 %	25.11	81.19
KPCADROP3	$comp = 10, \gamma = 0.0001$	0.92012(0.00200)	56.31 %	266.71	26.74
	T	**0.93478**(0.00200)	0 %	-	**152.86**
Covtype					
FCNN_MR	NA	0.87814(0.00200)	86.03 %	39937.84	29.98
FCNN	NA	0.91909(0.00200)	51.00 %	73.14	222.90
KFCNN	$\gamma = 500$	0.92374(0.00200)	14.51 %	177.16	528.71
KPCAFCNN	$com = 10\ \gamma = 5$	0.92097(0.00200)	40.80 %	185.14	281.9
DROP3	NA	0.89938(0.00200)	57.20 %	3296.64	123.56
KDROP3	$\gamma = 50$	0.905454(0.00200)	17.14 %	184.30	403.98
KPCADROP3	$comp = 10, \gamma = 10$	0.90057(0.00200)	50.29 %	2240.74	158.24
	T	**0.92573(0.00200)**	0 %	-	**713.06**
BNG-Australian					
FCNN_MR	NA	()	%		
FCNN	NA	( )	%		
KFCNN	$\gamma = 10$	()	%		
KPCAFCNN	$comp = 10, \gamma = 10$	()	%		
DROP3	NA	()	%		
KDROP3	$\gamma = 100$	( )	%		
KPCAFCNN	$comp = 10, \gamma = 0.0001$	( )	%		
	T	()	%	-	

4.3.6. Maquina de Soporte Vectorial con kernel Lineal

Tabla 4.9: Desempeño en el clasificador SVM

Metodología	**Parámetros**	**Métrica F1**	$1 - \frac{\|S\|}{\|T\|}$	**Tiempo S.I.**	Tiempo Evaluación
Airlines					
FCNN_MR	NA	0.79904(0.00374)	78.43 %	1773.35	33.51
FCNN	NA	0.79766(0.00416)	69.94 %	22.88	67.28
KFCNN	$\gamma = 100$	0.81789(0.00465)	10.88 %	187.22	476.63
KPCAFCNN	$comp = 10, \gamma = 10$	0.81537(0.00200)	44.10 %	147.77	194.10
DROP3	NA	0.81564(0.00172)	56.09 %	252.49	83.60
KDROP3	$\gamma = 100$	0.81582(0.00200)	55.59 %	297.733	85.77
KPCADROP3	$comp = 10, \gamma = 0.001$	0.81619(0.00200)	56.56 %	266.71	59.89
	T	0.81487(0.00200)	0 %	-	625.71
Covtype					
FCNN_MR	NA	0.75503(0.00200)	86.03 %	39937.84	16.25
FCNN	NA	0.76927(0.00200)	51 %	73.14	250.08
KFCNN	$\gamma = 50$	0.77055(0.00200)	16.55 %	178.03	994.95
KPCAFCNN	$com = 10\ \gamma = 5$	0.77222(0.00200)	41.69 %	155.64	400.91
DROP3	NA	0.77387(0.00200)	57.20 %	3296.64	171.51
KDROP3	$\gamma = 0.05$	0.77370(0.00200)	58.32 %	3739.39	147.05
KPCADROP3	$comp = 10, \gamma = 1$	0.77174(0.00200)	61.85 %	2824.46	120.94
	T	**0.77232(0.00200)**	0 %	-	**1470.67**
BNG-Australian					
FCNN_MR	NA	()	%		
FCNN	NA	( )	%		
KFCNN	$\gamma = 10$	()	%		
KPCAFCNN	$comp = 10, \gamma = 10$	()	%		
DROP3	NA	()	%		
KDROP3	$\gamma = 100$	( )	%		
KPCAFCNN	$comp = 10, \gamma = 0.0001$	( )	%		
	T	()	%	-	

4.3.7. Desempeño general por algoritmo

- **FCNN_MR :** La mayor compresión y reducción en el tiempo de evaluación se obtiene utilizando el algoritmo $FCNN\_MR$, y la pérdida de rendimiento no supera las 5 centésimas de la línea base en nuestra métrica de interés.
- **FCNN :** La metodología $FCNN$ muestra resultados ligeramente mejores a $FCNN\_MR$ en la métrica de interés, no obtiene la mayor compresión pero se obtiene una mejora significativa en el tiempo Selección de Instancias respecto al tiempo empleado por $FCNN\_MR$.
- **KFCNN :** $KFCNN$ muestra el mejor rendimiento, la velocidad de ejecución supera a $FCNN\_MR$ pero no logra una gran aceleración y la compresión mínima es de alrededor 10 %, si se utiliza .
- **KPCAFCNN :** es una alternativa que no destaca del todo en datos balanceados, puede ser considerado como un punto intermedio entre compresión y el valor de la métrica, se agrega una complejidad computacional al realizar Kernel-PCA lo que impacta directamente en el tiempo de selección de instancias.
- **DROP3 :** Al compararlo con FCNN, muestra un rendimiento similar en la compresión y en los valores máximos de la métrica F1, el tiempo de ejecución para llevar a cabo la selección de instancias es mayor.
- **KDROP3 :** Cuando se utiliza un parámetro alto en el kernel los tiempos de ejecución en la selección de instancias son menores y en consecuencia se obtiene una menor compresión, por el contrario cuando el parámetro del kernel es pequeño la compresión es mayor por lo que el proceso de selección de instancias será más largo.
- **KPCADROP3 :** Igual que en KPCAFCNN, solo busca ser una alternativa al mantener un equilibro entre compresión y la métrica de interés.

4.3.8. Desempeño general por clasificador

Particularmente en la metodología FCNN y KFCNN (Usando un parámetro γ pequeño), los clasificadores que se ven beneficiados por una ganancia en el tiempo de ejecución total que comprende el tiempo de realizar la Selección de Instancias más el tiempo de evaluación comparado con el tiempo de evaluación del conjunto original son: k-vecinos más cercanos, Red Neuronal y las maquinas de soporte vectorial independientemente del kernel que se quiera usar.

Los clasificadores como los bosques aleatorios y la regresión logística no logran una ganancia en el tiempo total y la ganancia se vería únicamente reflejada en el tiempo de evaluación de nuevas

observaciones además del espacio en disco al lograr una compresión mayor al 50 %.

En las metodologías donde se utiliza una distancia en el espacio kernel, a medida que γ tiene un valor más grande, en general se mantienen un mayor numero de instancias en el conjunto final, clasificadores como K vecinos más cercanos y redes neuronales claramente se ven beneficiados por ello pero no es caso del clasificador Bosques Aleatorios, quien obtiene sus valores más altos en las métricas con valores de kernel pequeños.

La Figura (4.1) presenta las aceleraciones que se pueden lograr y que dependen en gran medida de la compresión que se pueda lograr en la información al utilizar los algoritmos de selección de instancias.

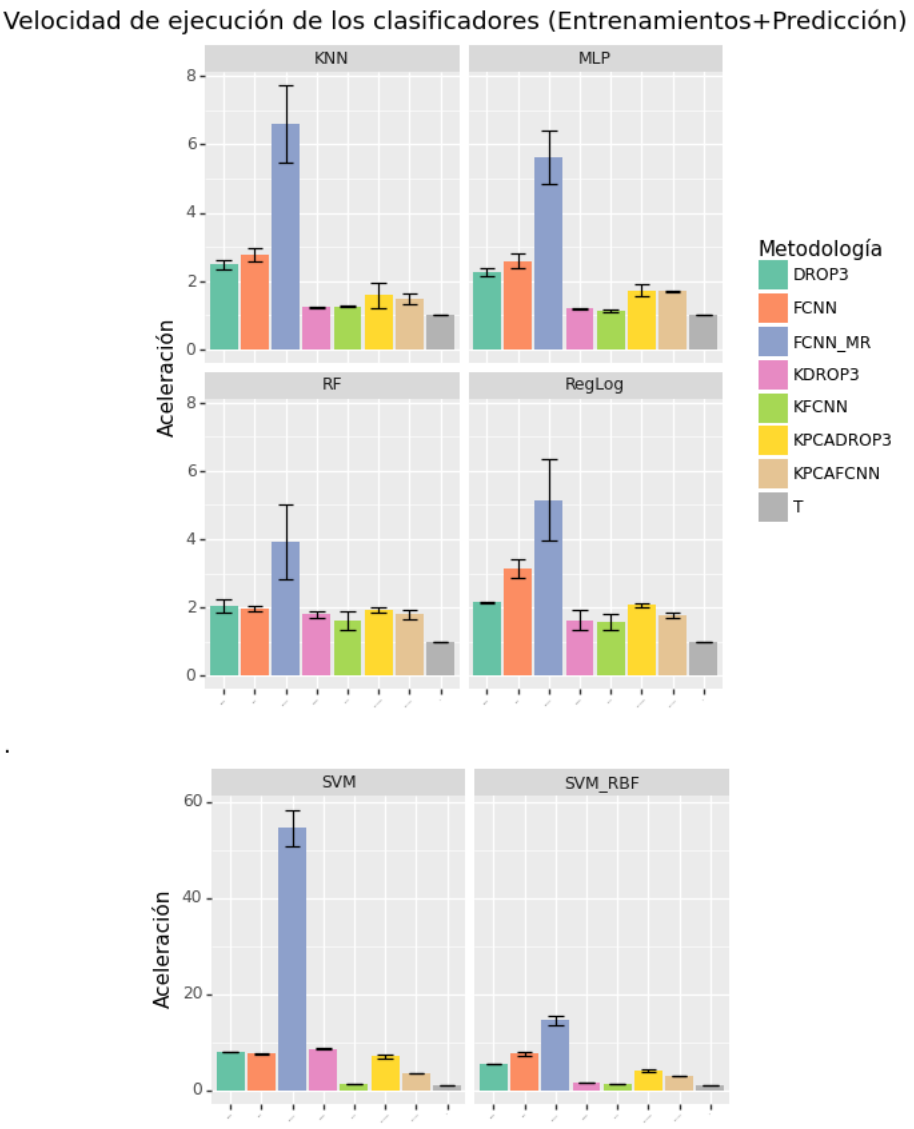

Figura 4.1: Aceleración de los algoritmos en el tiempo de evaluación

4.4. Análisis de resultados datos no balanceados

En esta Sección se muestran los resultados de las metodologías FCNN_MR, FCNN, KFCNN, KPCAFCNN adaptadas especialmente a datos no balanceados.

De igual manera que en la Sección Presentamos las evaluaciones priorizando la métrica f1-score-macro sobre las demás métricas (Véase Sección A) y los resultados se desglosan para cada uno de los clasificadores.

4.4.1. K Vecinos Más Cercanos

Tabla 4.10: Desempeño en el clasificador KNN datos asimetricos

Metodología	**Parámetros**	**Métrica F1**	$1 - \frac{\|S\|}{\|T\|}$	**Tiempo S.I.**	Tiempo Evaluación
FraudChallengue					
FCNN_MR	NA	**0.71496(0.00200)**	**83.00 %**	1366.86	**3.45**
FCNN	NA	0.68199(0.00200)	82.05 %	**15.72**	4.39
KFCNN	$\gamma = 0.0001$	0.68199(0.00200)	82.05 %	**15.72**	4.39
KPCAFCNN	$comp = 10, \gamma = 0.05$	0.68564(0.00200)	80.94 %	68.98	4.40
	T	0.72698(0.00200)	0 %	-	**5.96**
Diabetes					
FCNN_MR	NA	0.6083	**63.65 %**	5282.45	62.74
FCNN	NA	0.59980(0.00200)	60.43 %	23.47	63.59
KFCNN	$\gamma = 0.5$	0.59988(0.00200)	60.44 % %	**17.67**	63.62
KPCAFCNN	$com = 10\ \gamma = 10$	**0.61385(0.00200)**	59.71 %	52.77	**60.56**
	T	0.60436(0.00200)	0 %	-	**151.75**
CardTransaction					
FCNN_MR	NA	0.99427(0.00200)	**90.91 %**	6251.82	14.52
FCNN	NA	0.99564(0.00200)	85.20 %	45.05	18.87
KFCNN	$\gamma = 0.1$	0.99565(0.00200)	85.20 %	58.09	24.59
KPCAFCNN	$com = 10\ \gamma = 500$	0.99572(0.00200)	77.68 %	176.74	15.76
	T	**0.99631(0.00200)**	0 %	-	**19.28**
CreditRisk					
FCNN_MR	NA	()	%		
FCNN	NA	0.91843(0.00200)	85.81 %	48.86	236.73
KFCNN	$\gamma = 0.1$	0.91851(0.00200)	85.80 %	55.06	238.33
KPCAFCNN	$com = 10\ \gamma = 0.5$	0.92628(0.00200)	79.83 %	156.17	345.09
	T	**0.93099(0.00200)**	0 %	-	2133.86

4.4.2. Bosques Aleatorios

Tabla 4.11: Desempeño en el clasificador RANDOM FOREST datos asimétricos

Metodología	**Parámetros**	**Métrica F1**	$1 - \frac{\|S\|}{\|T\|}$	**Tiempo S.I.**	Tiempo Evaluación
FraudChallengue					
FCNN_MR	NA	0.75245(0.00200)	**83.00 %**	1366.86	0.72
FCNN	NA	0.75634(0.00200)	82.05 %	**15.72**	0.72
KFCNN	$\gamma = 5$	0.75789(0.00200)	82.65 %	15.77	0.72
KPCAFCNN	$comp = 10, \gamma = 1$	**0.76578(0.00200)**	80.62 %	69.27	0.72
	T	0.65220(0.00200)	0 %	-	**1.43**
Diabetes					
FCNN_MR	NA	0.60015(0.00200)	63.65 %	5282.45	0.41
FCNN	NA	0.60230(0.00200)	60.43 %	23.47	0.42
KFCNN	$\gamma = 1$	0.61251(0.00200)	61.35 % %	**17.60**	0.42
KPCAFCNN	$com = 10\ \gamma = 10$	**0.68036(0.00200)**	**63.95 %**	50.67	**0.40**
	T	0.48462(0.00200)	0 %	-	**0.9268**
CardTransaction					
FCNN_MR	NA	0.41777(0.00200)	**90.91 %**	6251.82	0.65
FCNN	NA	0.99544(0.00200)	85.20 %	45.05	0.86
KFCNN	$\gamma = 0.05$	0.99717(0.00200)	85.20 %	57.91	0.86
KPCAFCNN	$com = 10\ \gamma = 500$	**0.99878(0.00200)**	77.68 %	176.74	1.22
	T	0.99858(0.00200)	0 %	-	**5.73**
CreditRisk					
FCNN_MR	NA	()	%		
FCNN	NA	0.97680(0.00200)	85.81 %	48.86	1.16
KFCNN	$\gamma = 0.05$	**0.98080(0.00200)**	85.80 %	55.06	1.11
KPCAFCNN	$com = 10\ \gamma = 0.5$	0.97895(0.00200)	79.83 %	156.17	1.37
	T	0.91605(0.00200)	0 %	-	9.7699

4.4.3. Regresión Logística

Tabla 4.12: Desempeño en el clasificador REGRESIÓN LOGISTICA datos asimetricos

Metodología	**Parámetros**	**Métrica F1**	$1 - \frac{\|S\|}{\|T\|}$	**Tiempo S.I.**	Tiempo Evaluación
FraudChallengue					
FCNN_MR	NA	0.76357(0.00200)	**83.00 %**	1366.86	0.28
FCNN	NA	0.76581(0.00200)	82.05 %	**15.72**	**0.23**
KFCNN	$\gamma = 5$	0.76672(0.00200)	82.65 %	15.77	0.24
KPCAFCNN	$comp = 10, \gamma = 1$	**0.76699(0.00200)**	80.62 %	69.27	0.26
	T	0.72958(0.00200)	0 %	-	**0.71**
Diabetes					
FCNN_MR	NA	0.65145(0.00200)	**63.65 %**	5282.45	0.55
FCNN	NA	0.65557(0.00200)	60.43 %	23.47	0.5407
KFCNN	$\gamma = 1$	0.66095(0.00200)	61.95 %	**17.60**	**0.46**
KPCAFCNN	$com = 10\ \gamma = 10$	**0.67906(0.00200)**	59.71 %	52.77	0.55
	T	0.59476(0.00200)	0 %	-	**1.13**
CardTransaction					
FCNN_MR	NA	0.14519(0.00200)	**90.91 %**	6251.82	0.79
FCNN	NA	0.58710(0.00200)	85.20 %	45.05	1.00
KFCNN	$\gamma = 10$	**0.84947(0.00200)**	88.40 %	56.91	0.96
KPCAFCNN	$com = 10\ \gamma = 500$	0.83058(0.00200)	77.68 %	176.74	1.00
	T	0.84761(0.00200)	0 %	-	**2.92**
CreditRisk					
FCNN_MR	NA	()	**%**		
FCNN	NA	0.98768(0.00200)	85.81 %	48.86	7.79
KFCNN	$\gamma = 0.05$	0.98777(0.00200)	85.80 %	55.06	8.87
KPCAFCNN	$com = 10\ \gamma = 1$	**0.98884(0.00200)**	83.56 %	145.84	11.60
	T	0.98793(0.00200)	0 %	-	138.07

4.4.4. Red Neuronal Múltiples Capas

Tabla 4.13: Desempeño en el clasificador Perceptrón de Capa Múltiple datos asimétricos

Metodología	**Parámetros**	**Métrica F1**	$1 - \frac{\|S\|}{\|T\|}$	**Tiempo S.I.**	Tiempo Evaluación
FraudChallengue					
FCNN_MR	NA	0.78282(0.00200)	**83.00 %**	1366.86	**15.93**
FCNN	NA	0.78277(0.00200)	82.05 %	**15.72**	17.80
KFCNN	$\gamma = 0.0001$	0.78277(0.00200)	82.05 %	**15.72**	17.83
KPCAFCNN	$comp = 10, \gamma = 1$	**0.78347(0.00200)**	81.95 %	69.02	20.33
	T	0.76954(0.00200)	0 %	-	**34.53**
Diabetes					
FCNN_MR	NA	0.66144(0.00200)	**63.65 %**	5282.45	**36.02**
FCNN	NA	0.66686(0.00200)	60.43 %	23.47	38.07
KFCNN	$\gamma = 0.5$	0.66617(0.00200)	60.44 %	**17.67**	33.27
KPCAFCNN	$com = 10\ \gamma = 10$	**0.67622(0.00200)**	59.71 %	52.77	38.85
	T	0.60957(0.00200)	0 %	-	**45.12**
CardTransaction					
FCNN_MR	NA	0.99305(0.00200)	**90.91 %**	6251.82	37.02
FCNN	NA	**0.99786(0.00200)**	85.20 %	45.05	65.59
KFCNN	$\gamma = 0.0001$	**0.99786(0.00200)**	85.20 %	58.30	66.09
KPCAFCNN	$com = 10\ \gamma = 0.0001$	0.99768(0.00200)	85.20 %	154.51	67.78
	T	0.99657(0.00200)	0 %	-	**19.28**
CreditRisk					
FCNN_MR	NA	()	%		
FCNN	NA	0.98932(0.00200)	85.81 %	48.86	42.88
KFCNN	$\gamma = 0.0001$	0.98932(0.00200)	85.81 %	54.58	42.52
KPCAFCNN	$com = 10\ \gamma = 1$	0.99017(0.00200)	83.56 %	145.84	56.34
	T	**0.99275(0.00200)**	0 %	-	114.81

4.4.5. Maquina de Soporte Vectorial con kernel RBF

Tabla 4.14: Desempeño en el clasificador SVM RBF datos asimétricos

Metodología	**Parámetros**	**Métrica F1**	$1 - \frac{\|S\|}{\|T\|}$	**Tiempo S.I.**	Tiempo Evaluación
FraudChallengue					
FCNN_MR	NA	0.76289(0.00200)	**83.00 %**	1366.86	7.18
FCNN	NA	0.77414(0.00200)	82.05 %	**15.72**	6.88
KFCNN	$\gamma = 5$	0.77561(0.00200)	82.65 %	15.77	6.59
KPCAFCNN	$comp = 10, \gamma = 1$	**0.78077(0.00200)**	80.62 %	69.27	**5.94**
	T	0.74743(0.00200)	0 %	-	**52.22**
Diabetes					
FCNN_MR	NA	0.62630(0.00200)	63.65 %	5282.45	50.34
FCNN	NA	0.64840(0.00200)	60.43 %	23.47	52.61
KFCNN	$\gamma = 1$	0.649703(0.00200)	61.35 %	**17.67**	49.91
KPCAFCNN	$com = 10\ \gamma = 1$	**0.68615(0.00200)**	**63.95 %**	52.77	**35.73**
	T	0.55169(0.00058)	0 %	-	329.32
CardTransaction					
FCNN_MR	NA	0.79853(0.00200)	**90.91 %**	6251.82	6.97
FCNN	NA	0.99632(0.00200)	85.20 %	45.05	25.16
KFCNN	$\gamma = 0.001$	0.99633(0.00200)	85.20 %	57.84	25.53
KPCAFCNN	$com = 10\ \gamma = 0.1$	0.99654(0.00200)	85.23 %	153.53	25.02
	T	**0.99715(0.00200)**	0 %	-	177.31
CreditRisk					
FCNN_MR	NA	()	%		
FCNN	NA	0.98288(0.00200)	85.81 %	48.86	33.92
KFCNN	$\gamma = 0.05$	0.98312(0.00200)	85.81 %	54.58	33.91
KPCAFCNN	$com = 10\ \gamma = 1$	0.98490(0.00200)	83.56 %	145.84	43.55
	T	**0.98561(0.00200)**	0 %	-	455.17

4.4.6. Maquina de Soporte Vectorial con kernel Lineal

Tabla 4.15: Desempeño en el clasificador SVM LINEAR datos asimetricos

Metodología	**Parámetros**	**Métrica F1**	$1 - \frac{\|S\|}{\|T\|}$	**Tiempo S.I.**	Tiempo Evaluación
FraudChallengue					
FCNN_MR	NA	0.75808(0.00200)	**83.00 %**	1366.86	**2.70**
FCNN	NA	0.76296(0.00200)	82.05 %	**15.72**	3.05
KFCNN	$\gamma = 5$	0.76544(0.00200)	82.65 %	15.77	2.97
KPCAFCNN	$comp = 10, \gamma = 1$	**0.7677(0.00200)**	80.62 %	69.27	3.10
	T	0.64434(0.00200)	0 %	-	**52.22**
Diabetes					
FCNN_MR	NA	0.52378(0.00200)	**63.65 %**	5282.45	46.15
FCNN	NA	0.58934(0.00200)	60.43 %	23.47	**16.56**
KFCNN	$\gamma = 1$	0.63316(0.00200)	61.35 %	**17.60**	37.14
KPCAFCNN	$com = 10\ \gamma = 10$	**0.67974(0.00200)**	59.71 %	52.77	50.52
	T	0.46174(0.00200)	0 %	-	23.35?
CardTransaction					
FCNN_MR	NA	0.18720(0.00200)	**90.91 %**	6251.82	1.40
FCNN	NA	0.43115(0.00200)	85.20 %	45.05	38.81
KFCNN	$\gamma = 0.1$	0.84325(0.00200)	85.20 %	56.91	20
KPCAFCNN	$com = 10\ \gamma = 500$	0.85309(0.00200)	77.68 %	176.74	115.61
	T	**0.86055(0.00200)**	0 %	-	**647.01**
CreditRisk					
FCNN_MR	NA	()	%		
FCNN	NA	0.99058(0.00200)	85.81 %	48.86	4.39
KFCNN	$\gamma = 0.0005$	0.99059(0.00200)	85.81 %	55.20	5.07
KPCAFCNN	$com = 10\ \gamma = 1$	**0.99080(0.00200)**	83.56 %	145.84	6
	T	0.99037(0.00200)	0 %	-	17.61

4.4.7. Desempeño general

Si bien bajo esta propuesta no es fácil encontrar la metodología de selección de instancias cuyo tiempo de ejecución sumado al de entrenamiento y predicción sea menor que el utilizando todo el

conjunto T, notamos que la métrica de interés, por ejemplo f1-score-macro logra una mejora de hasta 20 puntos porcentuales, y en general se tiene una mejora en todos los algoritmos, exceptuando el clasificador KNN.

Los algoritmos que se ven más beneficiados de realizar una selección de instancias cuando se tiene clases no equilibradas son: Las Maquinas de soporte Vectorial con los dos tipos de kernel lineal además de los Bosques Aleatorios.

Capítulo 5

Conclusiones y trabajo futuro

Utilizando de base los algoritmos de ultima generación en selección de instancias para grandes conjuntos de datos, se ha realizado una adaptación a los mismos introduciendo el uso de distancias en el espacio kernel, con ello se logra tener un mejor control del numero de instancias seleccionadas y un mejor control de la métrica de interés.

Hemos mostrado de manera experimental que los tiempos de ejecución en las técnicas de reducción de datos son prácticos para abordar la tarea de clasificación en grandes volúmenes de datos, los tiempos de ejecución de FCNN y KFCNN combinados con clasificadores en donde la fase de entrenamiento o predicción pueden llegar a ser costosos computacionalmente logran que el tiempo total de ejecución se vea reducido.

Si bien no hay una mejora significativa en cuanto al tiempo de ejecución de los clasificadores basados en Bosques Aleatorios o aquellos que utilizan la Regresión Logística, es fácil encontrar un parámetro para el cual la pérdida de rendimiento en la clasificación sea mínimo comparado con el conjunto de clasificación completo T. En los experimentos realizados se logran compresiones de mas del 50 %, por lo que la principal ventaja de reducir el numero de instancias en estos casos se ve reflejada en el espacio de almacenamiento.

Se realizó una extensa experimentación ilustrando que la selección de instancias en grandes conjuntos de datos es factible para ser usada con otros clasificadores diferentes al clasificador K Vecinos mas cercanos.

Por ultimo, presentamos el uso de estas técnicas de selección de instancias cuando se tienen clases de datos desequilibradas con el objetivo de lograr una en el rendimiento de los clasificadores.

Las lineas de trabajo futuro que se plantean a partir del trabajo desarrollado se listan a continuación.

- En el presente trabajo de investigación se consideró un numero de instancias por *mapper* igual a 2000, por lo que un camino interesante a explorar consiste en realizar la experimentación con un numero mayor de instancias por *mapper* y responder la pregunta: ¿A partir de cuantas instancias por *mapper* el algoritmo FCNN o FCNN deja de ser viable en el tiempo de ejecución total?, esto para los clasificadores donde aseguramos una ganancia en el tiempo total de ejecución del algoritmo.
- Seguir la linea de estudio de las metodologías de Selección de Instancias para tratar las situaciones cuando se tienen clases desequilibradas en conjuntos de datos, y comparar el rendimiento con técnicas de submuestro que son actualmente utilizadas para tratar este tipo de situaciones, además de extender naturalmente la propuesta a cuando se tienen más de dos clases.
- En la experimentación realizada es fácil ver el algoritmo FCNN_MR es el algoritmo con el mayor tiempo de ejecución para llevar a cabo la Selección de Instancias, por lo que una pregunta a responder sería, ¿Cual es el tiempo de ejecución mínimo que puedo alcanzar y cuantos nodos se necesitan para llegar a dichos tiempos?

Referencias

Angiulli, F. (2005). Fast condensed nearest neighbor rule. En *Proceedings of the 22nd international conference on machine learning* (pp. 25–32).

Angiulli, F., y Folino, G. (2007). Distributed nearest neighbor-based condensation of very large data sets. *IEEE Transactions on Knowledge and Data Engineering*, *19*(12), 1593–1606.

Arnaiz-González, A., González-Rogel, A., Díez-Pastor, J.-F., y Nozal, C. (2017, 02). Mr-dis: democratic instance selection for big data by mapreduce. *Progress in Artificial Intelligence*, *6*, 10. doi: 10.1007/s13748-017-0117-5

Bäck, T., Fogel, D. B., y Michalewicz, Z. (1997). Handbook of evolutionary computation. *Release*, *97*(1), B1.

Beakta, R. (2015, 01). Big data and hadoop: A review paper. *international journal of computer science & information te*, *2*.

Bhatia, N., y Vandana. (2010). Survey of nearest neighbor techniques. *CoRR*, *abs/1007.0085*. Descargado de http://arxiv.org/abs/1007.0085

Bianchini, M., y Scarselli, F. (2014). On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE Transactions on Neural Networks and Learning Systems*, *25*(8), 1553-1565. doi: 10.1109/TNNLS.2013.2293637

Blachnik, M. (2019, 03). Ensembles of instance selection methods: A comparative study. *International Journal of Applied Mathematics and Computer Science*, *29*, 151-168. doi: 10.2478/amcs-2019-0012

Cano, J., Herrera, F., y Lozano, M. (2004, 01). Using evolutionary algorithms as instance selection for data reduction in kdd: An experimental study. *Evo-*

lutionary Computation, IEEE Transactions on, *7*, 561 - 575. doi: 10.1109/TEVC.2003.819265

Chung, M. K. (2020). *Introduction to logistic regression.* arXiv. Descargado de https://arxiv.org/abs/2008.13567 doi: 10.48550/ARXIV.2008.13567

Cover, T., y Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, *13*(1), 21–27.

Curry, E. (2016). The big data value chain: Definitions, concepts, and theoretical approaches. En J. M. Cavanillas, E. Curry, y W. Wahlster (Eds.), *New horizons for a data-driven economy: A roadmap for usage and exploitation of big data in europe* (pp. 29–37). Cham: Springer International Publishing. Descargado de https://doi.org/10.1007/978-3-319-21569-3_3 doi: 10.1007/978-3-319-21569-3_3

Damji, J. S., Wenig, B., Das, T., y Lee, D. (2020). *Learning spark.* O'Reilly Media.

Daumé III, H. (2004). From zero to reproducing kernel hilbert spaces in twelve pages or less. *Online: http://pub. hal3. name/daume04rkhs. ps*.

Dean, J., y Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. *USENIX Association, OSDI'04*.

Dua, D., y Graff, C. (2017). *UCI machine learning repository.* Descargado de http://archive.ics.uci.edu/ml

Feng, W. (2019). *Learning apache spark with python.*

Feng, Y., Zhou, M., y Tong, X. (2020). *Imbalanced classification: a paradigm-based review.* arXiv. Descargado de https://arxiv.org/abs/2002.04592 doi: 10.48550/ARXIV.2002.04592

Forsbach, L. E. M. (2005). *Una técnica robusta para kernel pca* (Tesis Doctoral no publicada). Centro de Investigación en Matemáticas.

García, S., Derrac, J., Cano, J., y Herrera, F. (2012). Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *34*(3), 417-435. doi: 10.1109/TPAMI.2011.142

García, S., Luengo, J., y Herrera, F. (2015, 12). Tutorial on practical tips of the

most influential data preprocessing algorithms in data mining. *Knowledge-Based Systems*, *98*. doi: 10.1016/j.knosys.2015.12.006

García-Osorio, C., de Haro Garcia, A., y García-Pedrajas, N. (2010, 04). Democratic instance selection: A linear complexity instance selection algorithm based on classifier ensemble concepts. *Artif. Intell.*, *174*, 410-441. doi: 10.1016/j.artint .2010.01.001

González, Á. A. (2018). *Estudio de métodos de selección de instancias.* Ph. D. dissertation, Dept. Computer Science, Universidad De Burgos, Burgos ....

Grandini, M., Bagli, E., y Visani, G. (2020). *Metrics for multi-class classification: an overview.*

Hang, H., Liu, X., y Steinwart, I. (2019). *Best-scored random forest classification.*

Hart, P. (1968). The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory*, *14*(3), 515–516.

Hassanat, A. B., Abbadi, M. A., Altarawneh, G. A., y Alhasanat, A. A. (2014). Solving the problem of the k parameter in the knn classifier using an ensemble learning approach. *arXiv preprint arXiv:1409.0919*.

Im, J.-F., Villegas, F., y McGuffln, M. (2013, 10). Visreduce: Fast and responsive incremental information visualization of large datasets. En (p. 25-32). doi: 10.1109/BigData.2013.6691710

Louppe, G. (2015). *Understanding random forests: From theory to practice.*

Markel, J., y Bayless, A. J. (2020). *Using random forest machine learning algorithms in binary supernovae classification.*

Mehlig, B. (2021). *Machine learning with neural networks.* Cambridge University Press. Descargado de https://doi.org/10.1017%2F9781108860604 doi: 10 .1017/9781108860604

Minka, T. P. (2003). A comparison of numerical optimizers for logistic regression. *Unpublished draft*, 1–18.

Olvera-López, J., Carrasco-Ochoa, J., Martínez-Trinidad, J. F., y Kittler, J. (2010, 08). A review of instance selection methods. *Artif. Intell. Rev.*, *34*, 133-143. doi: 10.1007/s10462-010-9165-y

Opitz, J., y Burst, S. (2021). *Macro f1 and macro f1.*

Oussous, A., Benjelloun, F.-Z., Lahcen, A. A., y Belfkih, S. (2018). Big data technologies: A survey. *Journal of King Saud University-Computer and Information Sciences*, *30*(4), 431–448.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Ramírez-Gallego, S., Krawczyk, B., García, S., Woźniak, M., Benítez, J. M., y Herrera, F. (2017). Nearest neighbor classification for high-speed big data streams using spark. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *47*(10), 2727–2739.

Ritter, G., Woodruff, H., Lowry, S., y Isenhour, T. (1975). An algorithm for a selective nearest neighbor decision rule (corresp.). *IEEE Transactions on Information Theory*, *21*(6), 665–669.

Si, L., Yu, J., Li, S., Ma, J., Luo, L., Wu, Q., ... Liu, Z. (2017). Fcnn-mr: A parallel instance selection method based on fast condensed nearest neighbor rule. *Journal of information and communication convergence engineering*, *11*, 855-861.

Triguero, I., Peralta, D., Bacardit, J., García, S., y Herrera, F. (2015). Mrpr: A mapreduce solution for prototype reduction in big data classification. *Neurocomputing*, *150*, 331-345. Descargado de https://www.sciencedirect.com/science/article/pii/S0925231214013009 (Bioinspired and knowledge based techniques and applications The Vitality of Pattern Recognition and Image Analysis Data Stream Classification and Big Data Analytics) doi: https://doi.org/10.1016/j.neucom.2014.04.078

Vance, A. (2009). Hadoop, a free software program, finds uses beyond search. *New York Times*.

Vanschoren, J., van Rijn, J. N., Bischl, B., y Torgo, L. (2013). Openml: Networked science in machine learning. *SIGKDD Explorations*, *15*(2), 49–60. Descargado de http://doi.acm.org/10.1145/2641190.2641198 doi: 10.1145/

2641190.2641198

Varley, M., y Belle, V. (2019). *Fairness in machine learning with tractable models.* arXiv. Descargado de https://arxiv.org/abs/1905.07026 doi: 10.48550/ARXIV.1905.07026

Wainer, J., y Cawley, G. (2018). Nested cross-validation when selecting classifiers is overzealous for most practical applications. *arXiv preprint arXiv:1809.09446*.

Wilson, D. L. (1972). Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*(3), 408–421.

Wilson, D. R., y Martinez, T. R. (1997). Instance pruning techniques. En *Icml* (Vol. 97, pp. 400–411).

Wilson, D. R., y Martinez, T. R. (2000). Reduction techniques for instance-based learning algorithms. *Machine learning*, *38*(3), 257–286.

Yu, K., Ji, L., y Zhang, X. (2002, 04). Kernel nearest neighbor algorithm. *Neural Processing Letters*, *15*, 147-156. doi: 10.1023/A:1015244902967

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., ... Stoica, I. (2016, oct). Apache spark: A unified engine for big data processing. *Commun. ACM*, *59*(11), 56–65. Descargado de https://doi.org/10.1145/2934664 doi: 10.1145/2934664

Apéndice A

Métricas de rendimiento del modelo

A.1. Medidas de rendimiento para evaluar el modelo

Para realizar la comparación y poder evaluar de una manera justa el desempeño de los algoritmos consideraremos aspectos como: la capacidad de generalización de nuevas instancias, el tiempo de ejecución del algoritmo y la reducción del conjunto de datos original.

A.1.1. Capacidad de generalización

En la clasificación multiclase, usualmente un modelo de clasificación genera probabilidades de pertenencia a cada una de las K categorías, en donde la categoría con la probabilidad más alta es asignada a la unidad en observación, bajo tal situación los indicadores de desempeño son una buena opción para evaluar y comparar diferentes modelos de clasificación o técnicas de aprendizaje automático (Grandini, Bagli, y Visani, 2020).

En la experimentación haremos uso de dos métricas, exactitud y la medida F1 , esta última medida cobra relevancia ya que ciertos conjuntos de datos tienen clases que son asimétricas en cuanto al número de observaciones.

Exactitud

La exactitud es una métrica muy popular que se define como:

$$\frac{VP + VN}{TOTAL}$$

En ella se consideran la suma de los elementos Verdadero Positivo y Verdadero Negativo en el

numerador y la suma de todas las observaciones o Total en el denominador, lo que se devuelve es una medida donde a todos los individuos se les asigna el mismo peso por lo que la exactitud tiende a ocultar fuertes errores de clasificación para clases con pocas unidades, ya que dichas clases son menos relevantes(contienen menos individuos) en comparación con las más grandes.

Medida F1 y Macro F1

La métrica F1 considera precisión y sensibilidad como un promedio ponderado entre los dos, para el caso con 2 categorías tenemos la siguiente expresión:

$$MedidaF1 = 2\,\frac{P \times R}{P + R} = 2\,\frac{precisión \times sensibilidad}{precisión + sensibilidad} \tag{A.1}$$

Los valores de precisión y sensibilidad toman valores en el rango [0,1] y puesto que la media armónica tiende a dar más peso a los valores más bajos, en general F1 sufrirá una gran caída si el valor de la precisión o la sensibilidad es pequeño.(Grandini *et al.*, 2020).

Ahora bien, cuando tenemos más de dos clases, lo que se hace usualmente es calcular la métrica Macro F1, para ello se necesita calcular los elementos Macro-precisión y Macro-sensibilidad.

La métrica Macro F1 es computada como:

$$MACRO\ F_1 = F_1 = \frac{1}{K}\sum_k F1_k = \frac{1}{K}\sum_k \frac{2 \times P_k \times R_k}{P_k + R_k}$$

$$P_k = \frac{TP_k}{TP_k + FP_k}$$

$$R_k = \frac{TP_k}{TP_k + FN_k}$$

donde P_k y R_k corresponden a la precisión y sensibilidad respectivamente, el subíndice k hace referencia a la k-ésima clase.

La métrica se usa a menudo con la intención de asignar igual peso a las clases con más observaciones de las clases con menos observaciones, además cabe mencionar que lo expuesto en la ecuación (A.1) no es la única forma para calcular el valor macro F1, pero de acuerdo al análisis realizado por (Opitz y Burst, 2021), esta expresión podría considerarse más robusta.

A.1.2. Tiempo de ejecución del algoritmo

Algoritmos que se ejecuten a mayor velocidad son importantes, idealmente la mayor parte del tiempo de ejecución debería ser invertido en el entrenamiento del clasificador para después proceder a hacer predicciones.

En ese sentido, lo ideal en nuestra experimentación es reportar el tiempo de ejecución del algoritmo de SI además del tiempo dedicado a entrenar el clasificador y predecir las instancias sin etiqueta, la unidad de tiempo utilizada cada que se haga referencia al tiempo de ejecución serán los segundos (s).

A.1.3. Reducción del conjunto de datos original

La compresión (cpm) final de los datos cobra relevancia cuando el objetivo es obtener una gran reducción en la información, es análogo al porcentaje de reducción después de aplicar el algoritmo de selección de instancias y se define como:

$$cmp = 1 - \frac{||S||}{||T||}$$

En la expresión anterior, $\|S\|$ hace referencia al numero de instancias retenidas por el algoritmo y $\|T\|$ es el numero de instancias en el conjunto original. Es una métrica que no siempre es reportada en los trabajos de investigación pero ha sido utilizada en trabajos como el de Blachnik (2019) o Triguero *et al.* (2015).

A.2. Validación Cruzada

La mayoría de los algoritmos de aprendizaje maquina tienen uno o más hiperparámetros, por lo que seleccionar un algoritmo y ajustar hiperparámetros son problemas que van de la mano, uno querrá elegir tanto el algoritmo como sus hiperparámetros de tal manera que se maximice su rendimiento esperado en datos no observados (Wainer y Cawley, 2018).

La forma más básica de validación cruzada, conocida como validación cruzada *k-fold*, divide los datos disponibles en k fragmentos separados de aproximadamente el mismo tamaño. En cada iteración, se forma un conjunto de entrenamiento a partir de una combinación diferente de k-1 fragmentos, y el fragmento restante se utiliza como conjunto de prueba (Ver Figura A.1).

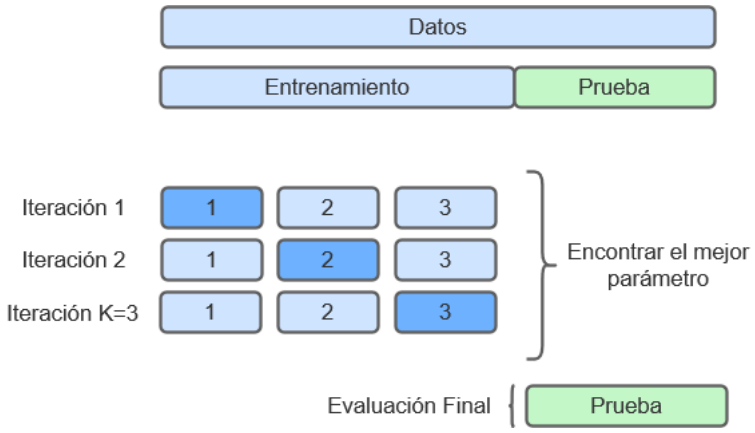

Figura A.1: 3-*fold CV*, Figura basada en (Pedregosa *et al.*, 2011)

Por las características de nuestro trabajo y retomando la experimentación reportada en nuestra revisión bibliográfica, nos limitaremos a una validación cruzada k=5, es decir se omitirá un conjunto de prueba final.

Apéndice B

Modelación post-selección de instancias

B.0.1. K VECINOS CERCANOS (KNN)

La regla del algoritmo consiste en asignar a un punto no clasificado de la muestra, la etiqueta de los puntos más cercanos del conjunto de entrenamiento (Cover y Hart, 1967), la función de distancia utilizada para identificar los k vecinos puede ser calculada de muchas maneras, pero es común utilizar la distancia Euclidiana.

La regla no requiere una fase de entrenamiento (todo el conjunto de entrenamiento se trata como un conjunto de referencia en la siguiente fase de clasificación) por tal motivo ciertos autores le llaman un algoritmo de aprendizaje perezoso o de aprendizaje lento (González, 2018).

Es uno de los métodos más populares en aprendizaje supervisado, el requisito de memoria y la complejidad del algoritmo son limitaciones que impactan en el tiempo de ejecución, ya que depende de cada ejemplo del conjunto de entrenamiento, en contraparte el clasificador tiene a K como único parámetro a calibrar, donde K es el número de vecinos más cercanos a considerar para asignar la etiqueta (Hassanat, Abbadi, Altarawneh, y Alhasanat, 2014). El clasificador también tiene una gran flexibilidad para ser usado en muchas tareas además de la clasificación, por lo que podemos encontrar una gran variedad de propuestas del algoritmo original tales como: *Weighted* KNN, *Condensed* NN, *Reduced* NN, *Generalized* NN, *K-d tree*, *ball tree*, *Principal Axis Tree* y *Nearest Feature Line*, otras propuestas y un análisis más detallado de estas modificaciones se pueden consultar en (Bhatia y Vandana, 2010).

B.0.2. REGRESIÓN LOGÍSTICA

La regresión logística establece un modelo probabilístico sobre la fuerza de conexión entre covariables y la respuesta esperada, desde el punto de vista estadístico pertenece a los llamados modelos lineales generalizados. Suponiendo que tenemos dos clases que llamaremos 0 y 1, para un punto de prueba, la probabilidad de que dicho punto esté en la clase 1 es dada por la expresión:

$$P(y_i = 1) = \frac{1}{1 + exp(-(\mathbf{w^t x} + b)}$$

Los parámetros de peso w se entrenan utilizando un procedimiento de descenso de gradiente para minimizar alguna función de pérdida como lo puede ser el error cuadrado medio (Varley y Belle, 2019). La clasificación del i-ésimo sujeto se puede realizar de acuerdo con la regla más simple que es: asignar al i-ésimo sujeto al grupo 1 si

$$P(y_i = 1) > P(y_i = 0)$$

donde la expresión anterior es equivalente a $P(y_i) > 1/2$, donde el valor $1/2$ puede ser ajustado dependiendo del sesgo y el error de estimación (Chung, 2020).

La complejidad de los modelos de regresión logística al igual que la mayoría de los algoritmos de aprendizaje no es estática y dependerá de muchos factores, en la regresión logística destacamos el tipo de optimizador y el numero de iteraciones necesarias para converger al momento de encontrar los pesos óptimos, los algoritmos *Quasi Newton* y 'BFGS' destacan por su velocidad, tienen una complejidad de $O(d^2 + nd)$, donde d es el numero de atributos y n el numero de observaciones, la complejidad de otros optimizadores se puede consultar en (Minka, 2003).

B.0.3. BOSQUES ALEATORIOS

De acuerdo con (Hang, Liu, y Steinwart, 2019), la tecnica de bosques aleatorios fue propuesta por Breiman a principios de la década de los 2000 y se encuentra en el *top* de los métodos más exitosos que se aplican actualmente para tratar una amplia gama de problemas de clasificacion.

El clasificador consiste en construir un conjunto (o bosque) de árboles de decisión desarrollados a partir de una variante aleatoria, la dificultad consiste en inyectar aleatoriedad mientras se minimiza la varianza y al mismo tiempo se mantiene un sesgo bajo en los árboles de decisión, propuestas que llevan por nombre, *Perfect Random Tree Ensembles (PERT)*, *Extremely Randomized Trees*(ETs) y *Rotation Forest* por mencionar ejemplos, son parte de la familia de Bosques Aleatorios y la principal diferencia entre ellos es la forma en que introducen perturbaciones aleatorias en los árboles de decisión (Louppe, 2015).

Un solo árbol de decisión puede ser propenso a sobreajustarse y ser sensible a pequeños cambios

en los datos de entrada, estos efectos se pueden mitigar de cierta manera combinando múltiples árboles de decisión en un conjunto, donde la clasificación final se determina por un voto mayoritario de los muchos árboles contenidos en el modelo, esto es posible porque suelen tener un sesgo bajo y una varianza alta, lo que hace que sea muy probable que se beneficien del proceso de promediar. (Markel y Bayless, 2020).

Sea K el numero de atributos, N el numero de observaciones, la complejidad computacional estudiada y derivada por Louppe (2015) nos indica que en el mejor de los casos, la complejidad es lineal con el número de variables (es decir, $O(K)$) y lineal o cuasilineal con el número de muestras (es decir, $O(N)log(N)$ o $O(Nlog^2N)$). En el peor de los casos, esta dependencia posterior se vuelve cuadrática (es decir, $O(N^2)$ u $O(N^2log(N))$, lo que podría afectar a grandes conjuntos de datos, el análisis del caso promedio muestra que los casos pesimistas no son dominantes y que en promedio, para todos los métodos, la complejidad se comporta como en el mejor de los casos.

B.0.4. REDES NEURONALES

El Perceptron Multi Capa (MLP, por sus siglas en ingles) es una red neuronal *feed-forward* se suele utilizar para la clasificación, típicamente consta de una capa de entrada que representa los atributos de entrada del problema y una capa de salida que representa los posibles resultados y una o más capas ocultas entre ellos. En esta red, las neuronas están completamente conectadas entre sí mediante la conexión de pesos y el conocimiento obtenido mediante el aprendizaje se almacena encontrando los valores óptimos para los pesos de cada neurona, cabe mencionar que las redes neuronales son capaces de resolver problemas linealmente no separables.

Para encontrar los pesos óptimos se suele utilizar la retro-propagación que es un algoritmo eficiente para el descenso de gradiente estocástico. Las redes con muchas neuronas ocultas tienen muchos parámetros libres (pesos y umbrales). Esto aumenta el riesgo de sobreentrenar, lo que reduce el poder de generalización de la red. Las redes profundas con muchas capas ocultas son particularmente propensas al sobreajuste (Mehlig, 2021).

Cuando se habla de la complejidad computacional en las redes neuronales hay una gran cantidad de elementos a considerar, por ejemplo, el optimizador utilizado, el tamaño del lote en cada iteración, el numero de neuronas por capa, el numero de capas ocultas, el numero de observaciones y la dimensión de cada observación son ejemplos. Para tener una idea de la complejidad, pensemos en una red neuronal *feedforward* con n entradas y una única salida, bajo las condiciones enunciadas retomamos la siguiente tabla.

Entradas	Capas Intermedias	Función de Activación	Complejidad
n	1	arctan	$O((n+h)^{n+2})$
n	l	arctan	$2^{h(2h-1)}O(nl+n)^{n+2h}$

Tabla B.1: Limite superior estimado para redes con h neuronas, n entradas y l capas ocultas

Los detalles relacionados al diseña de la métrica de complejidad utilizada, además de la complejidad en otras funciones de activación se pueden consultar en (Bianchini y Scarselli, 2014)